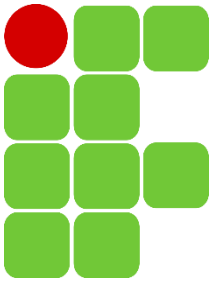


Instituto Federal de Educação, Ciência e Tecnologia
São Paulo, campus Cubatão.

História do Flutter

Projeto de Sistemas

Alexandre Siqueira Souza Costa, 179048X
Davi Fagundes Ferreira da Silva, 179082X
Fernanda da Silva Rezende, 1790366
Gabriela Alves Silva de Carvalho, 1990799
Juliana Almeida Santos, 1790048
Júlio Cesar Oliveira da Silva, 1790021
Marina Cova Lacerda, 1690264
Rayssa Oliveira Santos, 1790153
Rebecca Loiola Messali, 179034X
Reginaldo Beserra de Lima Filho, 1690884
Stefany Tam Pereira Mendes, 1790251

Cubatão, 2020

INTRODUÇÃO

A demanda por alternativas para o desenvolvimento de aplicativos mobile tem crescido cada vez mais nos últimos anos. Apesar de ser uma área relativamente recente – e em evolução – diversas alternativas foram apresentadas nos últimos anos.

A Figura 01 apresenta uma linha do tempo de lançamento das principais ferramentas desenvolvidas, em pouco mais de uma década, para a produção de aplicativos para dispositivos móveis.

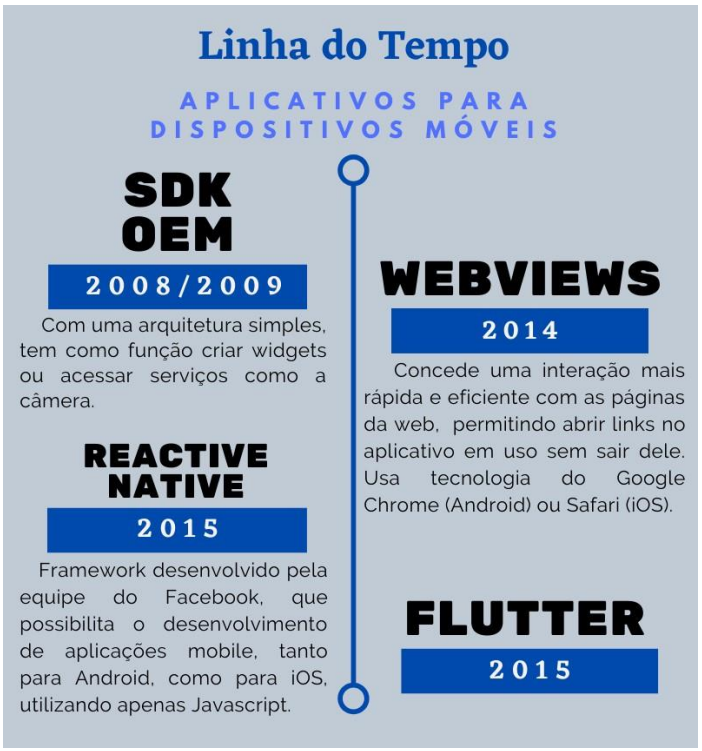


***Figura 1**- Linha do tempo do lançamento de aplicativos para dispositivos móveis.*

Fonte: Compilação do grupo.

Desenvolvido pelo Google e programado na linguagem Dart, o Flutter é um Framework - isto é, um conjunto de bibliotecas utilizadas para criar uma base, onde as aplicações são construídas - para o desenvolvimento de aplicativos mobile para Android e iOS, sendo seu principal objetivo a resolução de dificuldades recorrentes com uma abordagem generalista.

HISTÓRIA

A princípio conhecido pelo codinome Sky, o projeto do framework Flutter foi iniciado em 2014, com o objetivo de encontrar uma solução que desempenhasse

maior performance na construção de interfaces mobile. Já nomeado por Flutter, sua primeira apresentação ocorreu no ano de 2015, durante a Dart Developer Summit, onde foi possível observar o código Dart sendo executado em um aparelho Android.

A estreia do framework em uma grande conferência foi na Google IO 2017, onde foi usado em uma sessão de *live coding* para construir um aplicativo integrado ao Firebase – outro Framework de desenvolvimento mobile – com acesso à câmera. Em 2019, um ano após o lançamento de sua primeira versão estável, o Flutter chegou a sua segunda versão.

Em maio de 2020, ocorreram os lançamentos do Dart SDK 2.8 e do Flutter 1.17.0. Com isso, houve o desenvolvimento de novos recursos, como o aperfeiçoamento do desempenho em dispositivos iOS, a criação de novos widgets do material, além da disponibilização de novas ferramentas de rastreamento de rede.

LINGUAGEM DE PROGRAMAÇÃO

O Flutter conta com uma abordagem diferenciada a fim de evitar problemas de desempenho causados pela necessidade de uma ponte JavaScript, utilizando uma linguagem de programação compilada, neste caso, Dart.

Dart é um compilado “à frente do tempo” (AOT), ou seja, compila um alto nível de programação em código nativo para múltiplas plataformas. Logo, permite que o Flutter se relacione com a plataforma sem passar por uma ponte de JavaScript, o que poderia gerar uma mudança de contexto. Além disso, compilar para código nativo aperfeiçoa os tempos de inicialização do aplicativo.

Algumas características podem ser citadas a fim de tornar o Flutter um framework atrativo, como o fato de ser o único SDK móvel que oferece visões reativas sem exigir uma ponte JavaScript e a implementação de widgets.

CONCLUSÃO

Em resumo, o Flutter conta não somente com a vantagem de utilizar o Dart, linguagem de programação nova e simples, mas também com a característica de ser multiplataforma – isto é, capaz de desenvolver aplicações em qualquer sistema operacional, permitindo a criação de *apps* nativos a partir de um único código-base -, além do acesso aos recursos do dispositivo como a câmera, o Wi-Fi e a memória.

A partir desse conjunto de benefícios, grandes empresas, como Google e Nubank, têm investido no Flutter para desenvolvimento de suas soluções, propiciando a evolução do Framework e transformando-o em uma das principais opções de estudo para desenvolvedores que buscam uma alternativa para suas aplicações mobile.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDRADE, A. GUEDES, M., 2020. *O Que É Flutter?* - **Blog Da Treinaweb**. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-flutter/>. Acesso em: 13 de set. de 2020.

ARAJABAT. *O que é revolucionário sobre Flutter.* **Agatetepê**, 2018. Disponível em: <https://www.agatetepe.com.br/o-que-e-revolucionario-sobre-flutter/>. Acesso em: 13 set. de 2020.

DE MELLO, Rubens. *Arquitetura*. **Flutter para iniciantes**, 2019. Disponível em: <https://www.flutterparainiciantes.com.br/arquitetura>. Acesso em: 13 de set. de 2020.

DE MELLO, Rubens. *O que é Flutter?*. **Flutter para iniciantes**, 2019. Disponível em: <https://www.flutterparainiciantes.com.br/arquitetura>. Acesso em: 13 de set. de 2020.

LELER, Wm. *What's awesome about Flutter.* **code.talks**, 2017. Disponível em: <https://www.youtube.com/watch?v=n3FLHWF7UgM>. Acesso em: 13 de set. de 2020.

MAGALHÃES, Túlio. *Flutter: tudo sobre o queridinho do Google.* **Zup**, 2019. Disponível em: <https://www.zup.com.br/blog/flutter>. Acesso em: 15 de set. de 2020.

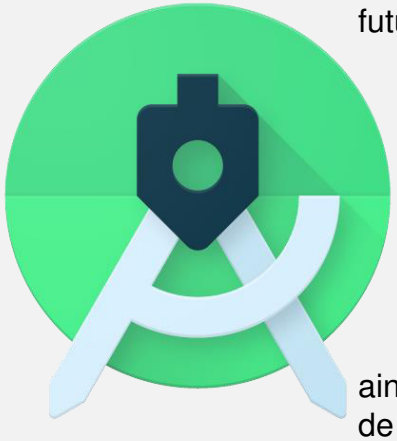
NUNES, Filipe. E este tal de flutter, bora iniciar e entender. **Medium**, 2019. Disponível em: <https://medium.com/@FilipeFNunes/e-este-tal-de-flutter-bora-iniciare-entender-5278cbebe2ca>. Acesso em: 15 de set. de 2020.

As Principais IDEs Utilizadas em Flutter

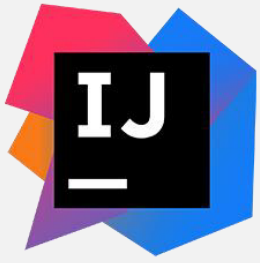
Integrated Development Environment – IDE (Ambiente de Desenvolvimento Integrado) é um software muito utilizado por desenvolvedores com o objetivo de facilitar diversos processos referentes ao desenvolvimento, combinando ferramentas comuns em uma única interface gráfica do usuário (GUI).

A principal vantagem da IDE é a criação de aplicações de maneira mais rápida, uma vez que ela auxilia em todo o processo de seu desenvolvimento, provendo diversos benefícios, como a análise de todo o código juntamente ao *debugger*, o recurso de autocomplemento de trechos de códigos e o compilador integrado.

Considerando o explicado e retomando a aplicação Flutter tratada anteriormente, escolheu-se o Android Studio como meio de programação para os futuros trabalhos, devido à gama de características e ferramentas de apoio próprias para a criação de aplicativos para dispositivos móveis Android. Além disso, essa IDE oferece recursos que aumentam a produtividade agilidade em sua compilação, como um emulador rápido com inúmeros recursos, frameworks e ferramentas de teste com diversas possibilidades e compatibilidade com C++ e NDK.



Além do Android Studio, encontram-se ainda outras opções de softwares para o desenvolvimento de programas utilizando o framework Flutter, como o VSCode, o IntelliJ e o Eclipse, que trabalham de modo semelhante ao Android Studio, auxiliando e facilitando o desenvolvimento do projeto. Vale pontuar que um dos vieses mais importantes na escolha da IDE é compreender as especificidades de cada meio de programação e tentar achar aquele que contempla os objetivos do seu projeto, de modo a conseguir aproveitar toda sua desenvoltura e presteza.



Referências Bibliográficas

HARADA, Eduardo. O que é o Android Studio, ferramenta criada para desenvolver apps mobile - Tecmundo. Disponível em: <https://www.tecmundo.com.br/software/146361-o-android-studio-ferramenta-criada-desenvolver-apps-mobile.htm>. Acesso em: 30 de set. de 2020.

ETO, Tremaine. What is the best IDE for developing in Dart and Flutter? Disponível em: <https://medium.com/cloud-native-the-gathering/what-is-the-best-ide-for-developing-in-dart-and-flutter-1d9e6ec50343>. Acesso em: 30 set. de 2020.

DIONISIO, Edson. Introdução ao Visual Studio Code. Disponível em: <https://www.devmedia.com.br/introducao-ao-visual-studio-code/34418>. Acesso em: 30 de set. de 2020.

Conheça o Android Studio. Disponível em: <https://developer.android.com/studio/intro?hl=pt-br>. Acesso em: 30 de set. de 2020.

Projeto de Sistemas

Alexandre Siqueira Souza Costa - 179048X
Davi Fagundes Ferreira da Silva - 179082X
Fernanda da Silva Rezende - 1790366
Gabriela Alves Silva de Carvalho - 1990799
Juliana Almeida Santos - 1790048
Júlio Cesar Oliveira da Silva - 1790021
Marina Cova Lacerda - 1690264
Rayssa Oliveira Santos - 1790153
Rebecca Loiola Messali - 179034X
Reginaldo Beserra de Lima Filho - 1690884
Stefany Tam Pereira Mendes – 1790251

Cubatão, 2020

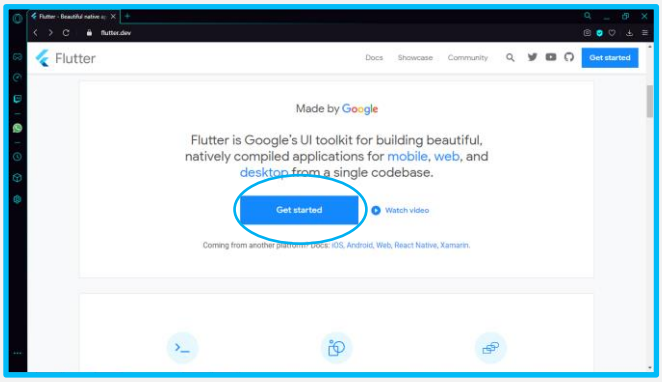
Guia de instalação

Flutter

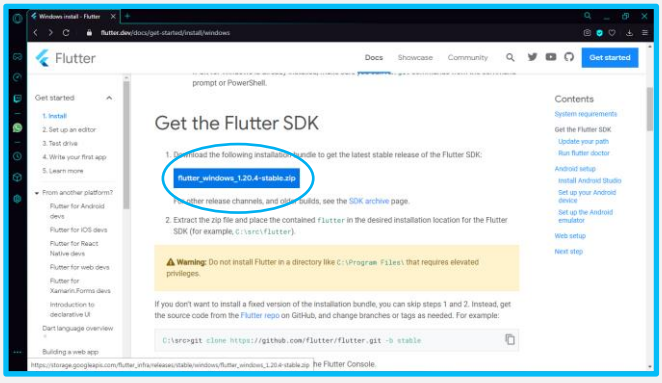
OLÁ! Neste material, você aprenderá como baixar, instalar e configurar todos os fatores necessários para começar a programar na linguagem Flutter.

1 Baixando o Flutter

- Acesse o site flutter.dev e clique em “Get Started”.



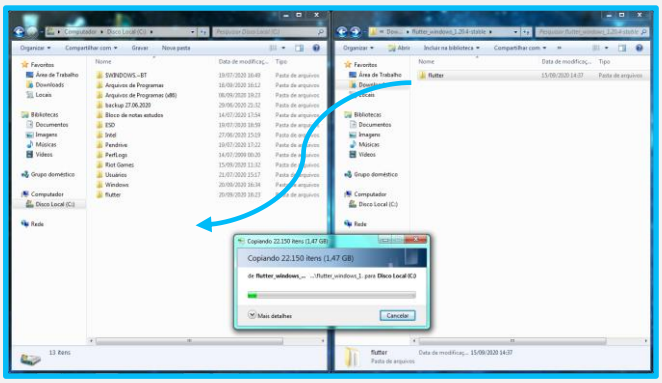
- Em seguida, escolha o sistema operacional correspondente à sua máquina e vá direto a “Get the Flutter SDK”.



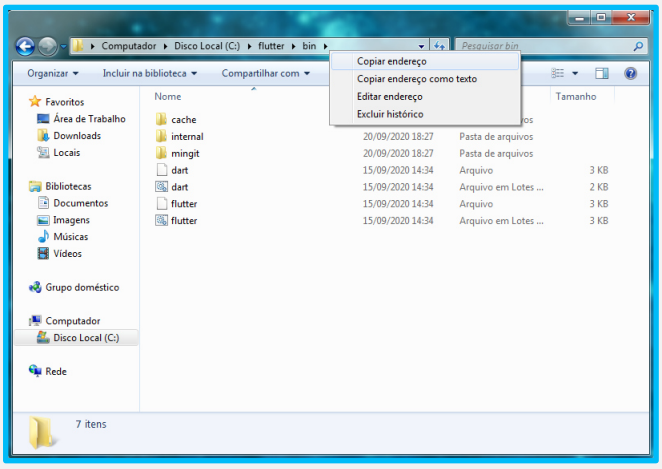
- Faça o download do SDK clicando no texto destacado em azul.

2 Instalando o Flutter

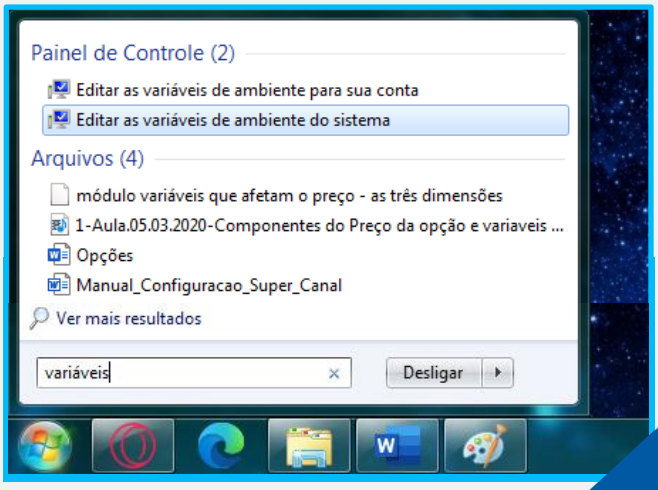
- Extraia o arquivo compactado, abra-o e mova a pasta “Flutter”, presente na pasta extraída, para o disco local do seu computador.



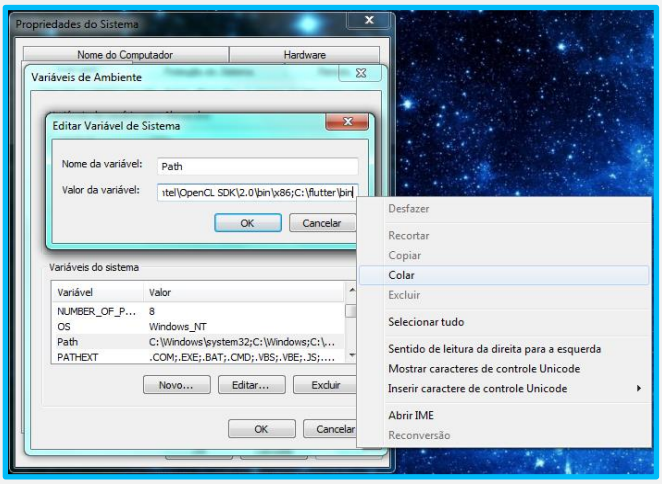
- Entre na pasta movida, clique em “bin” e copie o caminho até a pasta acessada (clicando com o botão direito na barra de diretório no canto superior).



- Vá para “Editar as variáveis de ambiente do sistema”.



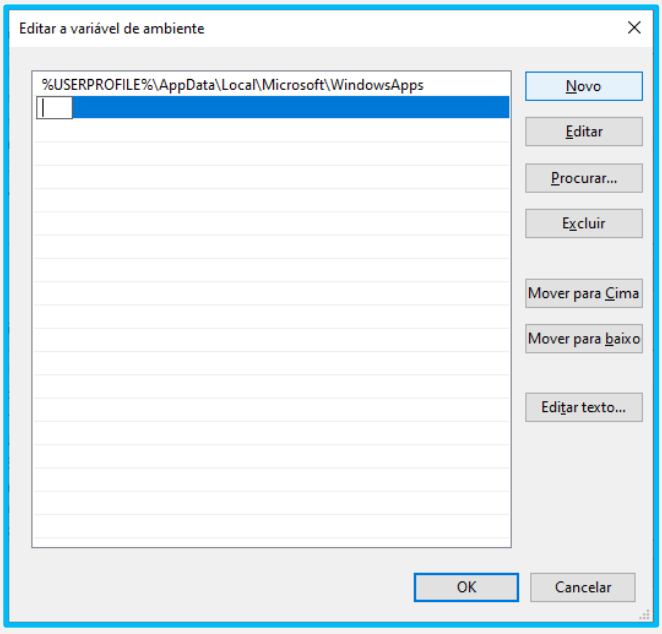
- Após isso, clique em “Variáveis de Ambiente”, no canto inferior da janela, localize e clique em “editar” selecionando a variável “Patch”.



- E então cole o endereço copiado anteriormente, colocando um ponto e vírgula (;), a fim de separar os diretórios. Clique em “OK”.

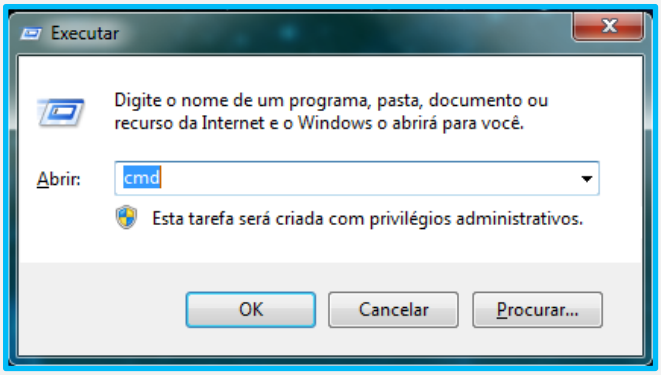
Observação: Caso esteja utilizando o Windows 10, siga a imagem abaixo. Caso contrário, vá direto para o passo 3.

- Após clicar em editar a variável “Patch”, aperte o botão “novo e insira o endereço copiado da pasta “bin” do Flutter. Clique em “OK”.

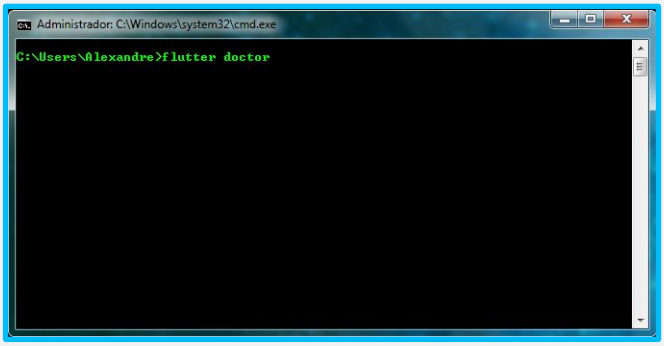


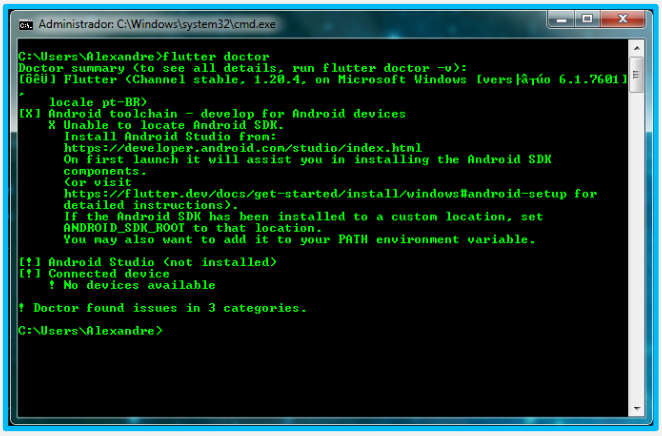
3 Verificando a instalação

- Inicie o CMD.



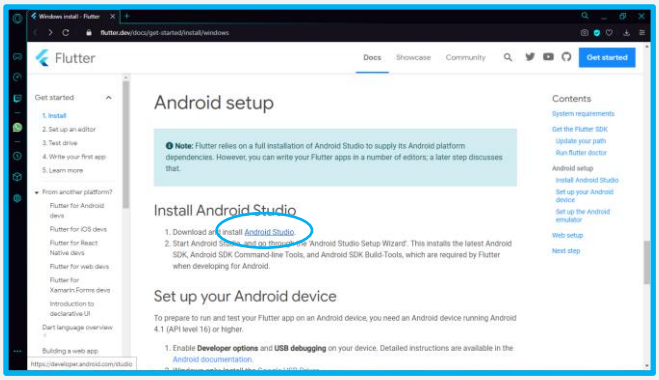
- Em seguida, digite *flutter doctor* e, no teclado, pressione Enter. Sua tela do CMD deverá estar igual à segunda imagem a seguir, caso não esteja, repita os passos anteriores.





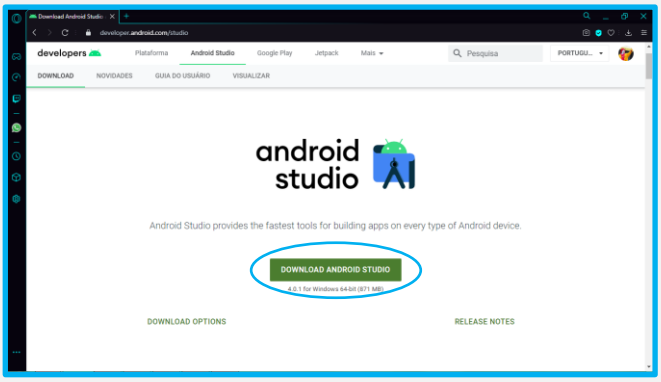
4 Baixando o Android Studio

- Retorne a “Get Started”, em flutter.dev, e busque por “Android setup”.



- Clique em “Android Studio”, destacado em azul, e logo você será redirecionado para outra página.

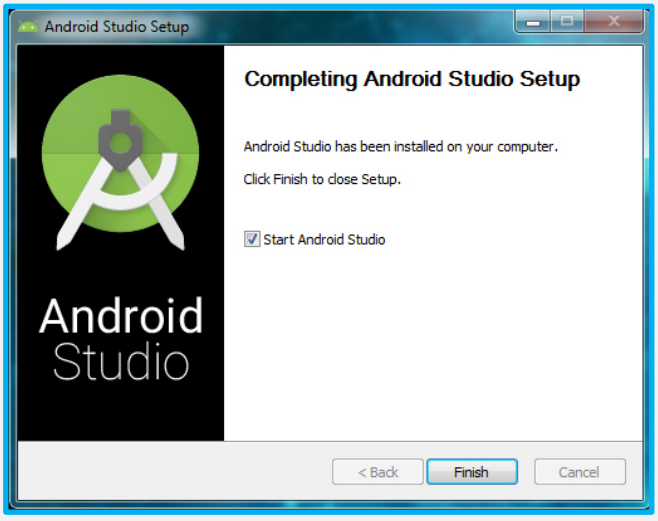
- Em “Download Android Studio”, encontra-se o aplicativo. Concorde com os termos de uso solicitados e baixe para o seu computador.



5 Instalando o Android Studio

- Com o aplicativo baixado, execute-o.

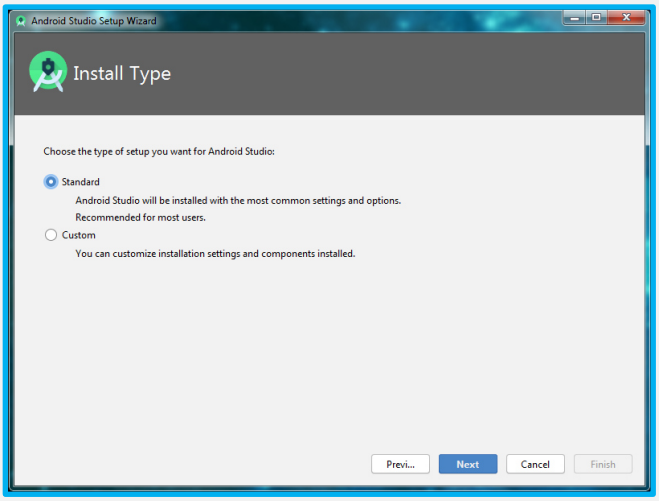
- Mantenha as configurações recomendadas pelo fornecedor e faça a instalação.



- Concorde ou não em enviar estatísticas para o Google e inicie o aplicativo.

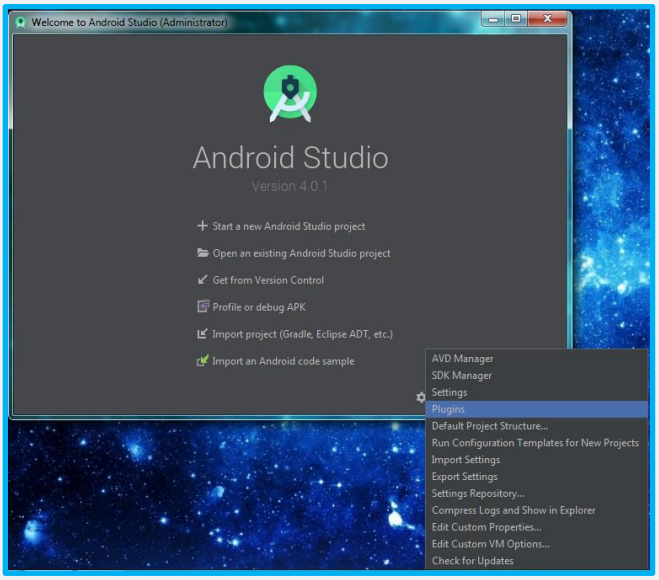
6 Configurando o Android Studio

- Após iniciar, clique em “Next” e opte pela versão Standard.

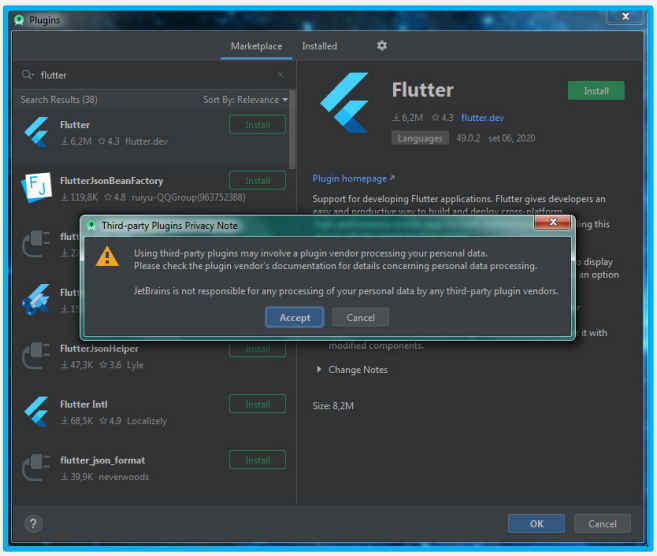


- Escolha a cor da interface que mais lhe agrada e finalize a configuração.

- Na tela de menu, clique em “Configure” e procure por “Plugins”.



- Pesquise “Flutter”, na barra de pesquisas, e instale seu plugin junto com o plugin “Dart”, que é necessário, aceitando seus termos de uso.

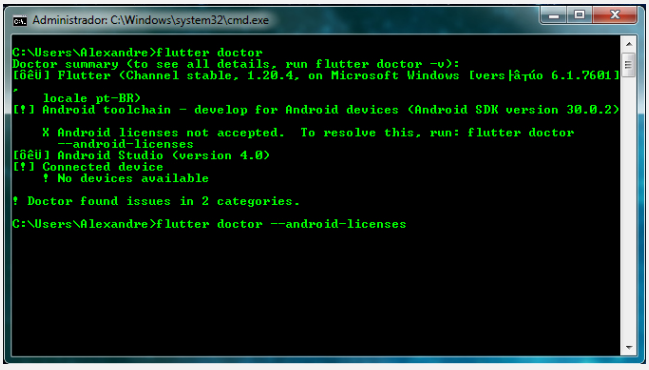


- Após a instalação, será solicitado que o Android Studio seja reiniciado. Concorde.

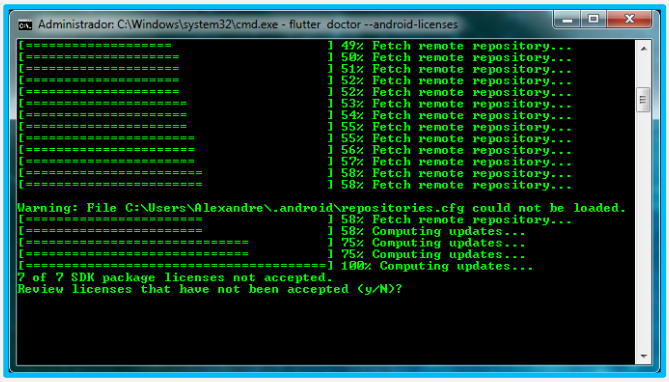
7 Finalizando a configuração

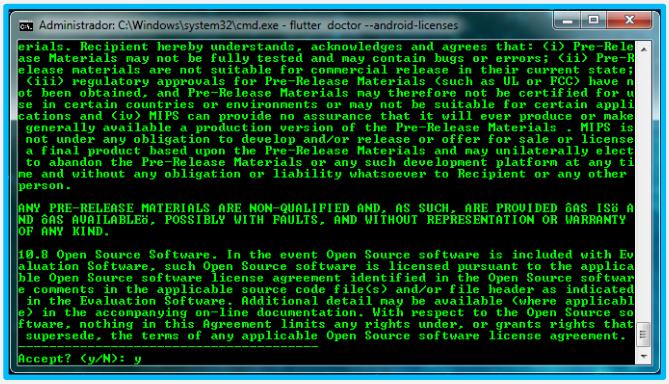
- Inicie o CMD e execute o comando *flutter doctor*.

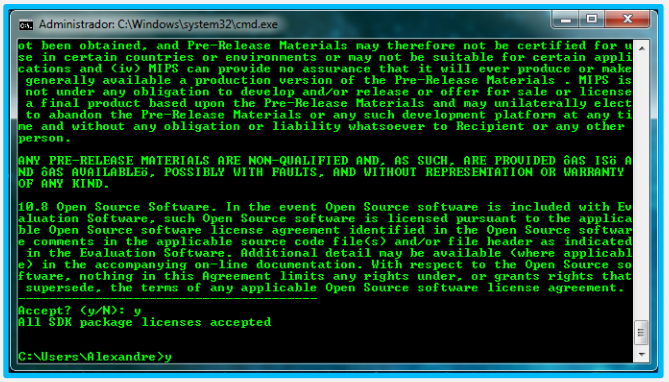
- Ao terminar a execução do primeiro comando, digite *flutter doctor –android-licenses*.



- Ao iniciar do comando, será apresentado para você mais sete termos de uso. É necessário que todos sejam aceitos, e para isso, digite “*y*” para confirmar cada um.

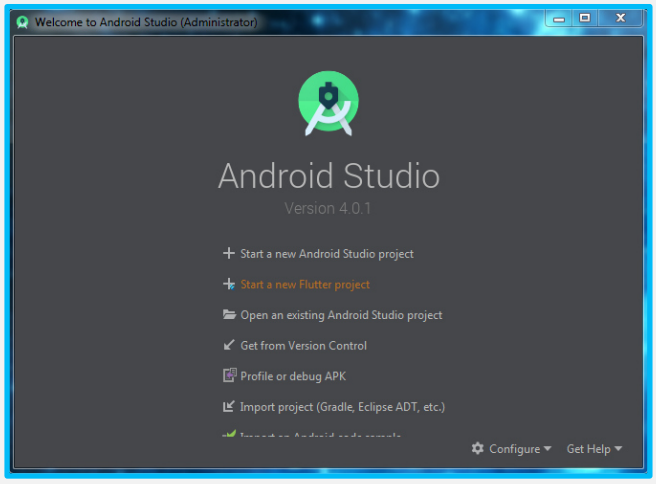




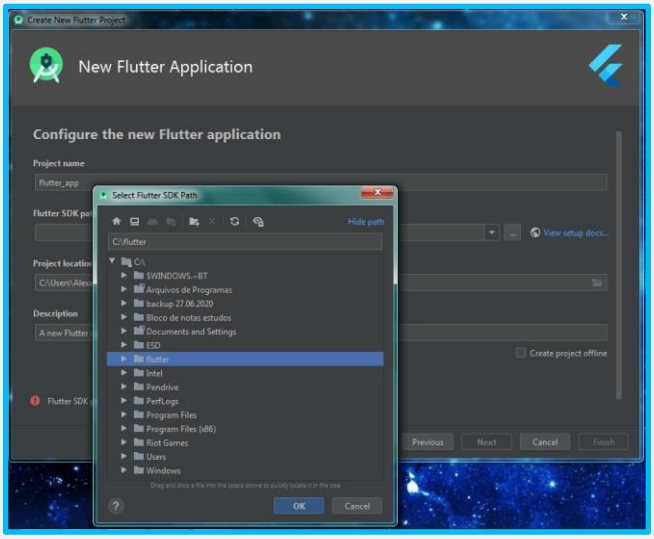


8 Iniciando um projeto

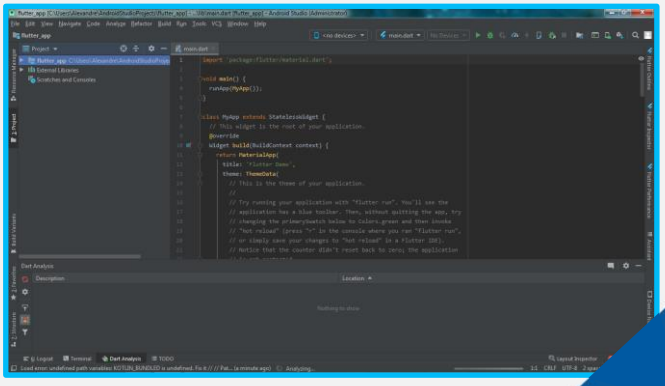
- Vá em *Start a new Flutter project*



- Clique em *Flutter Application* e em “next”.
- Selecione o SDK do Flutter que foi posto na pasta local do seu computador.



- Na página seguinte, clique em “Finish”
- Agora, você está pronto para desenvolver seus projetos em Flutter!



Projeto de Sistemas

Alexandre Siqueira Souza Costa - 179048X
Davi Fagundes Ferreira da Silva - 179082X
Fernanda da Silva Rezende - 1790366
Gabriela Alves Silva de Carvalho - 1990799
Juliana Almeida Santos - 1790048
Júlio Cesar Oliveira da Silva - 1790021
Marina Cova Lacerda - 1690264
Rayssa Oliveira Santos - 1790153
Rebecca Loiola Messali - 179034X
Reginaldo Beserra de Lima Filho - 1690884
Stefany Tam Pereira Mendes – 1790251

Cubatão, 2020

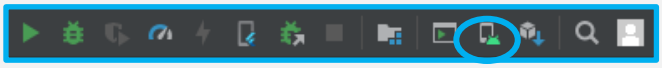
“Hello World”

Flutter

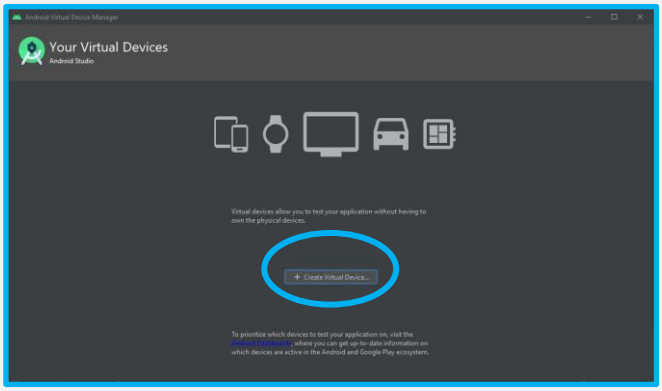
OLÁ! É comum, no universo da programação, ser colocada a frase “Hello World” como primeiro código a ser exibido pela máquina, sendo o marco do inicio da jornada como programador. Hoje, neste material, você aprenderá como fazê-la!

1 Instalando o Emulador

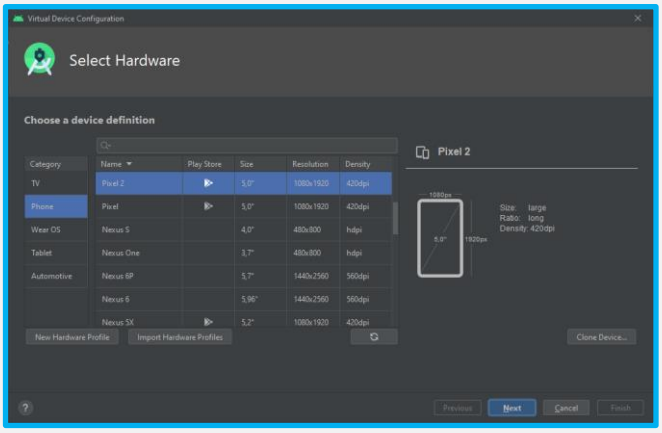
- Você deve primeiro configurar a maneira como verá seu programa executado. Para isso, é necessário um emulador Android ou um celular.
- Para este tutorial, utilizaremos o emulador presente no próprio Android Studio instalado anteriormente.



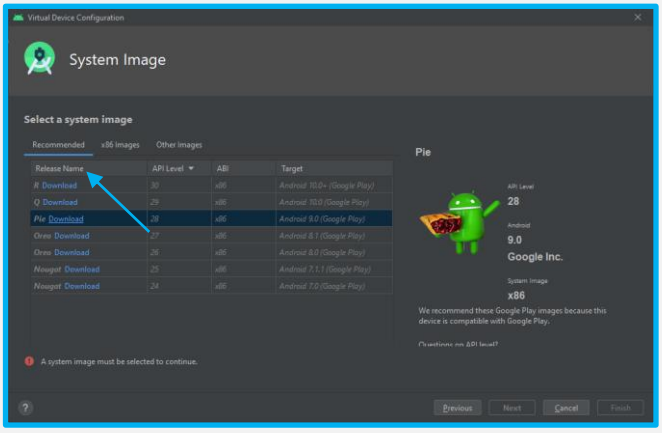
- Com o projeto iniciado, vá em “AVD Manager” no canto superior direito da tela. Com a nova janela aberta, clique em “Create Virtual Device”.



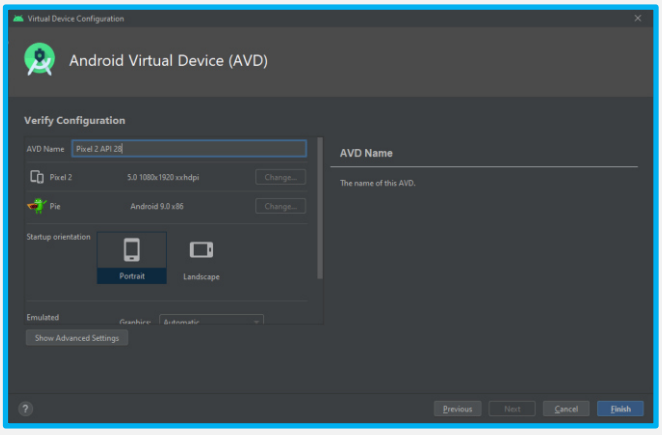
- Selecione o celular que deseja emular. É recomendado que se utilize as opção predefinidas pelo Android Studio.



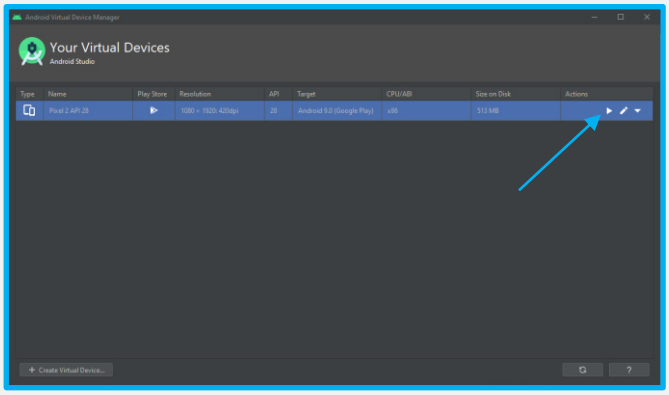
- Escolha o sistema que será utilizado e o baixe. Após baixado, clique em “next”.



- Selecione a orientação do aplicativo.



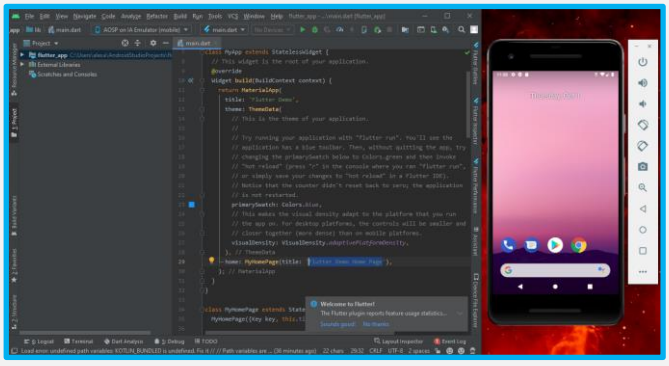
- Inicie o emulador.



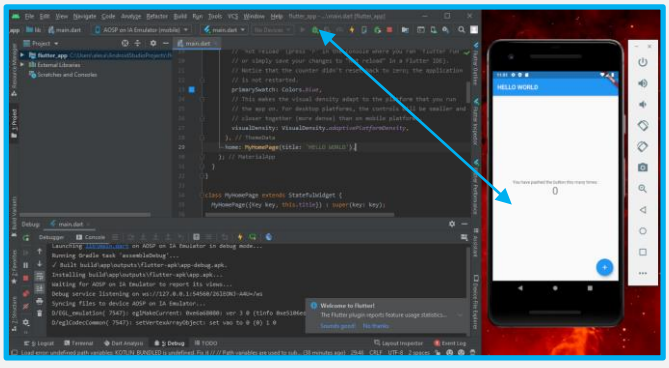
2 Codificando

- Com o emulador aberto, procure pela linha "*home: MyHomePage(title: "flutter Demo Home Page"),*".

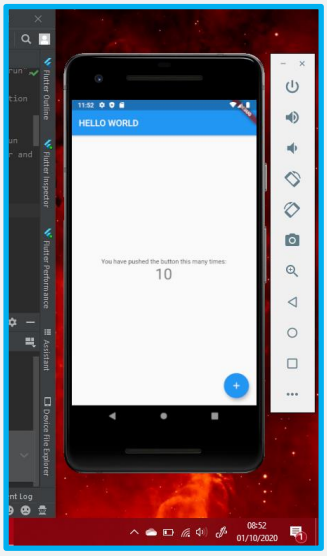
- Selecione *"flutter Demo Home Page"* e substitua por *"HELLO WORLD"*.



- Clique em *"Debug" que está ao lado do botão de "play"* e aguarde a inicialização do programa.



- Repare que a frase que você digitou, aparece no canto superior esquerdo do celular, ela representa, neste caso, o nome no menu onde você se encontra.
- E a cada clique que o botão "+", na tela, recebe, um numero é somado ao contador



- Ao criar o projeto, este pequeno programa já vem pronto para que novos programadores, como você, possam ter um primeiro contato com a linguagem.
- Repare que cada linha é explicada com "//" para um melhor entendimento daqueles que são novos nesse mundo.

- Agora você pode explorar as milhares funções que existem no Flutter!

Projeto de Sistemas

Alexandre Siqueira Souza Costa - 179048X
Davi Fagundes Ferreira da Silva - 179082X
Fernanda da Silva Rezende - 1790366
Gabriela Alves Silva de Carvalho - 1990799
Juliana Almeida Santos - 1790048
Júlio Cesar Oliveira da Silva - 1790021
Marina Cova Lacerda - 1690264
Rayssa Oliveira Santos - 1790153
Rebecca Loiola Messali - 179034X
Reginaldo Beserra de Lima Filho - 1690884
Stefany Tam Pereira Mendes – 1790251

Cubatão, 2020

“Área do triângulo”

Flutter

Ao se iniciar o desenvolvimento de uma aplicação em Flutter no Android Studio, o projeto já vem com diversos comentários em inglês, os quais auxiliam muito os novatos em desenvolvimento no Framework. Para ser possível a distinção dos comentários e partes já vindas com o projeto, será exibido em sequência todo o código, e durante o passo a passo, apenas cortes das partes modificadas.

1 Código completo

```
import 'package:flutter/material.dart';
import 'package:flutter/services.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: 'Triângulo',
   theme: ThemeData(
    // This is the theme of your application.
    //
    // Try running your application with "flutter run". You'll see the
    // application has a blue toolbar. Then, without quitting the app, try
    // changing the primarySwatch below to Colors.green and then invoke
    // "hot reload" (press "r" in the console where you ran "flutter run",
    // or simply save your changes to "hot reload" in a Flutter IDE).
    // Notice that the counter didn't reset back to zero; the application
    // is not restarted.
    primarySwatch: Colors.deepPurple,
    // This makes the visual density adapt to the platform that you run
    // the app on. For desktop platforms, the controls will be smaller and
    // closer together (more dense) than on mobile platforms.
    visualDensity: VisualDensity.adaptivePlatformDensity,
   ),
   home: MyHomePage(title: 'Área dos Triângulos'),
  );
 }
}

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

 // This widget is the home page of your application. It is stateful, meaning
 // that it has a State object (defined below) that contains fields that affect
 // how it looks.

 // This class is the configuration for the state. It holds the values (in this
 // case the title) provided by the parent (in this case the App widget) and
 // used by the build method of the State. Fields in a Widget subclass are
 // always marked "final".

 final String title;

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

class _MyHomePageState extends State<MyHomePage> {
 double _b = 0;
 double _h = 0;
 double _area = 0;


 void _atualizaArea() {
  setState(() {
   _area = double.parse((_b * _h / 2).toStringAsFixed(2));
  });
 }
@override
 Widget build(BuildContext context) {
  // This method is rerun every time setState is called, for instance as done
  // by the _incrementCounter method above.
  //
  // The Flutter framework has been optimized to make rerunning build methods
  // fast, so that you can just rebuild anything that needs updating rather
  // than having to individually change instances of widgets.
  return Scaffold(
   appBar: AppBar(
    // Here we take the value from the MyHomePage object that was created by
    // the App.build method, and use it to set our appbar title.
    title: Text(widget.title),
   ),
   body: Center(
    // Center is a layout widget. It takes a single child and positions it
    // in the middle of the parent.
    child: Column(
     // Column is also a layout widget. It takes a list of children and
     // arranges them vertically. By default, it sizes itself to fit its
     // children horizontally, and tries to be as tall as its parent.
     //
     // Invoke "debug painting" (press "p" in the console, choose the
     // "Toggle Debug Paint" action from the Flutter Inspector in Android
     // Studio, or the "Toggle Debug Paint" command in Visual Studio Code)
     // to see the wireframe for each widget.
     //
     // Column has various properties to control how it sizes itself and
     // how it positions its children. Here we use mainAxisAlignment to
     // center the children vertically; the main axis here is the
```

```
vertical
      // axis because Columns are vertical (the cross axis would be
      // horizontal).
      mainAxisAlignment: MainAxisAlignment.center,

      children: <Widget>[
       TextFormField(
         //controller: _controller,
         keyboardType: TextInputType.number,
         inputFormatters: <TextInputFormatter>[
           FilteringTextInputFormatter.allow(RegExp(r'^([0-9]+([.][0-
9]*)?|[.][0-9]+)$')),
         ],
         onChanged: (input) => _b = double.tryParse(input),
         cursorColor: Colors.amber,
         cursorWidth: 5.0,
         decoration: InputDecoration(labelText: "Digite a base do seu
triângulo")

       ),
       SizedBox(height: 25),
        TextFormField(
         //controller: _controller,
         keyboardType: TextInputType.number,
         inputFormatters: <TextInputFormatter>[
           FilteringTextInputFormatter.allow(RegExp(r'^([0-9]+([.][0-
9]*)?|[.][0-9]+)$')),
         ],
         onChanged: (input) => _h = double.tryParse(input),
         cursorColor: Colors.amber,
         cursorWidth: 5.0,
         decoration: InputDecoration(labelText: "Digite a altura do seu
triângulo")

       ),
       SizedBox(height: 25),
       Text(
        'A área do triângulo é $_area',
        style: Theme.of(context).textTheme.headline5,
       ),
      ],
     ),
    ),
   floatingActionButton: FloatingActionButton(
     onPressed: _atualizaArea,
     tooltip: 'Atualizar',
     child: Icon(Icons.arrow_forward_ios),
    ), // This trailing comma makes auto-formatting nicer for build methods.
   );
  }
}
```

2 Passo a passo

1. Definição de variáveis em Double.

```
class _MyHomePageState extends State<MyHomePage> {
 double _b = 0;
 double _h = 0;
 double _area = 0;
```

2. Definição do método que calcula e retorna a área do triângulo toda vez que o mesmo for chamado.

```
void _atualizaArea() {
  setState(() {
    _area = double.parse((_b * _h / 2).toStringAsFixed(2));
  });
 }
```

3. Definição da classe que exibe a solicitação da base do triângulo e trata o resultado.

```
TextFormField(
          //controller: _controller,
          keyboardType: TextInputType.number,
          inputFormatters: <TextInputFormatter>[
            FilteringTextInputFormatter.allow(RegExp(r'^([0-
9]+([.][0-9]*)?|[.][0-9]+)$')),
          ],
          onChanged: (input) => _b = double.tryParse(input),
          cursorColor: Colors.amber,
          cursorWidth: 5.0,
          decoration: InputDecoration(labelText: "Digite a
base do seu triângulo")
           ),
        SizedBox(height: 25),
```

3. a) Definição do teclado como numérico

```
          keyboardType: TextInputType.number,
```

3. b) Processo de tratamento do conteúdo digitado pelo usuário, lembrando que o mesmo deve ser um número maior que zero e só pode conter nele uma vírgula.

```
          inputFormatters: <TextInputFormatter>[
           FilteringTextInputFormatter.allow(RegExp(r'^([0-
9]+([.][0-          9]*)?|[.][0-9]+)$')),
          ],
```

3. c) Atualização da variável já definida em Double quando um novo valor for inserido pelo usuário.

```
onChanged: (input) => _b = double.tryParse(input),
```

3. d) Definição a cor do cursor. Nesse caso a cor escolhida foi *amber*, que é amarelo.

```
cursorColor: Colors.amber,
```

3. e) Definição do tamanho do cursor.

```
cursorWidth: 5.0,
```

3. f) Definição do texto que será exibido em cima do campo de texto.

```
decoration: InputDecoration(labelText: "Digite a base do seu triângulo")
```

4. Definição da classe que exibe a solicitação da base do triângulo e trata o resultado.

```
TextFormField(
        //controller: _controller,
        keyboardType: TextInputType.number,
        inputFormatters: <TextInputFormatter>[
          FilteringTextInputFormatter.allow(RegExp(r'^([0-9]+([.][0-9]*)?|[.][0-9]+)$')),
        ],
        onChanged: (input) => _h = double.tryParse(input),
        cursorColor: Colors.amber,
        cursorWidth: 5.0,
        decoration: InputDecoration(labelText: "Digite a altura do seu triângulo")
        ),
        SizedBox(height: 25),
```

4. a) Definição do teclado como numérico

```
keyboardType: TextInputType.number,
```

4. b) Processo de tratamento do conteúdo digitado pelo usuário, lembrando que o mesmo deve ser um número maior que zero e só pode conter nele uma vírgula.

```
inputFormatters: <TextInputFormatter>[
          FilteringTextInputFormatter.allow(RegExp(r'^([0-9]+([.][0-9]*)?|[.][0-9]+)$')),
        ],
```

4. c) Atualização da variável já definida em Double quando um novo valor for inserido pelo usuário.

```
onChanged: (input) => _h = double.tryParse(input),
```

4. d) Definição a cor do cursor. Nesse caso a cor escolhida foi *amber*, que é amarelo.

```
cursorColor: Colors.amber,
```

4. e) Definição do tamanho do cursor.

```
cursorWidth: 5.0,
```

4. f) Definição do texto que será exibido em cima do campo de texto.

```
decoration: InputDecoration(labelText: "Digite a altura do seu triângulo")
```

5. Definição do método que exibe o resultado da área do triângulo.

```
Text(
          'A área do triângulo é $_area',
        style: Theme.of(context).textTheme.headline5,
      ),
```

5. a) Define o texto que será exibido, sendo *$_area* o placeholder que contém o valor da área do triângulo

```
Text(
        'A área do triângulo é $_area',
```

5. b) Definição do estilo do texto exibido

```
style: Theme.of(context).textTheme.headline5,
```

6. Atribuição do botão ao método _atualizarArea, ou seja, toda vez que o botão for clicado, o método será chamado.

```
floatingActionButton: FloatingActionButton(
    onPressed: _atualizaArea,
    tooltip: 'Atualizar',
    child: Icon(Icons.arrow_forward_ios),
  ),
```

6. a) Definição do texto exibido quando o cursor estiver sobre o botão

```
tooltip: 'Atualizar',
```

6. b) Definição do ícone que será exibido no botão.

```
child: Icon(Icons.arrow_forward_ios),
),
```

7. A partir de agora, será tratada a parte visual do aplicativo, também feita em código. Na linha a seguir, é escolhida a cor dos detalhes da aplicação.

```
primarySwatch: Colors.deepPurple,
```

8. Definição do texto exibido na página inicial do aplicativo.

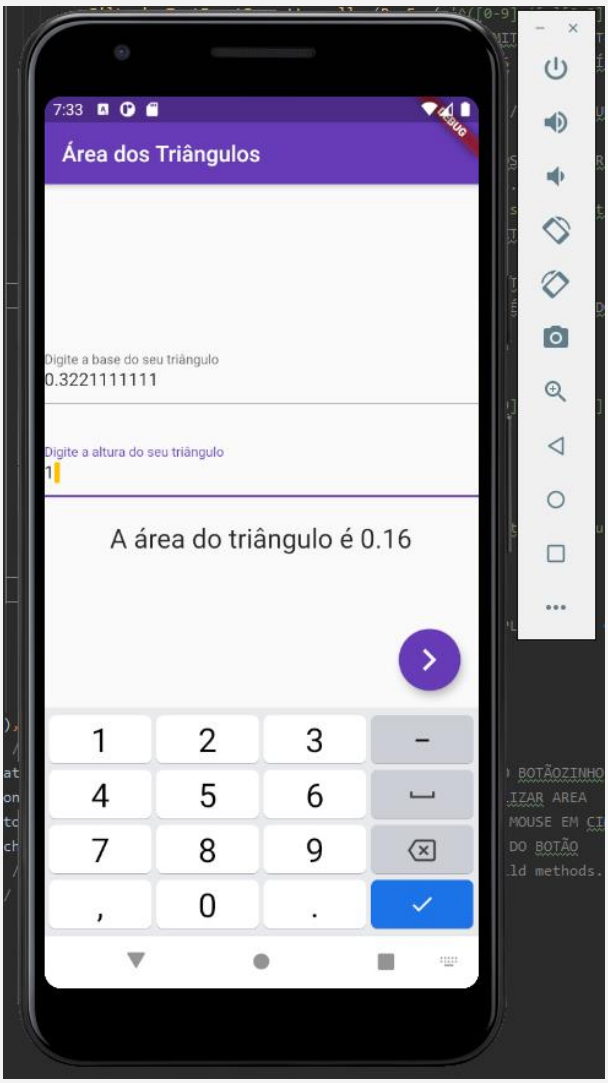
```
home: MyHomePage(title: 'Área dos Triângulos'),
```

9. E, por fim, designa-se um nome ao aplicativo.

```
title: 'Triângulo',
```

Todas as demais partes do código, não foram feitas por nós. Elas já se encontravam na abertura do projeto.

Para encerrar esse tutorial, a seguir se encontra uma foto que mostra o aplicativo feito exibindo o resultado da área do triângulo.



Projeto de Sistemas

Alexandre Siqueira Souza Costa - 179048X
Davi Fagundes Ferreira da Silva - 179082X
Fernanda da Silva Rezende - 1790366
Gabriela Alves Silva de Carvalho - 1990799
Juliana Almeida Santos - 1790048
Júlio Cesar Oliveira da Silva - 1790021
Marina Cova Lacerda - 1690264
Rayssa Oliveira Santos - 1790153
Rebecca Loiola Messali - 179034X
Reginaldo Beserra de Lima Filho - 1690884
Stefany Tam Pereira Mendes – 1790251

Cubatão, 2020

“Acesso ao GPS”

Flutter

A aplicação que permite acesso ao GPS do dispositivo móvel em que está rodando é feita a partir de dois arquivos distintos: a página padrão, a *main.dart*; e a página onde estabelece-se a conexão com o sistema de posicionamento global, a qual chamamos de *map.page.dart*. Ambas serão apresentadas separadamente de forma completa, para só após os processos de construção do código serem explicados a partir de recortes.

1 map.page.dart

```
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
class MapPage extends StatefulWidget {
  @override
  _MapPageState createState() => _MapPageState();
}

class _MapPageState extends State<MapPage> {
  GoogleMapController mapController;
  Position posicao = Position();

Future<Position>getLocation() async{
    Position = await
Geolocator.getCurrentPosition(desiredAccuracy:LocationAccuracy.
high);
    posicao = position;
    marker.add(
      Marker(markerId: MarkerId("Eu"),
        draggable: false,
        position: LatLng(posicao.latitude, posicao.longitude),
        infoWindow: InfoWindow(title: "Eu")
      )
    );
    setState(() {});
  }

  List<Marker> marker = [];
  @override
  void initState() {
    getLocation();
    super.initState();
  }
```

```
@override
  Widget build(BuildContext context) {
    return Container(
      child:
      GoogleMap(
        initialCameraPosition: CameraPosition(
          target: LatLng(posicao.latitude, posicao.longitude),
          zoom: 16.0,
        ),
        markers: Set.from(marker),
        onMapCreated: (mapController) {

        },
      ),
    );
  }
}
```

i Passo a passo

1. Definição da classe que receberá as coordenadas do usuário.

```
class _MapPageState extends State<MapPage> {
  GoogleMapController mapController;
  Position posicao = Position();

Future<Position>getLocation() async{
    Position = await
Geolocator.getCurrentPosition(desiredAccuracy:Lo
cationAccuracy.high);
    posicao = position;
    marker.add(
      Marker(markerId: MarkerId("Eu"),
        draggable: false,
        position: LatLng(posicao.latitude,
posicao.longitude),
        infoWindow: InfoWindow(title: "Eu")
      )
    );
    setState(() {});
  }

  List<Marker> marker = [];
  @override
  void initState() {
    getLocation();
    super.initState();
  }
```

1. a) Linha que faz a conexão com o Google Maps.

```
GoogleMapController mapController;
```

1. b) Definição a função que pega a latitude e a longitude do usuário. Ela não se atualiza sozinha, ou seja, se o usuário mudar de local, mas ela não for chamada novamente, os dados armazenados continuarão os mesmos.

```
Future<Position>getLocation() async{
   Position = await
Geolocator.getCurrentPosition(desiredAccuracy:Lo
cationAccuracy.high);
   posicao = position;
   marker.add(
      Marker(markerId: MarkerId("Eu"),
         draggable: false,
         position: LatLng(posicao.latitude,
posicao.longitude),
         infoWindow: InfoWindow(title: "Eu")
      )
   );
   setState(() {});
 }
```

1. c) Definição do marcador de posição vermelho no mapa exatamente nos pontos em que o usuário se encontra.

```
 List<Marker> marker = [];
@override
void initState() {
 getLocation();
 super.initState();
}
```

2. Definição do método onde há a construção do mapa

```
@override
 Widget build(BuildContext context) {
  return Container(
    child:
    GoogleMap(
     initialCameraPosition: CameraPosition(
      target: LatLng(posicao.latitude,
posicao.longitude),
       zoom: 16.0,
      ),
      markers: Set.from(marker),
      onMapCreated: (mapController) {

      },
    ),
  );
 }
```

2. a) Recebe a posição inicial, a partir da latitude e longitude, como pontos coordenados no plano.

```
initialCameraPosition: CameraPosition(
target: LatLng(posicao.latitude, posicao.longitude),
```

2. b) Estipula o zoom do mapa.

```
zoom: 16.0,
```

2. c) Nesse caso o mapa serve apenas para mostrar a localização, mas caso fosse desejado atribuir uma outra função a ele, seria na linha a seguir

```
onMapCreated: (mapController) {

    },
```

2 main.dart

```
import 'package:app_mapa/map.page.dart';
import 'package:flutter/material.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: 'Flutter Demo',
   debugShowCheckedModeBanner: false,
   theme: ThemeData(

    primarySwatch: Colors.blue,

    visualDensity: VisualDensity.adaptivePlatformDensity,
   ),
   home: MapPage(),
  );
 }
}
```

i Passo a passo

Grande parte das coisas contidas no maind.dart já vem com a inicialização da aplicação. A seguir será mostrado apenas as modificações realizadas:

1. Linha que estipula a exibição do banner de debug. Como está falso, o banner não será exibido na aplicação.

```
debugShowCheckedModeBanner: false,
```

2. Linha responsável por chamar o arquivo map.page.dart.

```
home: MapPage(),
```

3 Resultado

Projeto de Sistemas

Alexandre Siqueira Souza Costa - 179048X
Davi Fagundes Ferreira da Silva - 179082X
Fernanda da Silva Rezende - 1790366
Gabriela Alves Silva de Carvalho - 1990799
Juliana Almeida Santos - 1790048
Júlio Cesar Oliveira da Silva - 1790021
Marina Cova Lacerda - 1690264
Rayssa Oliveira Santos - 1790153
Rebecca Loiola Messali - 179034X
Reginaldo Beserra de Lima Filho - 1690884
Stefany Tam Pereira Mendes – 1790251

Cubatão, 2020

“Acesso à agenda”

Flutter

A aplicação que permite acesso a agenda do dispositivo móvel em que está rodando é feita a partir de dois passos: a importação de pacotes cruciais para o funcionamento do programa e configurações na linguagem Flutter, as quais já estamos acostumados. Ambos serão apresentados de forma completa, para, só após, os processos de construção do código serem explicados a partir de recortes.

1 Código completo

```
import 'package:flutter/material.dart';
import 'package:contacts_service/contacts_service.dart';
import 'package:permission_handler/permission_handler.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: 'Flutter Demo',
   theme: ThemeData(

    primarySwatch: Colors.blue,

    visualDensity: VisualDensity.adaptivePlatformDensity,
   ),
   home: MyHomePage(title: 'Flutter Contact List'),
  );
 }
}

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

final String title;

 @override
 _MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
 List<Contact> contacts = [];
 List<Contact> contactsFiltered = [];
 TextEditingController searchController = new
TextEditingController();

 @override
 void initState() {
  // TODO: implement initState
   super.initState();
   getPermissions();
 }
getPermissions() async {
  if (await Permission.contacts.request().isGranted) {
   getAllContacts();
   searchController.addListener(() {filterContacts();});
  }
 }
 getAllContacts() async{
  List<Contact> _contacts = (await
ContactsService.getContacts(withThumbnails: false)).toList();
  setState(() {
   contacts = _contacts;
  });
 }
 String flattenPhoneNumber(String phoneStr)
 {
  return phoneStr.replaceAllMapped(RegExp(r'^(\+)|\D'), (Match
m)
  {return m[0] == "+" ? "+" : " ";}
  );
 }

 filterContacts(){
  List<Contact> _contacts = [];
  _contacts.addAll(contacts);
  if(searchController.text.isNotEmpty)
  {
   _contacts.retainWhere((contact) {
    String searchTerm = searchController.text.toLowerCase();
    String searchTermFlattened =
flattenPhoneNumber(searchTerm);
    String contactName = contact.displayName.toLowerCase();
    bool match = contactName.contains(searchTerm);

    if(searchTermFlattened.isEmpty){return false;}

    if(match == true){return true;}

    var phone = contact.phones.firstWhere((phn) {
     String phnFlattened = flattenPhoneNumber(phn.value);
     return phnFlattened.contains(searchTermFlattened);
    }, orElse: () => null);
    return phone != null;
   });
  }
  setState(() {
  contactsFiltered = _contacts;
  });
 }
 @override
 Widget build(BuildContext context) {
  bool isSearching = searchController.text.isNotEmpty;
```

```
return Scaffold(
    appBar: AppBar(

     title: Text(widget.title),
    ),
    body: Container(
     padding: EdgeInsets.all(20),
     child: Column(
      children: <Widget>[
       Container(
        child: TextField(
         controller: searchController,
         decoration: InputDecoration(labelText: "Procurar",
prefixIcon: Icon(Icons.search, color: Colors.blueAccent,),),
         ),
       ),
      Expanded(child:
        ListView.builder(
         shrinkWrap: true,
         itemCount: isSearching == true ?
contactsFiltered.length : contacts.length,
         itemBuilder: (context, index){
          Contact contact = isSearching == true ?
contactsFiltered[index] : contacts[index];
          return ListTile(
           title: Text(contact.displayName),
           subtitle: Text(contact.phones.length > 0 ?
contact.phones.elementAt(0).value : "Sem telefone"),
           leading:  CircleAvatar(child:
Text(contact.initials())),
          );
         },
        ),
      ),
      ],
     ),
    )
   );
  }
}
```

2 Passo a passo

1. Importação dos pacotes fundamentais para o funcionamento do programa

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

- Informações adicionais:
- O pacote contacts_service.dart‘ é fundamental, pois é responsável por pegar os contatos e as informações deles, além de poder apagar e modificar os contatos.
- O pacote 'package:permission_handler/permission_handler.dart é o responsável por pedir permissão ao usuário para acessar os contatos.

2. Declaração das variáveis onde os contatos serão armazenados em forma de listas, as quais serão utilizadas novamente em breve:

```
class _MyHomePageState extends
State<MyHomePage> {
 List<Contact> contacts = [];
 List<Contact> contactsFiltered = [];
 TextEditingController searchController = new
TextEditingController();
```

- Informações adicionais:
- A primeira lista (List<Contact> contacts = [];) é responsável apenas pelo armazenamento dos contatos normalmente;
- A segunda lista (List<Contact> contactsFiltered = [];) é responsável pelo armazenamento dos contatos após a filtragem que será realizada mais na frente.

3. Declaração dos métodos initState(), o qual define quando a aplicação será iniciada, getPermissions(), o qual concede as permissões e getAllContacts().

```
@override
 void initState() {
  // TODO: implement initState
   super.initState();
   getPermissions();
 }
getPermissions() async {
  if (await Permission.contacts.request().isGranted) {
   getAllContacts();
   searchController.addListener(() {filterContacts();});
  }
 }

 getAllContacts() async{
  List<Contact> _contacts = (await
ContactsService.getContacts(withThumbnails:
false)).toList();
  setState(() {
   contacts = _contacts;
  });
```

- Informações adicionais:
- Quando a função assíncrona getPermissions() é dada, a função getAllContacts() será executada, o que atribuirá um *listener* ao controller serachController;
- A função getAllContacts() é assíncrona e é responsável por pegar todos os contatos, sem as fotos, e convertê-los para o tipo lista, para, assim, ser possível o armazenamento na variável local _contacts.

4. Declaração da função filterContacts(), que é responsável pela filtragem da lista.

```
filterContacts(){
  List<Contact> _contacts = [];
  _contacts.addAll(contacts);
  if(searchController.text.isNotEmpty)
  {
   _contacts.retainWhere((contact) {
    String searchTerm =
searchController.text.toLowerCase();
    String searchTermFlattened =
flattenPhoneNumber(searchTerm);
    String contactName =
contact.displayName.toLowerCase();
    bool match = contactName.contains(searchTerm);
 if(searchTermFlattened.isEmpty){return false;}

    if(match == true){return true;}

    var phone = contact.phones.firstWhere((phn) {
     String phnFlattened =
flattenPhoneNumber(phn.value);
     return
phnFlattened.contains(searchTermFlattened);
    }, orElse: () => null);
    return phone != null;
   });
  }
  setState(() {
  contactsFiltered = _contacts;
  });
 }
```

5. Declaração da função flattenPhoneNumber(), que é responsável pela formatação dos números de telefone.

```
String flattenPhoneNumber(String phoneStr)
 {
  return
phoneStr.replaceAllMapped(RegExp(r'^(\+)|\D'), (Match
m)
  {return m[0] == "+" ? "+" : " ";}
  );
 }
```

6. Atribuição de valores à variável *isSearching*, que exibirá a lista filtrada. Caso a lista filtrada não exista, ela exibirá a lista completa, exibindo a filtrada apenas se existir.

```
Expanded(child:
        ListView.builder(
         shrinkWrap: true,
         itemCount: isSearching == true ?
contactsFiltered.length : contacts.length,
         itemBuilder: (context, index){
          Contact contact = isSearching == true ?
contactsFiltered[index] : contacts[index];
          return ListTile(
           title: Text(contact.displayName),
           subtitle: Text(contact.phones.length > 0 ?
contact.phones.elementAt(0).value : "Sem telefone"),
           leading:  CircleAvatar(child:
Text(contact.initials())),
```

- Informações adicionais:
- Caso não fosse verificado o número do telefone, o programa daria erro.

3 Adendo

Caso venha a ser necessário, aqui estão as informações exatas utilizadas no aplicativo referentes ao pubspec.yaml.

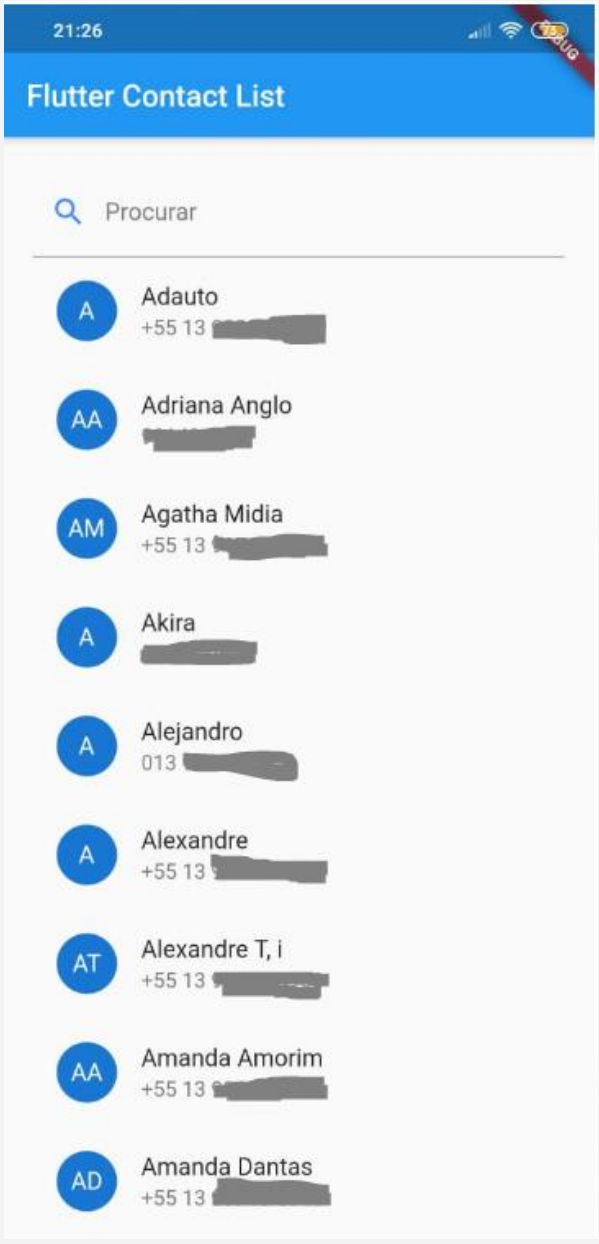
```
dependencies:
  flutter:
    sdk: flutter
  contacts_service: ^0.4.6
  permission_handler: ^5.0.1+1
```

Para você que não sabe, o pubspec.yaml é um arquivo que introduz no programa algum pacote que pode a vir a ser necessário. Ele se difere da simples importação de pacotes porque, nesse último caso, o pacote é interno, enquanto no pubspec.yaml o pacote é externo.

4 Resultado

A fim de preservar a integridade da agenda de contatos do membro do grupo que se ofereceu para rodar o programa em seu dispositivo móvel, a imagem contendo o exemplo está com os números de telefone censurados.

Entretanto, acreditamos que as informações expostas são suficientes para proporcionar o entendimento do resultado da experiência proposta.



Projeto de Sistemas

Alexandre Siqueira Souza Costa - 179048X
Davi Fagundes Ferreira da Silva - 179082X
Fernanda da Silva Rezende - 1790366
Gabriela Alves Silva de Carvalho - 1990799
Juliana Almeida Santos - 1790048
Júlio Cesar Oliveira da Silva - 1790021
Marina Cova Lacerda - 1690264
Rayssa Oliveira Santos - 1790153
Rebecca Loiola Messali - 179034X
Reginaldo Beserra de Lima Filho - 1690884
Stefany Tam Pereira Mendes – 1790251

Cubatão, 2020

“Lista de tarefas”

Flutter

A aplicação que permite a criação de uma lista de tarefas com ligação a um banco de dados é feita a partir da programação no *main.dart* e de uma classe chamada *TodoList()*, a qual será responsável pela conexão ao banco de dados. Ambos serão apresentados de forma completa, para, só após, informações a respeito do código serem expostas, a fim de contribuir para o entendimento dos processos de criação.

1 main.dart()

```
import 'package:app_crud_2/loading.dart';
import 'package:app_crud_2/todolist.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';

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())));
     }
     if (snapshot.connectionState == ConnectionState.waiting) {
      return Loading();
     }
     return MaterialApp(
       debugShowCheckedModeBanner: false,
       home: TodoList(),
       theme: ThemeData(
        scaffoldBackgroundColor: Colors.white,
        primarySwatch: Colors.cyan,
       ));
    });
 }
}
```

i Informações

O conteúdo encontrado no *main.dart()* é o mesmo encontrado nos tutoriais anteriores, exceto um único componente, o qual será explicado em seguida:

1. O componente *FutureBuilder()* é responsável por criar um *widget* de acordo com o último snapshot disponível. O *builder* não pode ser vazio:

```
Widget build(BuildContext context) {
  return FutureBuilder(
    future: Firebase.initializeApp(),
    builder: (context, snapshot) {
     if (snapshot.hasError) {
      return Scaffold(
        body: Center(child: Text(snapshot.error.to
String())));
     }
```

2 TodoList()

```
import 'package:app_crud_2/loading.dart';
import 'package:app_crud_2/model/todo.dart';
import 'package:app_crud_2/services/database_services.dart';
import 'package:flutter/material.dart';

class TodoList extends StatefulWidget {
 @override
 _TodoListState createState() => _TodoListState();
}

class _TodoListState extends State<TodoList> {
 bool isComplete = false;
 TextEditingController textTitleController = TextEditingController();
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text(
     "Coisas a Fazer",
     style: TextStyle(
       fontSize: 30, fontWeight: FontWeight.bold, color: Colors.w
hite),
    ),
    backgroundColor: Colors.blue,
   ),
   body: SafeArea(
    child: StreamBuilder<List<Todo>>(
      stream: DatabaseService().listTarefas(),
      builder: (context, snapshot) {
       if (!snapshot.hasData) {
        return Loading();
       }
```

```
List<Todo> tarefas = snapshot.data;
return Padding(
 padding: EdgeInsets.all(25),
 child: Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
   SizedBox(height: 20),
   Expanded(
    child: ListView.separated(
     separatorBuilder: (context, index) => Divider(),
     shrinkWrap: true,
     itemCount: tarefas.length,
     itemBuilder: (context, index) {
      return Dismissible(
       key: Key(tarefas[index].title),
       background: Container(
        padding: EdgeInsets.only(left: 20),
        alignment: Alignment.centerLeft,
        child: Icon(Icons.delete),
        color: Colors.red,
       ),
       onDismissed: (direction) async {
        await DatabaseService()
          .removeTarefa(tarefas[index].uid);
       },
       child : ListTile(
         onTap: () {
          DatabaseService()
            .completeTarefa(tarefas[index].uid);
         },
  leading : Container(
            padding : EdgeInsets.all(2),
            height: 30,
            width: 30,
            decoration: BoxDecoration(
             color: Theme.of(context).primaryColor,
             shape: BoxShape.circle,
            ),
            child : tarefas[index].isComplete
              ? Icon(
                Icons.check,
                color : Colors.white,
               )
              : Container(),
           ),
           title: Text(
            tarefa:[index].title,
            style: TextStyle(
              fontSize: 20,
              fontWeight : FontWeight.w500),
           )),
      );
     },
```

```
              ),
             )
            ],
           ),
          );
         }),
    ),
    floatingActionButtonLocation: FloatingActionButtonLocation.c
enterFloat,

floatingActionButton: FloatingActionButton(
     child: Icon(Icons.add),
     backgroundColor: Theme.of(context).primaryColor,
     onPressed: () {
      showDialog(
       context: context,
       child: SimpleDialog(
        contentPadding: EdgeInsets.symmetric(
         vertical: 20,
         horizontal: 25,
        ),
        shape: RoundedRectangleBorder(
         borderRadius: BorderRadius.circular(20),
        ),
        title: Row(
         children: [
          Text(
           "Adicionar Tarefa",
           style: TextStyle(fontSize: 20),
          ),
          Spacer(),
          IconButton(
           icon: Icon(
            Icons.cancel,
            color: Colors.grey,
            size: 30,
           ),
           onPressed: () => Navigator.pop(context),
          )
         ],
        ),

        children: [
         Divider(),
         TextFormField(
          controller: textTitleController,
          style: TextStyle(fontSize: 20, height: 1.5),
          autofocus: true,
          decoration: InputDecoration(
            hintText: "Fazer o app número 4",
            border: InputBorder.none),
         ),

         SizedBox(
          height: 20,
         ),
```

```
                    SizedBox(
              height: 50,
              width: MediaQuery.of(context).size.width,
              child: FlatButton(
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(20),
                ),

                onPressed: () async {
                  if (textTitleController.text.isNotEmpty) {
                    await DatabaseService()
                        .createnewTarefa(textTitleController.text.trim());
                    setState(() {
                      textTitleController.text = "";
                    });
                    Navigator.pop(context);
                  }
                },
                child: Text("Adicionar à lista")),
            )
          ],
        ),
      );
    },
  ),
 );
 }
}
```

i Informações

O conteúdo encontrado no *Todo.List()* é refere-se, basicamente, na definição do layout em questão e a conexão ao banco de dados.

No entanto, as funções da classe *DatabaseService()* estão-se em um arquivo denominado *DatabaseService()*, localizado em outra pasta, a *services*.

Por fim, o modelo *Todo*, o qual é uma classe, fica na pasta *model*.

ii DatabaseService

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:app_crud_2/model/todo.dart';

class DatabaseService {
 CollectionReference tarefaCollection =
    FirebaseFirestore.instance.collection("Tarefas");

 Future createnewTarefa(String title) async {
      return await tarefaCollection.add({"title": title, "isComplete":
false});
 }

 Future completeTarefa(uid) async {
  await tarefaCollection.doc(uid).update({"isComplete": true});
 }

 Future removeTarefa(uid) async {
  await tarefaCollection.doc(uid).delete();
 }

 List<Todo> tarefafromFirestore(QuerySnapshot snapshot) {
  if (snapshot != null) {
   return snapshot.docs.map((e) {
    return Todo(
       isComplete: e.data()["isComplete"],
       title: e.data()["title"],
       uid: e.id);
   }).toList();
  } else {
   return null;
  }
 }

 Stream<List<Todo>> listTarefas() {
  return tarefaCollection.snapshots().map(tarefafromFirestore);
 }
}
```

iii Model

```
 class Todo{
  String uid;
  String title;
  bool isComplete;

  Todo({this.uid,this.title,this.isComplete});
 }
```

3 Adendo

> É muito importante que não seja esquecido de adicionar as informações a respeito das versões do *cloud firebase* no *pubspec.yaml*.

A seguir encontra-se a maneira como o fizemos:

```
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: ^0.1.3
        cloud_firestore: ^0.16.0
        firebase_core: ^0.7.0
```

> Por fim , é importante lembrar que as instruções de como integrar o *firebase* no projeto são obtidas a partir da criação do próprio banco no *firebase*.

> Ademais, caso o programa não rode após a utilização deste guia, é recomendado mudar o campo *minSdkVersion* para 21 no arquivo *build.grandle*, localizado na pasta *app*.

4 Resultado

A seguir, encontram-se uma capturas de tela do aplicativo rodando.

Vale ressaltar que ao iniciar o programa, apresentou-se algumas travas, no entanto, após o aplicativo voltar a funcionar, nenhum outro transtorno foi observado.

Projeto de Sistemas

Alexandre Siqueira Souza Costa - 179048X
Davi Fagundes Ferreira da Silva - 179082X
Fernanda da Silva Rezende - 1790366
Gabriela Alves Silva de Carvalho - 1990799
Juliana Almeida Santos - 1790048
Júlio Cesar Oliveira da Silva - 1790021
Marina Cova Lacerda - 1690264
Rayssa Oliveira Santos - 1790153
Rebecca Loiola Messali - 179034X
Reginaldo Beserra de Lima Filho - 1690884
Stefany Tam Pereira Mendes – 1790251

Cubatão, 2021