Trabalhando com Flutter

Atividade - 01

Para começar seu primeiro projeto Flutter, siga estes passos:

Instale o Flutter: Certifique-se de ter o Flutter instalado e configurado corretamente no seu sistema.

Abra o terminal: Abra o terminal e navegue até o diretório onde deseja criar seu projeto.

Crie o projeto: Use o comando flutter create <nome_do_projeto> para criar um novo projeto.

Abra o projeto no IDE: Abra o projeto no seu IDE preferido (como VS Code, Android Studio, etc.).

Execute o projeto: Use o comando flutter run para executar o projeto e ver o aplicativo em execução.

Personalize o aplicativo: Comece a editar os arquivos do projeto para personalizar a aparência e o comportamento do seu aplicativo.

Passos detalhados:

1. Instalação e configuração:

Baixe e instale o Flutter SDK de acordo com as instruções no site do Flutter.

Configure o Flutter SDK, incluindo a variável de ambiente PATH.

Instale as dependências necessárias (ex: Android Studio, emulador Android).

2. Criação do projeto:

Abra o terminal e navegue até o diretório onde deseja criar seu projeto.

Use o comando flutter create <nome_do_projeto> (substituindo <nome_do_projeto> pelo nome desejado).

3. Abrir o projeto:

Use seu IDE favorito (ex: VS Code, Android Studio) para abrir o projeto criado.

4. Execução do projeto:

No terminal, navegue para o diretório do projeto.

Execute o comando flutter run para iniciar o aplicativo.

5. Personalização:

O Flutter usa a linguagem Dart.

Explore os arquivos do projeto (principalmente lib/main.dart) para entender a estrutura do aplicativo.

Modifique o código para personalizar a interface, funcionalidades e recursos do seu aplicativo.

Dicas:

Use o comando flutter doctor para verificar se o Flutter está configurado corretamente.

Considere usar o comando flutter pub add <pacote_nome> para adicionar dependências de pacotes ao seu projeto.

Explore a documentação do Flutter e a comunidade online para aprender mais sobre o framework.

Atividade - 02

Código VS

#

Para criar um aplicativo Flutter com o VS Code e outros editores Code OSS, primeiro você precisa instalar o Flutter e configurar o VS Code para desenvolvimento Flutter. Em seguida, siga estes passos:

1. Inicie o VS Code

Abra o VS Code ou seu editor Code OSS preferido.

2. Abra a paleta de comandos

Vá em **Exibir** > **Paleta de Comandos** ou pressione Control+ Shift+ P.

3. Encontre os comandos do Flutter

Na paleta de comandos, comece a digitar flutter:. O VS Code deve exibir comandos do plugin Flutter.

4. Execute o novo comando do projeto

Selecione o comando **Flutter: Novo Projeto**. Seu sistema operacional ou VS Code pode solicitar acesso aos seus documentos. Concorde em prosseguir para a próxima etapa.

5. Escolha um modelo

O VS Code deverá perguntar " **Qual modelo do Flutter?"**. Dependendo do tipo de projeto do Flutter que você deseja criar, escolha o modelo correspondente. Para um novo aplicativo do Flutter, escolha **"Aplicativo"**.

6. Selecione um local para o projeto

Uma caixa de diálogo de arquivo será exibida. Selecione ou crie o diretório pai onde o projeto será criado. Não crie a pasta do projeto em si, a ferramenta Flutter o fará. Para confirmar sua seleção, clique em " **Selecionar uma pasta para criar o projeto"**.

7. Digite um nome de projeto

O VS Code solicitará que você insira um nome para o seu novo projeto. Insira um nome para o seu aplicativo que siga a lowercase_with_underscoresconvenção de nomenclatura, seguindo as diretrizes do Effective Dart. Para confirmar sua seleção, pressione Enter.

8. Aguarde a inicialização do projeto

Com base nas informações inseridas, o VS Code usará flutter createo bootstrap para inicializar seu aplicativo. O progresso geralmente é exibido como uma notificação no canto inferior direito e também pode ser acessado no painel **Saída**.

9. Execute seu aplicativo

Seu novo aplicativo agora deve estar criado e aberto no VS Code. Para testar seu novo aplicativo, siga os passos para executar e depurar no VS Code.

Você criou com sucesso um novo aplicativo Flutter no VS Code! Se precisar de mais ajuda para desenvolver o Flutter no VS Code, confira a referência do VS Code para Flutter .

Estúdio Android

#

Para criar um aplicativo Flutter com o Android Studio, primeiro você precisa instalar o Flutter e configurar o Android Studio para desenvolvimento com Flutter. Em seguida, siga estes passos:

1. Inicie o Android Studio

Abra o Android Studio com os plugins Dart e Flutter instalados.

2. Iniciar a criação do projeto

Se você estiver na caixa de diálogo de boas-vindas do IDE que diz **Bem-vindo ao Android Studio**, localize e clique no botão **Novo projeto Flutter** no centro.

Se você já tiver um projeto aberto, feche-o ou vá em **Arquivo** > **Novo** > **Novo Projeto Flutter...** .

3. Escolha um tipo de projeto

Na caixa de diálogo **Novo Projeto**, em **Geradores** no painel esquerdo, selecione **Flutter** .

4. Verificar a configuração do Flutter SDK

Na parte superior do painel direito, certifique-se de que o valor **do caminho do Flutter SDK** corresponda ao local do Flutter SDK com o qual você deseja desenvolver. Caso contrário, atualize-o escolhendo ou especificando o caminho correto.

5. Configure seu projeto

Clique em **Avançar** para continuar a configuração do projeto. Diversas opções de configuração deverão aparecer.

No campo **Nome do projeto** , insira um nome para seu aplicativo que siga a lowercase_with_underscoresconvenção de nomenclatura, seguindo as diretrizes do Effective Dart .

Se você não estiver criando um aplicativo, selecione outro modelo no menu suspenso **Tipo de projeto**.

Se você estiver criando um aplicativo que poderá ser publicado no futuro, defina o campo **Organização** como o domínio da sua empresa.

Os outros campos podem ser mantidos como estão ou configurados de acordo com as necessidades do seu projeto.

6. Concluir a criação do projeto

Depois de concluir a configuração do seu projeto, clique em **Criar** para iniciar a inicialização do projeto.

7. Aguarde a inicialização do espaço de trabalho

O Android Studio agora inicializará seu espaço de trabalho, inicializará a estrutura dos arquivos do projeto e recuperará as dependências do aplicativo. Isso pode demorar um pouco e pode ser monitorado na parte inferior da janela.

8. Execute seu aplicativo

Seu novo aplicativo agora deve estar criado e aberto no Android Studio. Para testar seu novo aplicativo, siga os passos para executar e depurar no Android Studio.

Você criou com sucesso um novo aplicativo Flutter no Android Studio! Se precisar de mais ajuda para desenvolver o Flutter no Android Studio, confira a referência do Android Studio para Flutter.

IntelliJ

#

Para criar um aplicativo Flutter com o IntelliJ ou outras IDEs da JetBrains, primeiro você precisa instalar o Flutter e configurar o IntelliJ para desenvolvimento com Flutter. Em seguida, siga estes passos:

1. Inicie o IntelliJ

Abra o IntelliJ IDEA ou seu IDE preferido baseado em IntelliJ da JetBrains que tenha os plugins Dart e Flutter instalados.

2. Iniciar a criação do projeto

Se você estiver na caixa de diálogo de boas-vindas do IDE que diz **Bem-vindo ao IntelliJ IDEA**, localize e clique no botão **Novo projeto** no canto superior direito.

Se você já tiver um projeto aberto, feche-o ou vá em **Arquivo** > **Novo** > **Novo Projeto...** .

3. Escolha um tipo de projeto

Na caixa de diálogo **Novo Projeto**, em **Geradores** no painel esquerdo, selecione **Flutter** .

4. Verificar a configuração do Flutter SDK

Na parte superior do painel direito, certifique-se de que o valor **do caminho do Flutter SDK** corresponda ao local do Flutter SDK com o qual você deseja desenvolver. Caso contrário, atualize-o escolhendo ou especificando o caminho correto.

5. Configure seu projeto

Clique em **Avançar** para continuar a configuração do projeto. Diversas opções de configuração deverão aparecer.

No campo **Nome do projeto**, insira um nome para seu aplicativo que siga a lowercase_with_underscoresconvenção de nomenclatura, seguindo as diretrizes do Effective Dart .

Se você não estiver criando um aplicativo, selecione outro modelo no menu suspenso **Tipo de projeto**.

Se você estiver criando um aplicativo que poderá ser publicado no futuro, defina o campo **Organização** [para o domínio da sua empresa][ij-set-org].

Os outros campos podem ser mantidos como estão ou configurados de acordo com as necessidades do seu projeto.

6. Concluir a criação do projeto

Depois de concluir a configuração do seu projeto, clique em **Criar** para iniciar a inicialização do projeto.

7. Aguarde a inicialização do espaço de trabalho

O IntelliJ agora inicializará seu espaço de trabalho, inicializará a estrutura de arquivos do seu projeto e recuperará as dependências do seu aplicativo. Isso pode demorar um pouco e pode ser monitorado na parte inferior da janela.

8. Execute seu aplicativo

Seu novo aplicativo agora deve estar criado e aberto no IntelliJ. Para testar seu novo aplicativo, siga os passos para executar e depurar no IntelliJ.

Você criou com sucesso um novo aplicativo Flutter no IntelliJ! Se precisar de mais ajuda para desenvolver o Flutter no IntelliJ, confira a referência do IntelliJ para Flutter .

Firebase Studio

#

Para criar um aplicativo Flutter com o Firebase Studio , primeiro você precisa de uma conta do Google e configurar o Firebase Studio . Depois, siga estes passos:

1. Inicie o Firebase Studio

No seu navegador preferido, acesse o painel do Firebase Studio em studio.firebase.google.com/. Caso ainda não tenha feito isso, talvez seja necessário fazer login na sua conta do Google.

2. Criar um novo espaço de trabalho

No painel do Firebase Studio, encontre a seção **"Começar a programar um aplicativo"**. Ela deve incluir uma variedade de modelos para você escolher. Selecione o modelo **Flutter** . Se não conseguir encontrá-lo, ele pode estar na categoria **"Mobile"**.

3. Dê um nome ao seu espaço de trabalho

O Firebase Studio solicitará que você **nomeie seu workspace**. Esse nome é diferente do nome do seu aplicativo Flutter. Escolha um nome descritivo que você reconheça em uma lista de seus workspaces.

4. Provisione seu novo espaço de trabalho

Depois de escolher um nome e configurar seu espaço de trabalho, clique em **Criar** para provisionar seu novo espaço de trabalho.

5. Aguarde a inicialização do espaço de trabalho

O Firebase Studio agora inicializará seu espaço de trabalho, inicializará a estrutura de arquivos do seu projeto e recuperará as dependências do seu aplicativo. Isso pode demorar um pouco.

6. Execute seu aplicativo

Seu novo aplicativo agora deve ser criado e aberto no editor do Firebase Studio. Para testar seu novo aplicativo, siga a documentação fornecida pelo Firebase Studio para visualizá-lo na web ou no Android.

Você criou com sucesso um novo aplicativo Flutter no Firebase Studio! Se precisar de ajuda para configurar seu espaço de trabalho, confira Personalizar seu espaço de trabalho do Firebase Studio.

terminal

#

Para criar um aplicativo Flutter no seu terminal, primeiro você precisa instalar e configurar o Flutter . Depois, siga estes passos:

1. Abra seu terminal

Abra seu método preferido para acessar a linha de comando, como Terminal no macOS ou PowerShell no Windows.

2. Navegue até o diretório desejado

Certifique-se de que seu diretório de trabalho atual seja o diretório pai desejado para seu novo aplicativo. Não crie a pasta do projeto, a flutterferramenta o fará.

3. Configurar a criação do projeto

No seu terminal, digite o flutter createcomando e passe as flags e opções desejadas para configurar o seu projeto. Por exemplo, para criar um aplicativo com um main.dartarquivo mínimo, você pode adicionar a --emptyopção:

flutter create --empty

cópia_do_conteúdo

Para saber mais sobre as opções de criação disponíveis, execute flutter create --helpem outra janela de terminal.

4. Digite um nome de projeto

Como único argumento não opcional para flutter create, especifique o diretório e o nome padrão do seu aplicativo. O nome deve seguir a lowercase_with_underscoresconvenção de nomenclatura, seguindo as diretrizes do Effective Dart .

Por exemplo, se você quisesse criar um aplicativo chamado my_app:

flutter create my_app

cópia_do_conteúdo

5. Execute o comando configurado

Para criar um projeto com a configuração especificada, execute o comando criado na etapa anterior.

6. Aguarde a inicialização do projeto

A flutterferramenta agora inicializará a estrutura de arquivos do seu projeto e recuperará todas as dependências necessárias. Isso pode demorar um pouco.

7. Navegue até o diretório do projeto

Agora que seu projeto foi criado, você pode navegar até ele no seu terminal ou no editor de sua preferência. Por exemplo, com um shell bash e um projeto chamado my_app:

cd my_app

cópia_do_conteúdo

8. Execute seu aplicativo

Para testar seu novo aplicativo, execute o flutter runcomando no seu terminal e responda aos prompts para selecionar um dispositivo de saída.

Você criou com sucesso um novo aplicativo Flutter no seu terminal! Se precisar de ajuda para configurar seu projeto ou com a flutterferramenta CLI, consulte a referência da CLI do Flutter .

Atividade - 03

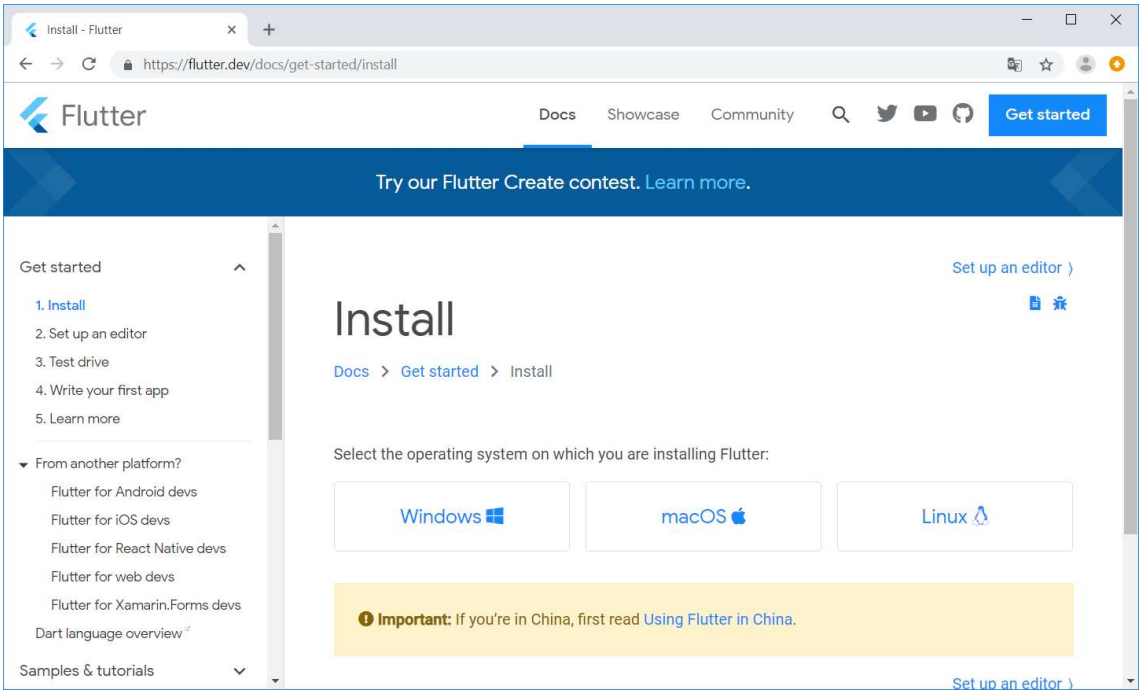
1. Na página de download do **Flutter SDK**, clique no botão “Windows”.
2. 

Figura 1. Página com as opções de download do Flutter SDK.

3. Na sessão “Get the Flutter SDK”, logo abaixo dos requerimentos do sistema, clique no botão flutter_windows_v1.0.0-stable.zip.

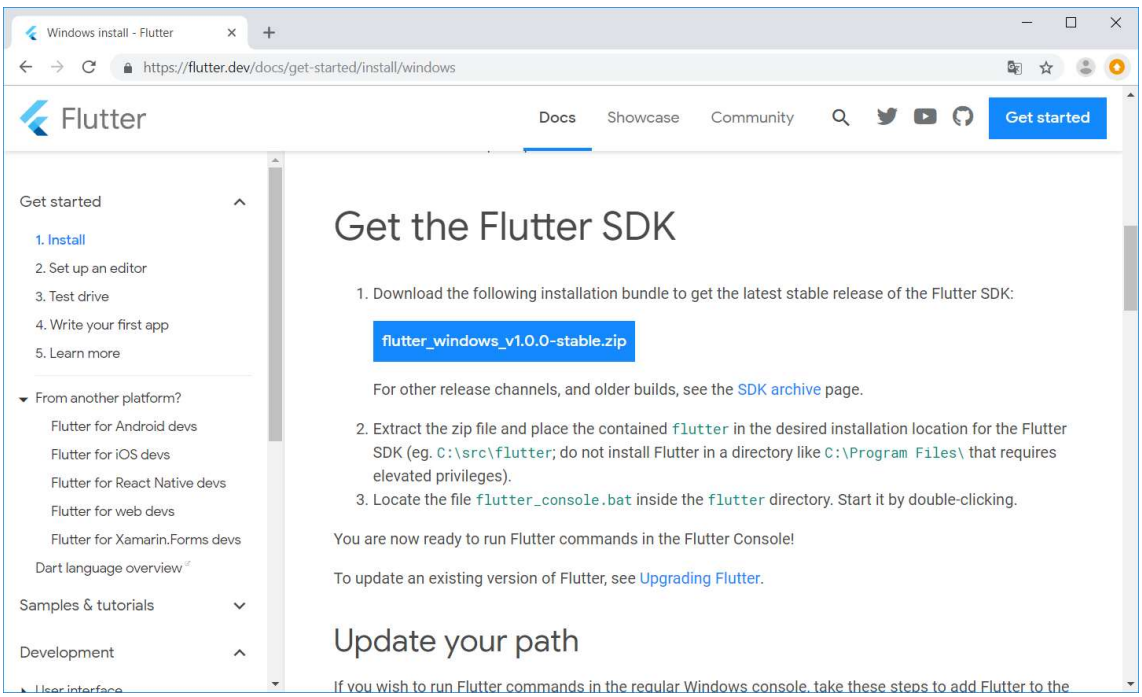


Figura 2. Página de download do Flutter SDK para Windows.

4. Na partição C:/ do seu HD, crie um diretório chamado src/ e extraia os arquivos do SDK nesse diretório:

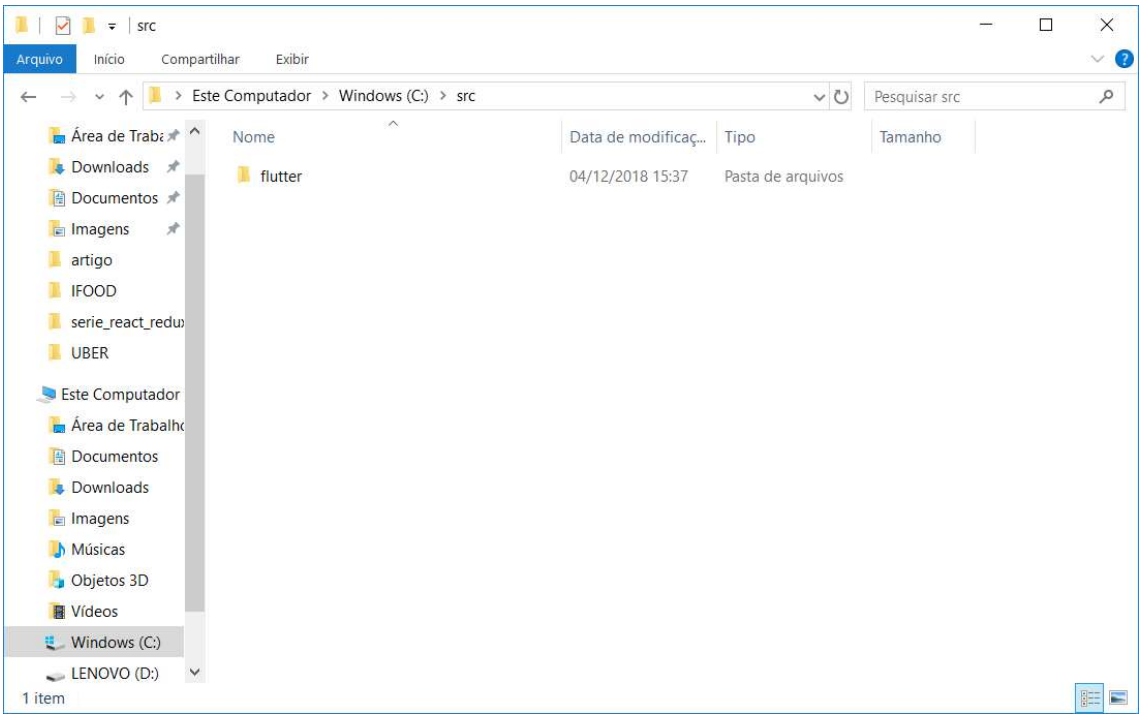


Figura 3. Diretório C:/src/

Agora que realizamos o download do SDK, criaremos uma variável de ambiente do Windows para sermos capazes de utilizar a ferramenta de linha de comando do Flutter através do terminal do Windows:

1. Na barra de busca do Windows pesquise por "Editar as variáveis de ambiente do sistema” e clique no primeiro resultado:

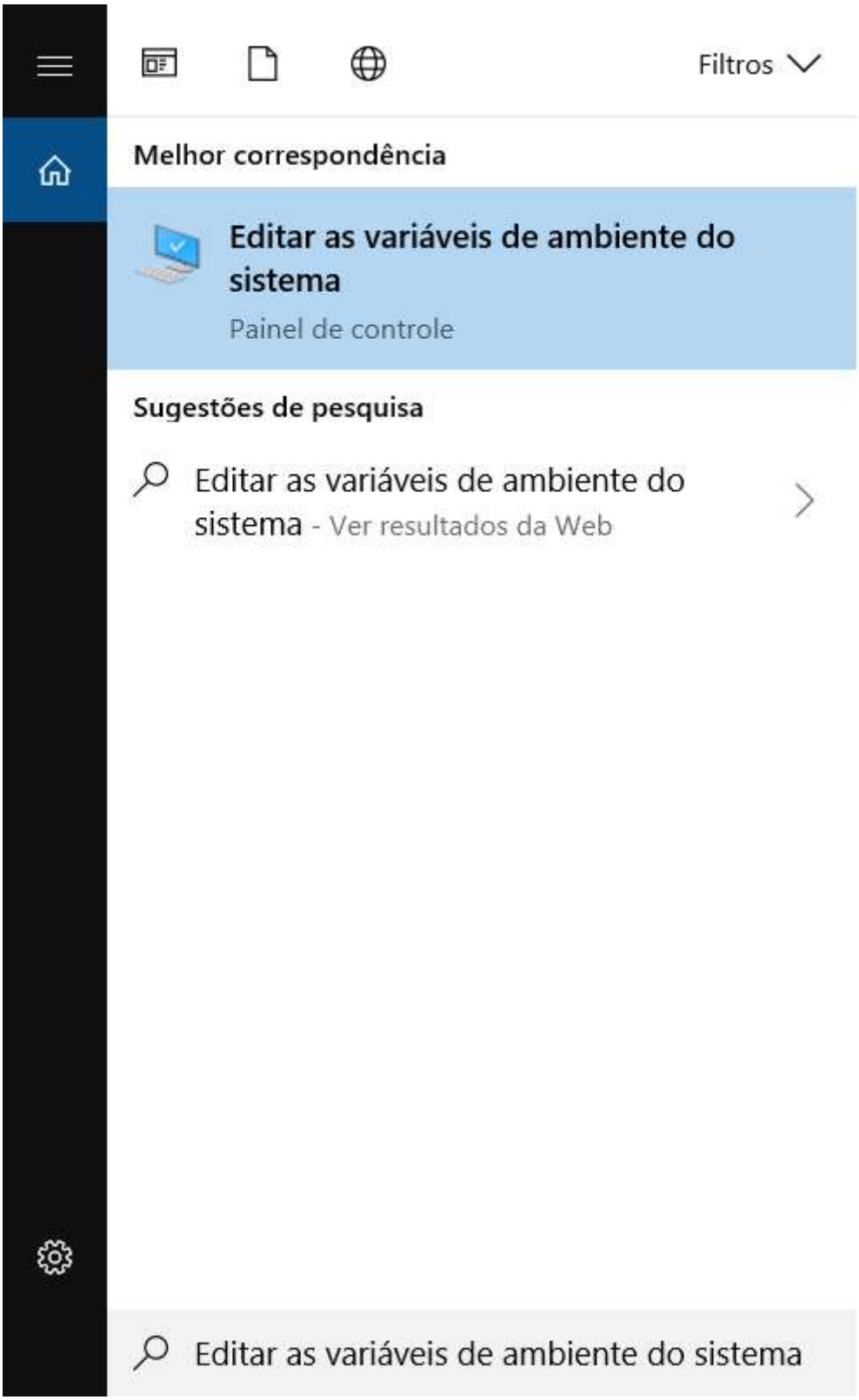


Figura 4. Barra de busca do Windows.

2. Na janela que abrir, clique no botão “Variáveis de Ambiente...”:

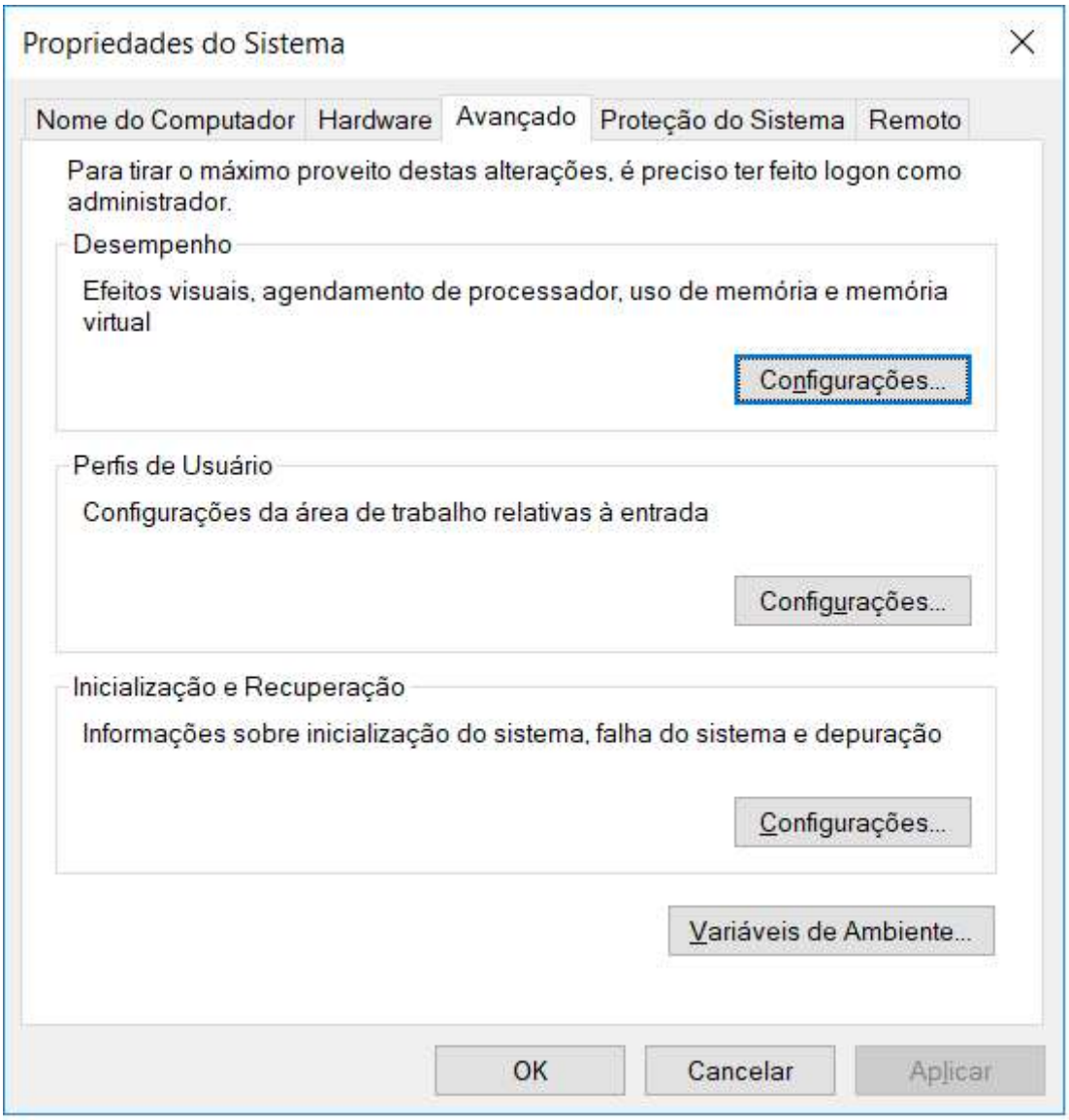


Figura 5. Janela de propriedades do sistema no Windows 10.

3. No campo superior, chamado “Variáveis de usuário...”, clique na variável “Path” e, então, no botão “Editar”:

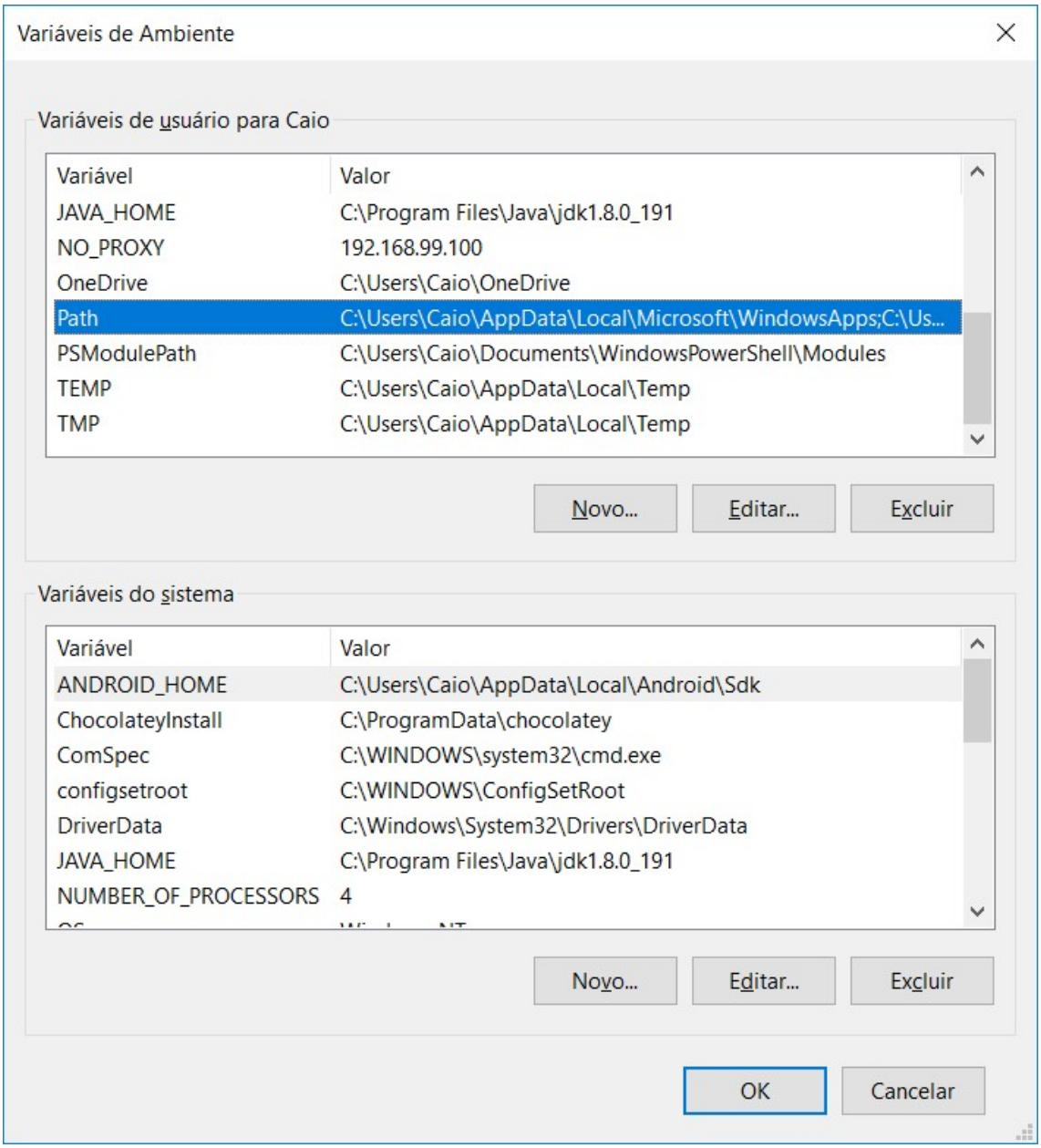


Figura 6. Variáveis de ambiente do sistema.

4. Na janela que abrir encontraremos uma lista de diretórios. Clique em “Novo” e adicione o endereço do diretório C:/src/flutter/bin do SDKe clique em “OK”:

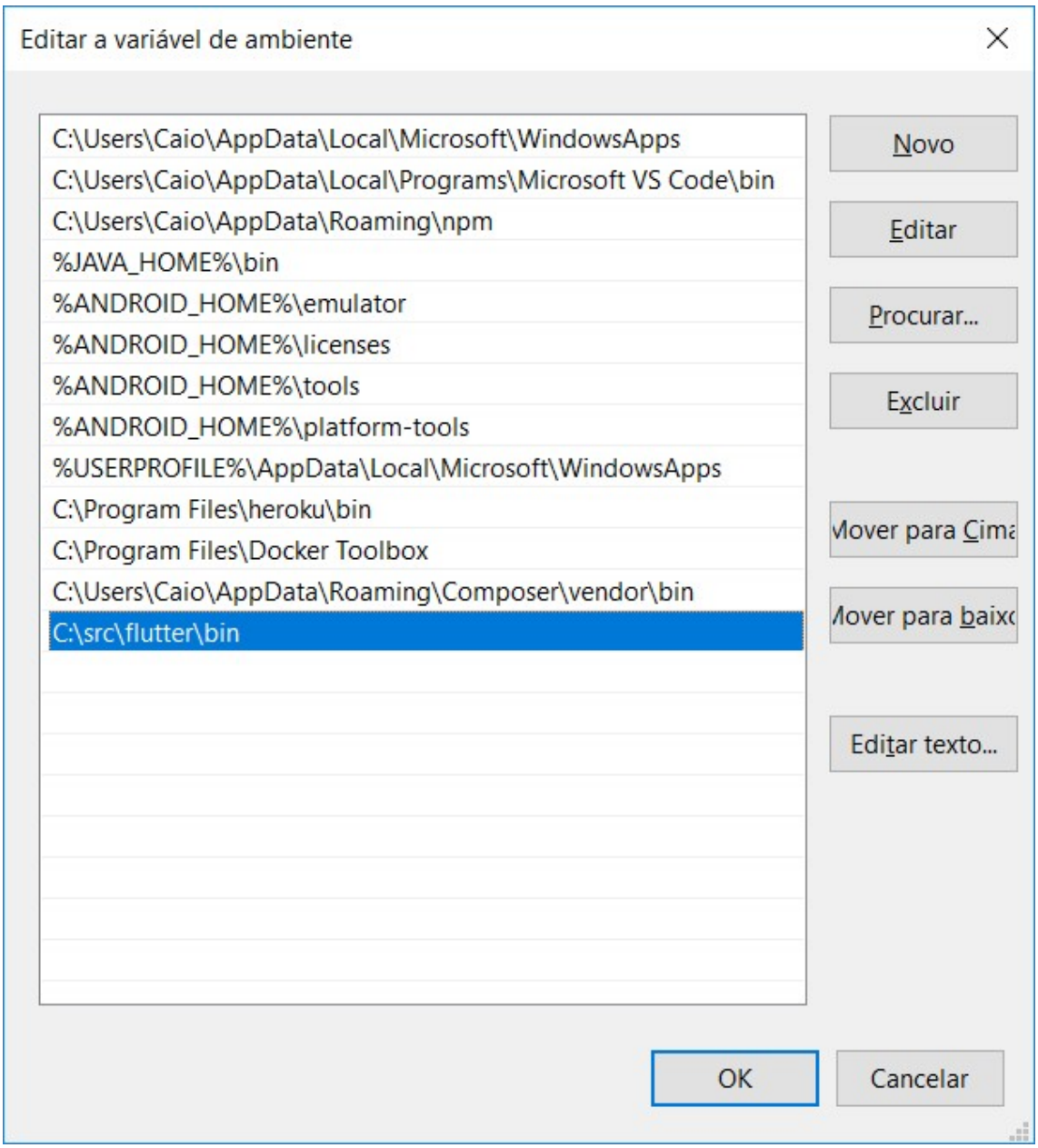


Figura 7. Caminho do Flutter SDK adicionado a variável de ambiente Path.

5. Para verificar se o SDK foi instalado corretamente, abra uma sessão do CMD e execute o seguinte comando:

flutter doctor

Se tudo estiver instalado corretamente, deverá receber uma resposta similar a essa:

Doctor summary (to see all details, run flutter doctor -v):

[√] Flutter (Channel beta, v1.1.8, on Microsoft Windows

[versão 10.0.17134.590], locale pt-BR)

[!] Android toolchain - develop for Android devices (Android SDK version 28.0.3)

! Some Android licenses not accepted. To resolve this,

run: flutter doctor --android-licenses

[√] Android Studio (version 3.2)

[√] VS Code (version 1.31.1)

[!] Connected device

! No devices available

Pronto! Agora que já possuímos o SDK, estamos prontos para configurar nossa IDE. Nesse artigo utilizaremos o Visual Studio Code, mas se quiser também pode utilizar o Android Studio (basta ter memória RAM suficiente):

1. Com o Visual Studio Code aberto, no menu lateral, selecione a aba “extensões”:

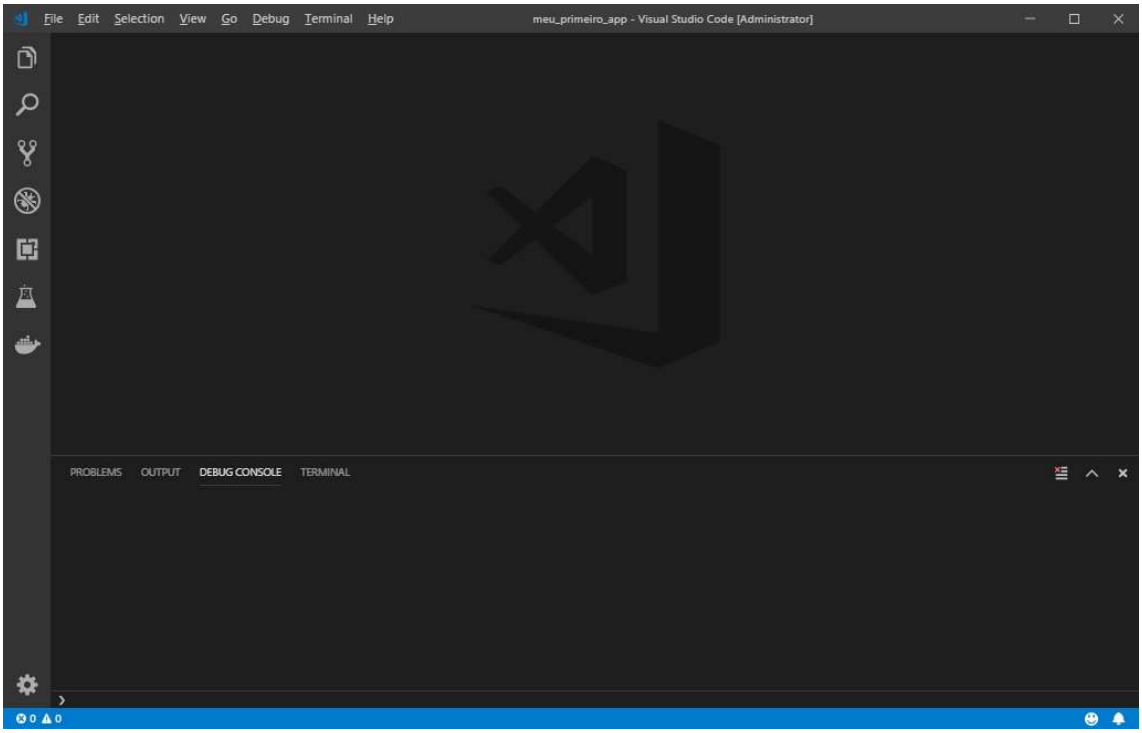


Figura 8. Botão de extensões do Visual Studio Code

2. No campo de busca, digite “Flutter”. O primeiro resultado da busca será o plugin oficial. Clique no pequeno botão “Install” verde. O plugin já contém tudo que precisamos para executar e debugar nosso código Dart.

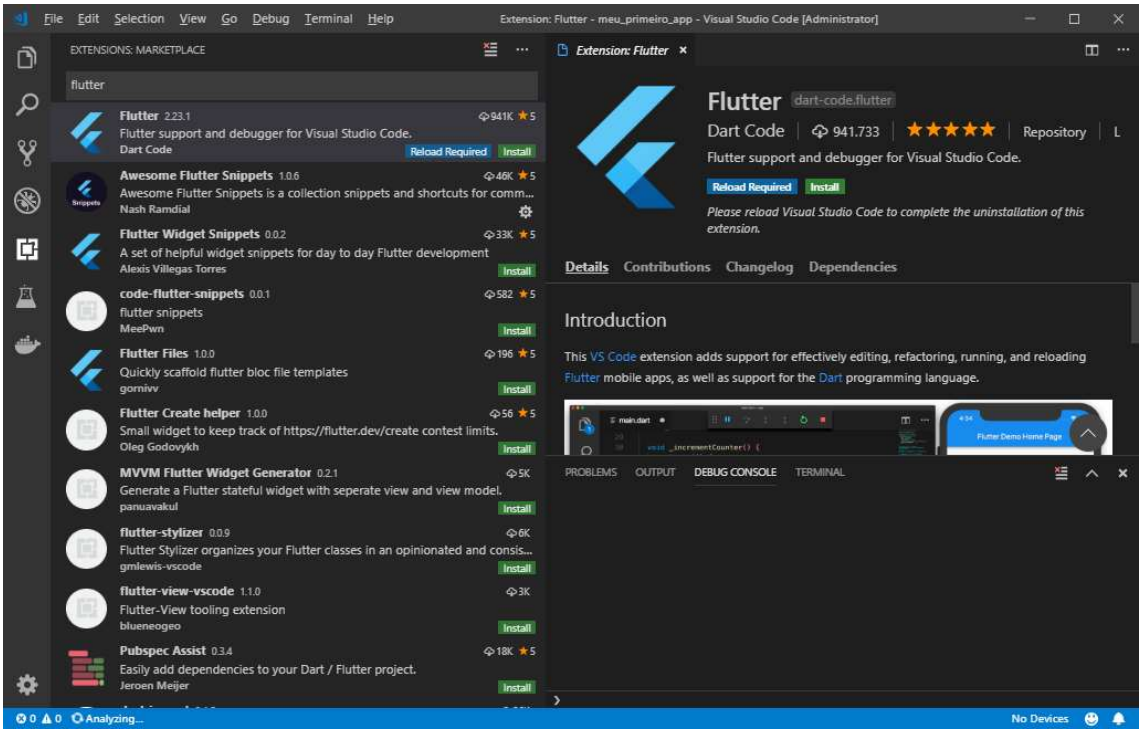


Figura 9. Plugin oficial do Flutter para Visual Studio Code

Pronto! Finalmente podemos criar nosso primeiro aplicativo em Flutter!

Criando um aplicativo com Flutter

Com o SDK instalado e adicionado à variável de ambiente “Path”, criar um novo projeto Flutter é extremamente simples. Para isso, no terminal do Windows, navegue até o diretório onde deseja iniciar o seu projeto e execute o seguinte comando:

flutter create meu_primeiro_app

O parâmetro meu_primeiro_app é o nome da nossa aplicação. Sinta-se livre para colocar o nome que quiser.

O Flutter ficará encarregado de criar todos os arquivos da nossa aplicação. Feito isso, ainda no terminal, navegue até o diretório que foi criado pelo CLI:

cd meu_primeiro_app

Para abrir o projeto no Visual Studio Code basta executar o comando:

code .

Executando o aplicativo Flutter em um Emulador Android

O Android Studio, além do Android SDK, também nos fornece um emulador Android que contém todas as suas versões. Dessa forma, podemos testar o comportamento do nosso aplicativo em diversos dispositivos, ao mesmo tempo que elimina a necessidade de conectarmos um dispositivo físico para realizar testes, embora essa opção ainda seja possível.

No Visual Studio Code, abra o arquivo lib/main.dart. Ele será o ponto de entrada do nosso aplicativo, ou seja, toda aplicação será iniciada através da função main no início do arquivo. Todo o código que você vê abaixo dessa função é o aplicativo que o Flutter oferece como exemplo. Por enquanto não realizaremos nenhuma modificação, apenas executaremos este aplicativo. Para isso, com o arquivo aberto, pressione:

CRTL + F5

No topo superior do Visual Studio Code abrirá uma pequena aba. Se você já possui emuladores configurados, eles devem aparecer nessa aba. Se não, basta selecionar a opção “Create New”:

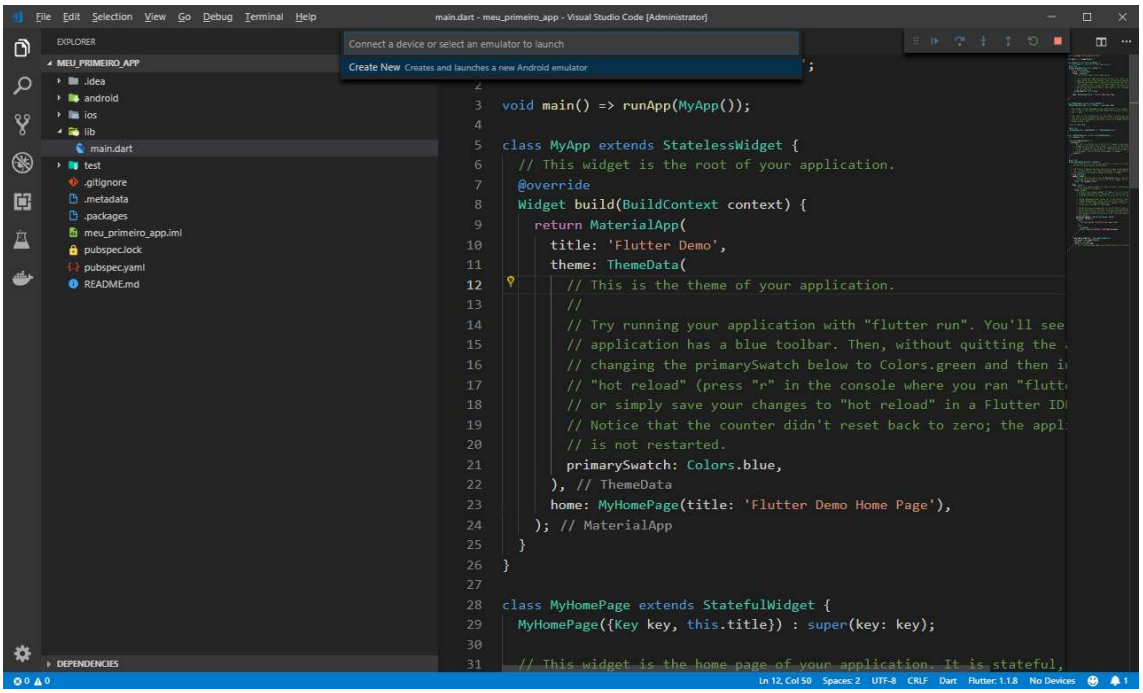


Figura 10. Executando o emulador Android a partir do Visual Studio Code.

O processo deve demorar alguns minutos, e você pode acompanhá-lo através da pequena aba aberta no canto inferior direito:

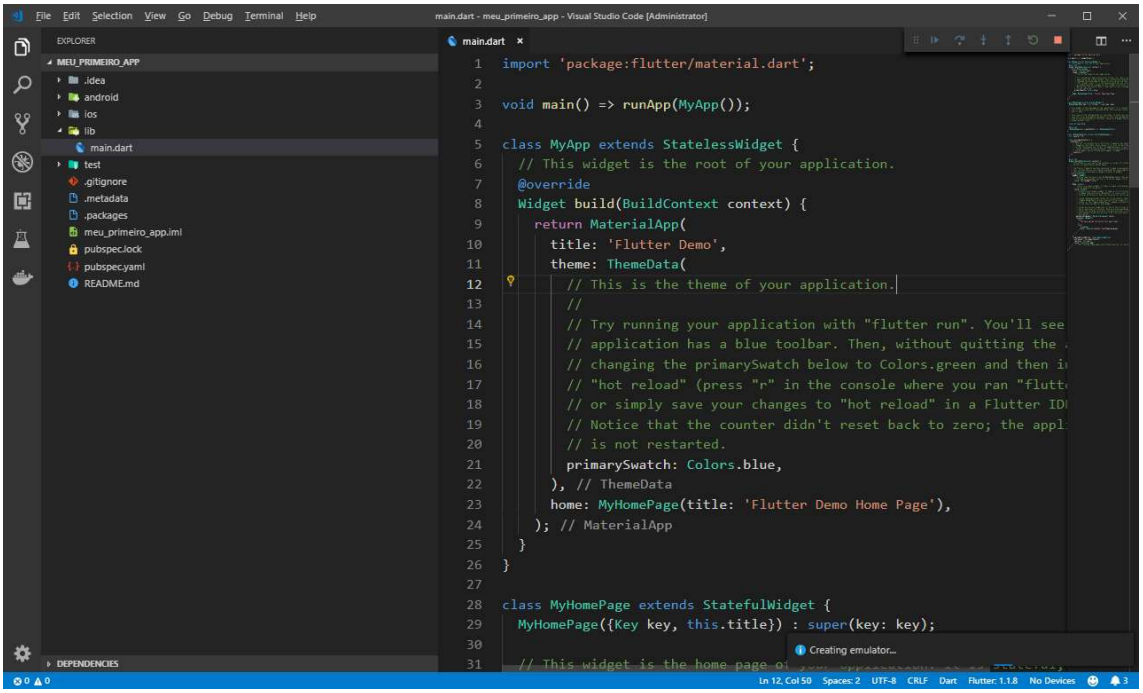


Figura 11. Processo de carregamento do emulador.

Ao final do processo, quando o emulador for automaticamente aberto, pode ser que seu aplicativo não seja executado. Dessa forma, no Visual Studio Code, pressionamos “CRTL + F5” novamente. Entretanto, como o emulador já estará aberto, não será necessário escolher nenhuma opção. O Flutter ficará responsável por iniciar a aplicação no emulador e emitir todos os eventos e erros através da aba “DEBUG CONSOLE” no rodapé do Visual Studio Code:

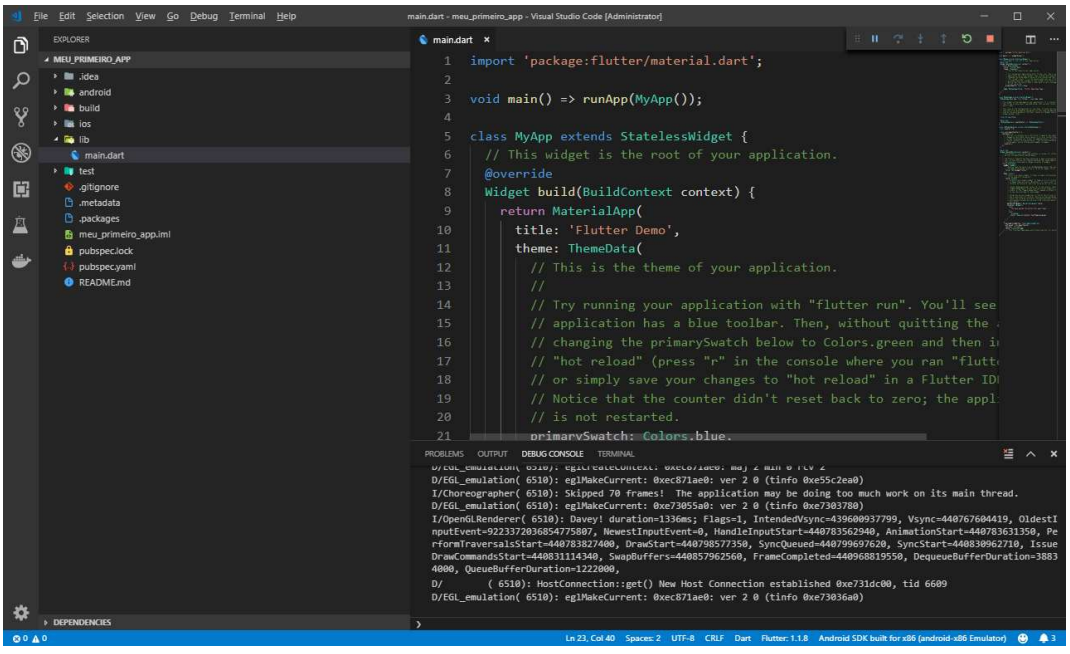


Figura 12. Debug Console do Flutter.

Nesse momento, no emulador, seu aplicativo já será executado:

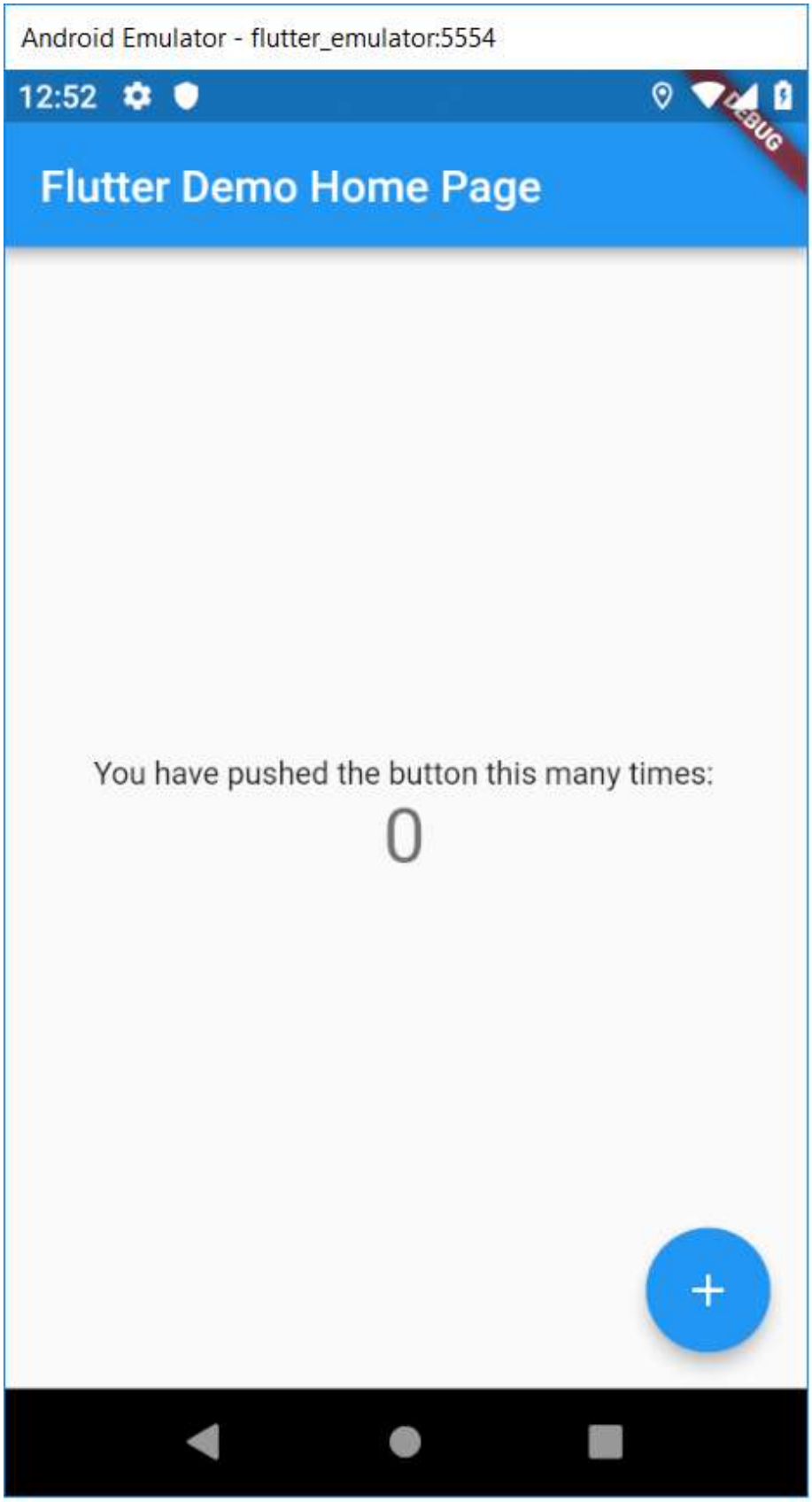


Figura 13. Aplicativo padrão criado automaticamente pelo comando flutter create.

E é isso ai! Você acabou de entrar no mundo do desenvolvimento multiplataforma nativo. Que tal darmos uma explorada no aplicativo de exemplo que foi criado?

Widgets! Explorando um aplicativo Flutter

Tudo no Flutter é um Widget. Seja uma página, um botão ou um texto. Sempre que quisermos exibir qualquer tipo de elemento para o usuário, usaremos Widgets.

O Flutter fornece diversos Widgets que são essenciais para o funcionamento da nossa aplicação. Por isso, no arquivo main.dart onde podemos encontrar todos os elementos do nosso aplicativo de exemplo, logo na primeira linha, encontramos a importação dos Widgets do Material Design:

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

Mais abaixo encontramos o Widget superior da aplicação. É nele, chamado de MyApp, que seremos capazes de definir um esquema de cores padrão, bem como o título e a página inicial do nosso aplicativo. Perceba também que a função main, a “porta de entrada” cria uma instância desse Widget, iniciando o aplicativo:

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


class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);


  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
```

```
    );
  }
}
```

context é uma instância de BuildContext que serve para informar ao Widget em que lugar ele se encontra dentro da árvore de Widgets da aplicação.

Repare que na **linha 9**, no parâmetro primarySwatch do construtor da classe ThemeData, passamos o valor Colors.blue. A classe Colors é fornecida pelo Flutter Material, onde encontramos diversas cores comumente utilizadas pelo Material Design. Que tal mudarmos a cor primária do nosso aplicativo para vermelho?

```
ThemeData(
    primarySwatch: Colors.red,
),
```

É só salvar e, em um passe de mágica, o recurso de Hot Reload do Flutter modificará nosso aplicativo executado no emulador:

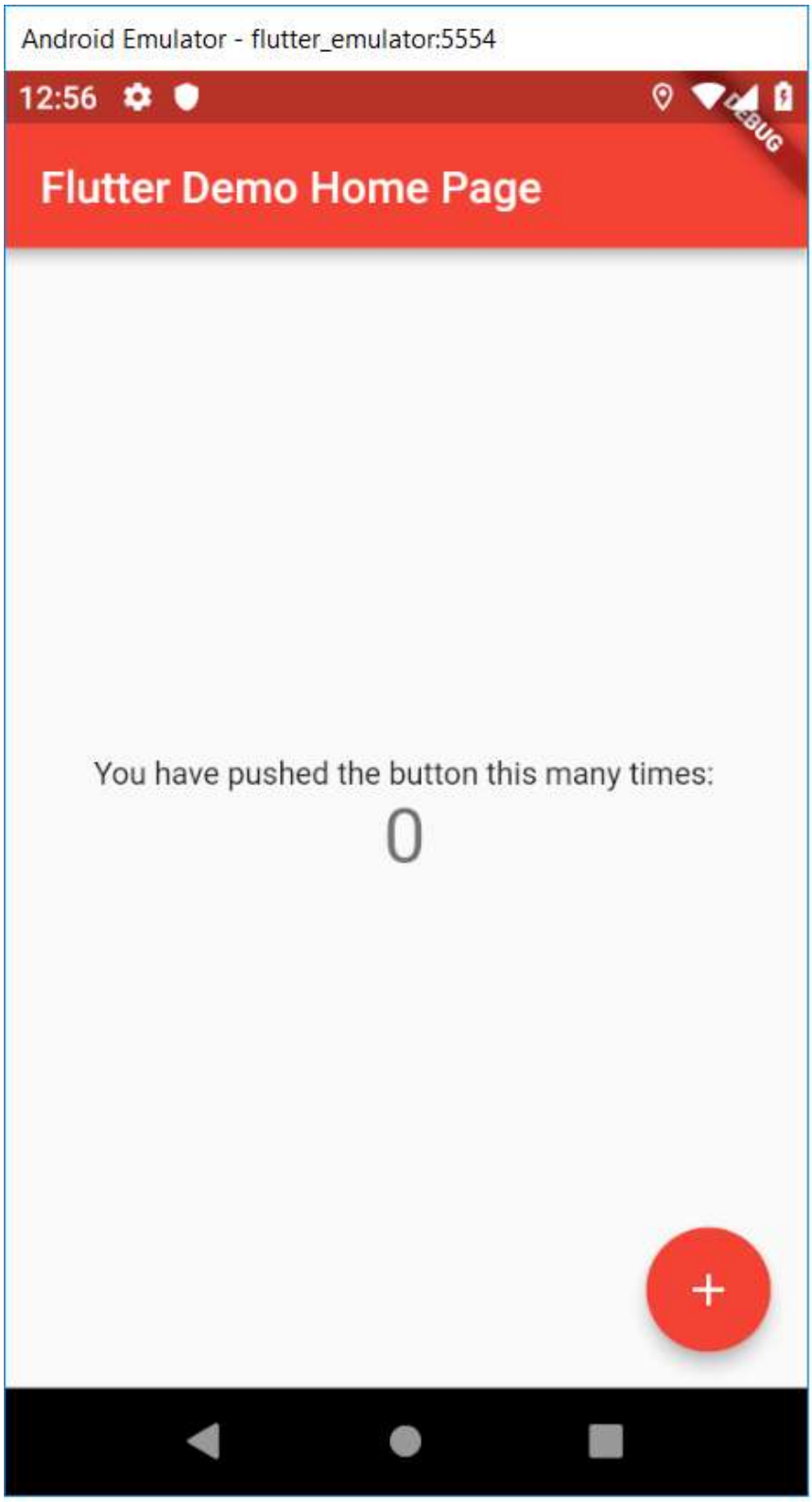


Figura 14. Aplicativo padrão pintado de vermelho.

Mais abaixo encontraremos outras duas classes: MyHomePage e _MyHomePageState. Diferentemente do Widget superior da aplicação, que se tratava de um StatelessWidget (um Widget que não armazena dados), o Widget da nossa página inicial se trata de um StatefulWidget (um Widget que armazena dados e possui Lifecycle). Por isso possuímos duas classes: a primeira, chamada MyHomePage, conterá apenas os dados imutáveis do nosso Widget, ou seja, apenas os dados que forem passados como parâmetro para o Widget. É essa classe que instanciamos dentro do Widget MyApp para abrir a página inicial.

```
class MyHomePage extends StatefulWidget {

 const MyHomePage({Key? key, required this.title}) : super(key: key);


 final String title;


 @override

 State<MyHomePage> createState() => _MyHomePageState();

}
```

A segunda classe, _MyHomePageState, é onde armazenamos os dados mutáveis do componente (estado) e implementamos o método build utilizado pelo Flutter para criar o Widget na tela do usuário:

```
class _MyHomePageState extends State<MyHomePage> {

 int _counter = 0;


 void _incrementCounter() {

  setState(() {

   _counter++;

  });

 }
```

```
@override
Widget build(BuildContext context) {

 return Scaffold(
  appBar: AppBar(
   title: Text(widget.title),
  ),
  body: Center(
   child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
     const Text(
      'You have pushed the button this many times:',
     ),
     Text(
      '$_counter',
      style: Theme.of(context).textTheme.headline4,
     ),
    ],
   ),
  ),
  floatingActionButton: FloatingActionButton(
   onPressed: _incrementCounter,
   tooltip: 'Increment',
   child: const Icon(Icons.add),
  ),
 );
}
```

```
}
```

Perceba que é nesse Widget que encontramos todo o código da nossa aplicação responsável por exibir o contador e, ao clique do botão de incremento, realizamos a atualização dos dados. Agora que alteramos a cor do nosso aplicativo, que tal implementar mais uma funcionalidade? Apenas um botão de incremento não é suficiente, também precisamos decrementar!

Ainda na classe _MyHomePageState criaremos mais um método chamado _decrementCounter. Este método ficará responsável por decrementar o atributo _counter da nossa aplicação e por executar o método setState disponível em todo Widget do tipo Stateful:

```
...
void _decrementCounter() {
  setState(() {
    _counter--;
  });
 }
...
```

Por fim precisaremos, no atributo floatingActionButton do Widget Scaffold, adicionar mais um botão ao lado do botão de incremento. Faremos isso agrupando ambos os botões no Widget Row, também disponibilizado pelo Flutter:

```
floatingActionButton: Row(
   mainAxisAlignment: MainAxisAlignment.end,
   children: <Widget>[
     FloatingActionButton(
       onPressed: _incrementCounter,
       tooltip: 'Increment',
       child: const Icon(Icons.add),
     ),
     const SizedBox(
       width: 10.0,
     ),
```

```
    FloatingActionButton(
      onPressed: _decrementCounter,
      tooltip: 'Decrement',
      child: const Icon(Icons.remove),
    ),
  ],
),
```

Salve as alterações e agora conseguimos mudar a cor de tema da aplicação, adicionando uma nova funcionalidade em alguns poucos passos.

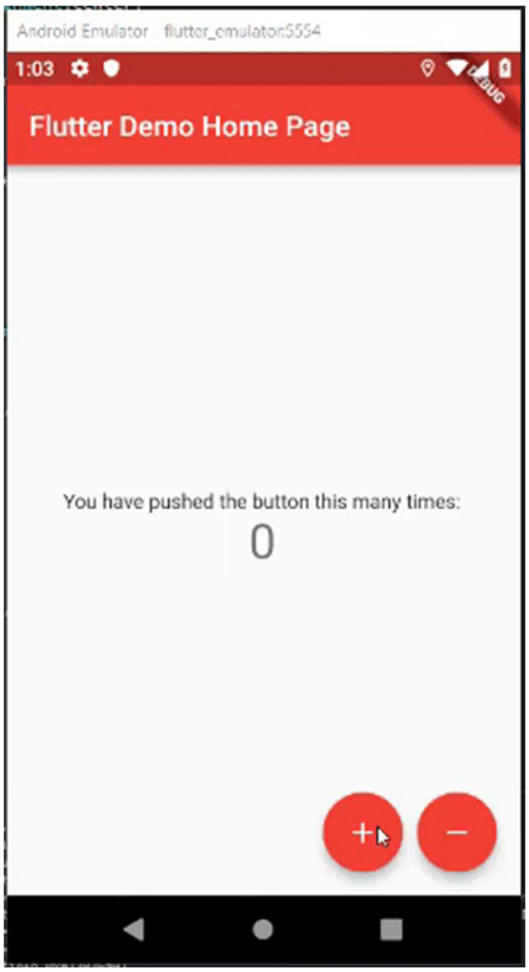


Figura 15. Aplicativo aperfeiçoado com decremento do contador.

Veja abaixo o código completo

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.red,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

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

  final String title;
```

```
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  void _decrementCounter() {
   setState(() {
     _counter--;
   });
  }

  @override
  Widget build(BuildContext context) {

    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
```

```
  child: Column(
   mainAxisAlignment: MainAxisAlignment.center,
   children: <Widget>[
    const Text(
     'You have pushed the button this many times:',
    ),
    Text(
     '$_counter',
     style: Theme.of(context).textTheme.headline4,
    ),
   ],
  ),
 ),
 floatingActionButton: Row(
  mainAxisAlignment: MainAxisAlignment.end,
  children: <Widget>[
   FloatingActionButton(
    onPressed: _incrementCounter,
    tooltip: 'Increment',
    child: const Icon(Icons.add),
   ),
   const SizedBox(
    width: 10.0,
   ),
   FloatingActionButton(
    onPressed: _decrementCounter,
    tooltip: 'Decrement',
    child: const Icon(Icons.remove),
```

```
          ),
        ],
      ),
    );
  }
}
```