INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE SÃO PAULO CAMPUS CUBATÃO

FLUTTER

ALICIA OLIVEIRA DE AGUIAR
ALVARO PRIMITZ
ALLYSON WESLLEY
CAIQUE PEREIRA
DANTON MELLO
GUILHERME FARIAS
LUCAS GOMES
PEDRO DE FRANÇA
RICARDO FERNANDEZ
TIAGO AMARO
VITOR VIEIRA

Cubatão,
Setembro de 2020

HISTÓRICO E PRINCIPAIS CARACTERÍSTICAS DO FRAMEWORK ESCOLHIDO

DEFINIÇÃO:

O Flutter é um framework para o desenvolvimento de aplicações mobile da Google que permite desenvolver aplicações para iOS e Android a partir um código base. Ele é um código aberto e gratuito, com mecanismo de renderização em 2D rápido, com um ótimo desempenho e integração de plataformas móveis, com desenvolvimento rápido e alcance multiplataforma e com ferramentas para a criação de interfaces para multiplataformas.

A primeira versão do Flutter era conhecida com o codinome "Sky" e era executada no sistema operacional Android, baseada em Dark, uma linguagem orientada a objeto também desenvolvida pela Google, anunciada em 2011. Apesar de ser uma linguagem nova, é de fácil aprendizado, pois possui uma sintaxe similar a Java e C#.

CARACTERÍSTICAS DO FLUTTER:

Diversas são as características do Flutter. Dentre elas podemos citar:

Multiplataforma – Podemos desenvolver aplicações com Flutter em qualquer sistema operacional (Windows, Linux e MacOS);

Criação de aplicações nativas a partir de um único código base – Com o Flutter é possível desenvolvermos aplicações nativas para Android e iOS;

Acesso direto aos recursos nativos do sistema – Uma aplicação criada com Flutter possui acesso nativo aos recursos do dispositivo (câmera, wifi, memória, etc);

Maior desempenho – As aplicações criadas com Flutter possuem um maior desempenho quando comparadas ao React Native, por exemplo, pois todo seu código-fonte é transformado em código nativo.

QUAIS SÃO AS VANTAGENS DO FLUTTER?

Aplicativos móveis mais rápidos - Apps desenvolvidos usando Flutter são mais rápidos em comparação com outras frameworks multiplataforma. Por possuir widgets próprios, os aplicativos em Flutter tem uma interface leve que permite ótima experiência ao usuário, além de ser compilado para código nativo.

Suporte oficial da Google - Assim como todas as ferramentas desenvolvidas e disponibilizadas pela Google, Flutter possui suporte oficial.

A mesma interface até em dispositivos antigos - O pacote do SDK de Flutter vem com widgets próprios, portanto dispositivos antigos terão os mesmo widgets que os dispositivos novos, já que não são utilizados os widgets nativos das plataformas.

E QUAIS SÃO AS DESVANTAGENS?

Só suporta mobile - Enquanto seu principal concorrente, o React Native, também é utilizado para desenvolvimento web, Flutter só é compatível com desenvolvimento mobile.

Recente - O beta 2 de Flutter foi anunciado em abril deste ano (2018), portanto Flutter ainda não é muito estável em comparação com React.

Poucas bibliotecas - Flutter é relativamente novo e ainda está em beta, por esse motivo algumas funcionalidades essenciais para o desenvolvimento da aplicação podem não estar prontas ainda, e isso significa que o desenvolvedor precisará implementá-las por conta própria.

TABELA DE ESPECIFICAÇÕES DO FRAMEWORK:

Desenvolvedor	Google
Plataforma	Android, iOS, Google Fuchsia, Web, Linux, macOS e Windows
Lançamento	Maio de 2017
Versão estável	1.20.1 (5 de agosto de 2020)
Linguagem	C, C++, Dart
Gênero(s)	Framework de aplicação
Licença	BSD3- Clause
Página oficial	Flutter.dev

REFERÊNCIAS:

Desenvolvimento Hibrído com Flutter: Prós e Contras - https://medium.com/@devmob/desenvolvimento-h%C3%ADbrido-com-flutter-pr%C3%B3s-e-contras-6f3f422c480c

O que é flutter - https://www.treinaweb.com.br/blog/o-que-e-flutter/

Wikipedia: Flutter - https://pt.m.wikipedia.org/wiki/Flutter#:~:text=9%20Liga%C3%A7%C3%B5es%20externas-,Hist%C3%B3ria,a%20120%20quadros%20por%20segundo

Framework: Conhecendo um pouco mais do Flutter - https://imasters.com.br/framework/conhecendo-um-pouco-mais-flutter

INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE SÃO PAULO CAMPUS CUBATÃO

FLUTTER - INSTALAÇÃO DE IDE

ALICIA OLIVEIRA DE AGUIAR
ALVARO PRIMITZ
ALLYSON WESLLEY
CAIQUE PEREIRA
DANTON MELLO
GUILHERME FARIAS
LUCAS GOMES
PEDRO DE FRANÇA
RICARDO FERNANDEZ
TIAGO AMARO
VITOR VIEIRA

Cubatão,
Setembro de 2020

AMBIENTE DE DESENVOLVIMENTO INTEGRADO

IDE (Integrated Development Environment, ou, em português, Ambiente de Desenvolvimento Integrado) consiste em um software responsável por reunir ferramentas de apoio ao desenvolvimento de códigos, reunindo funcionalidades em uma interface gráfica de usuários, facilitando assim o processo de programação.
Uma IDE geralmente apresentará as seguintes funcionalidades:

- **Editor de código-fonte:** Editor de texto que apresenta funcionalidades para que haja uma maior facilidade ao escrever códigos, funções essas como: Destaque da sintaxe com indicadores visuais; Recurso de preenchimento automático; Verificação de bugs durante a criação.
- **Automação de compilação local:** Execução de tarefas como compilação de código-fonte em código binário, criação de pacotes de código binário e execução de testes automatizados.
- **Debugger:** Funcionalidade que permite o teste de outros programas e mostrar graficamente a localização do bug no código original.

Resumidamente, as IDE's são responsáveis por facilitar e agilizar o processo de programação de softwares. Dado isso, posteriormente iremos destacar as principais IDE's para a programação em flutter.

PRINCIPAIS IDES PARA PROGRAMAÇÃO EM FLUTTER

Visual Studio Code (VSCode)

Um dos maiores IDEs no mercado e financiado pela Microsoft. Essa IDE é uma das preferidas por boa parte do público de programação devido ao seu crescimento e suporte da equipe responsável pela IDE. Alguns de seus prós são o preenchimento de códigos, exibição dos erros no código em tempo real e também a facilidade visual de identificação de sintaxe.

Eclipse

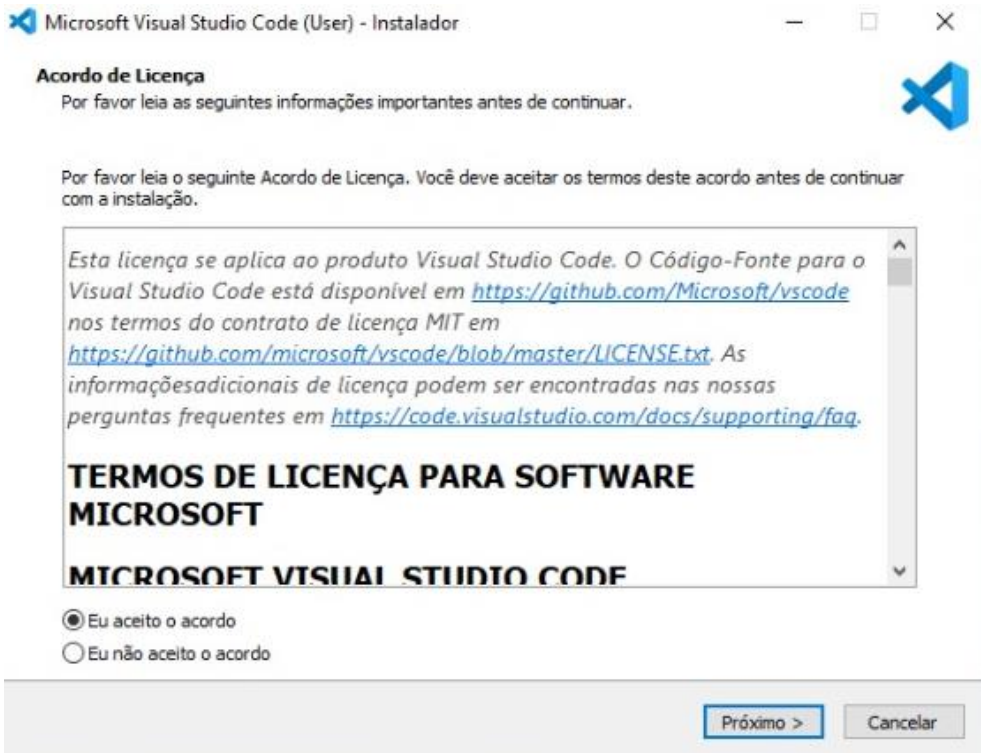
A IDE eclipse é um plugin do GitHub ”open-source”, isto é, qualquer um pode modificar e distribuir o código fonte. Seus pontos forte são o destaque promovido na sintaxe a fim de facilitar a busca pelo erro e a possibilidade de rodar o código diretamente pela IDE.

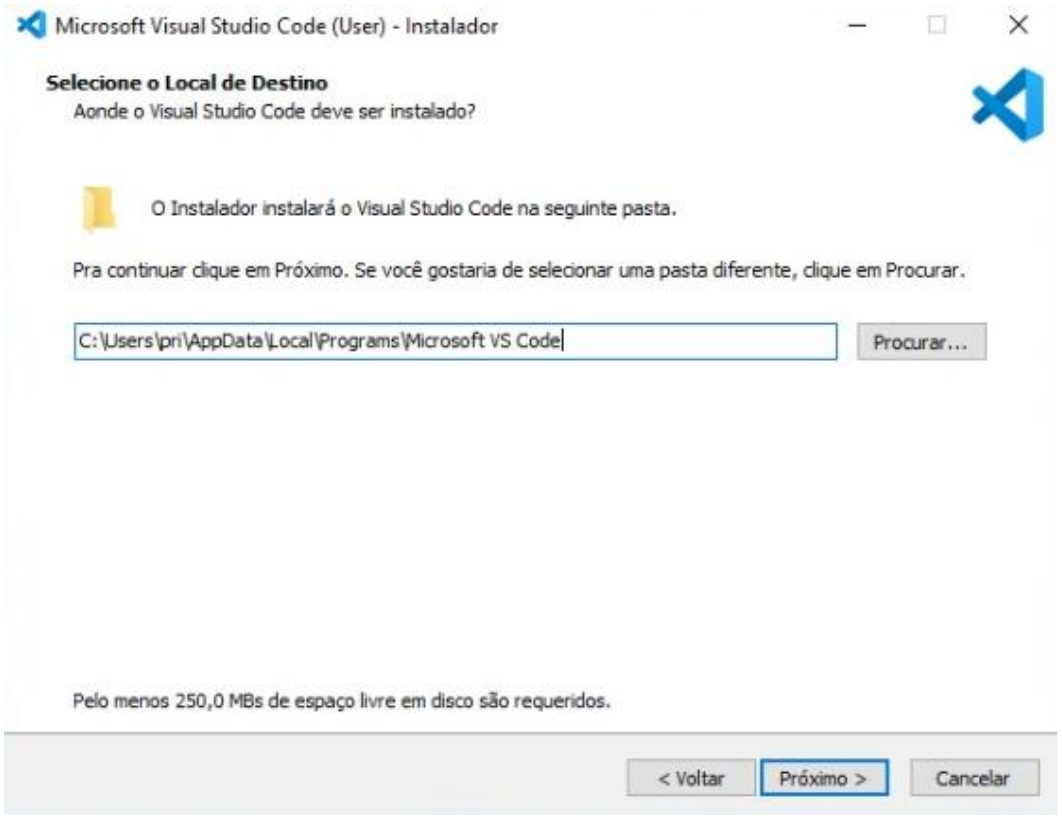
Android Studio

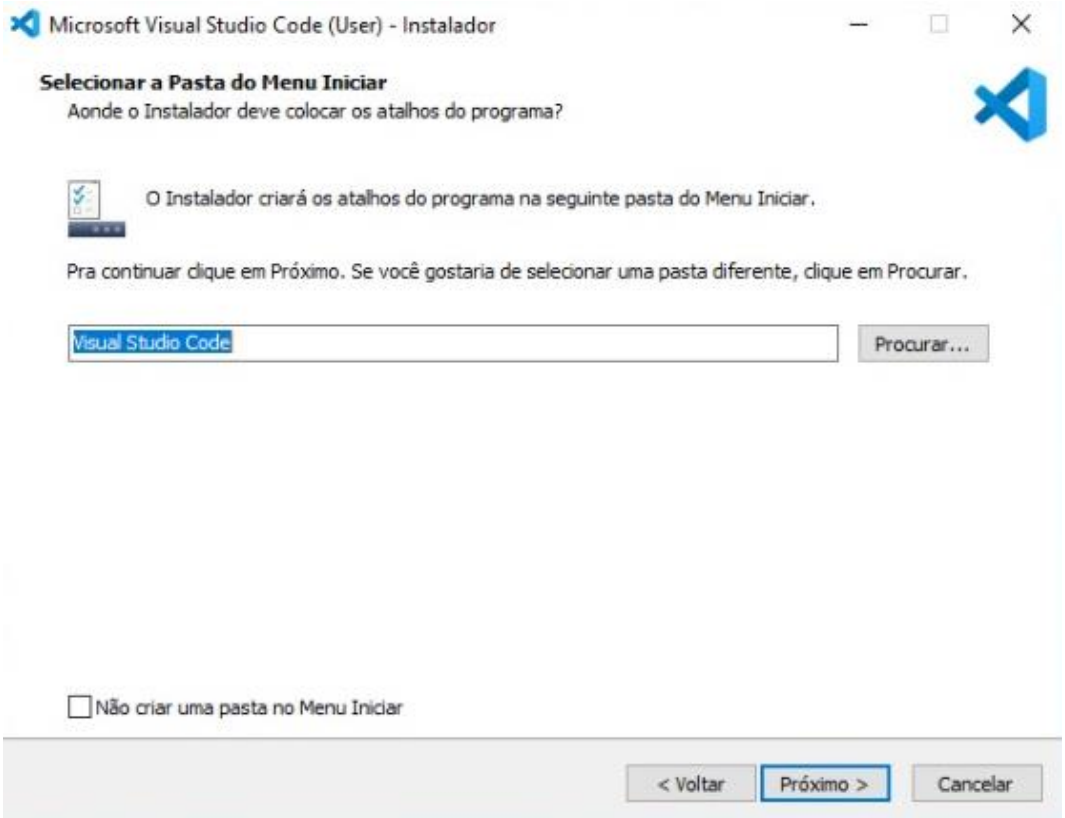
O android Studio é uma IDE projetada principalmente para o desenvolvimento na plataforma android. Baseado no InteliJ Community Version, possui o mesmo objetivo do eclipse + adt (Android Developer Tools), ele provê um ambiente de desenvolvimento, debug, testes e profile multiplataforma para Android.

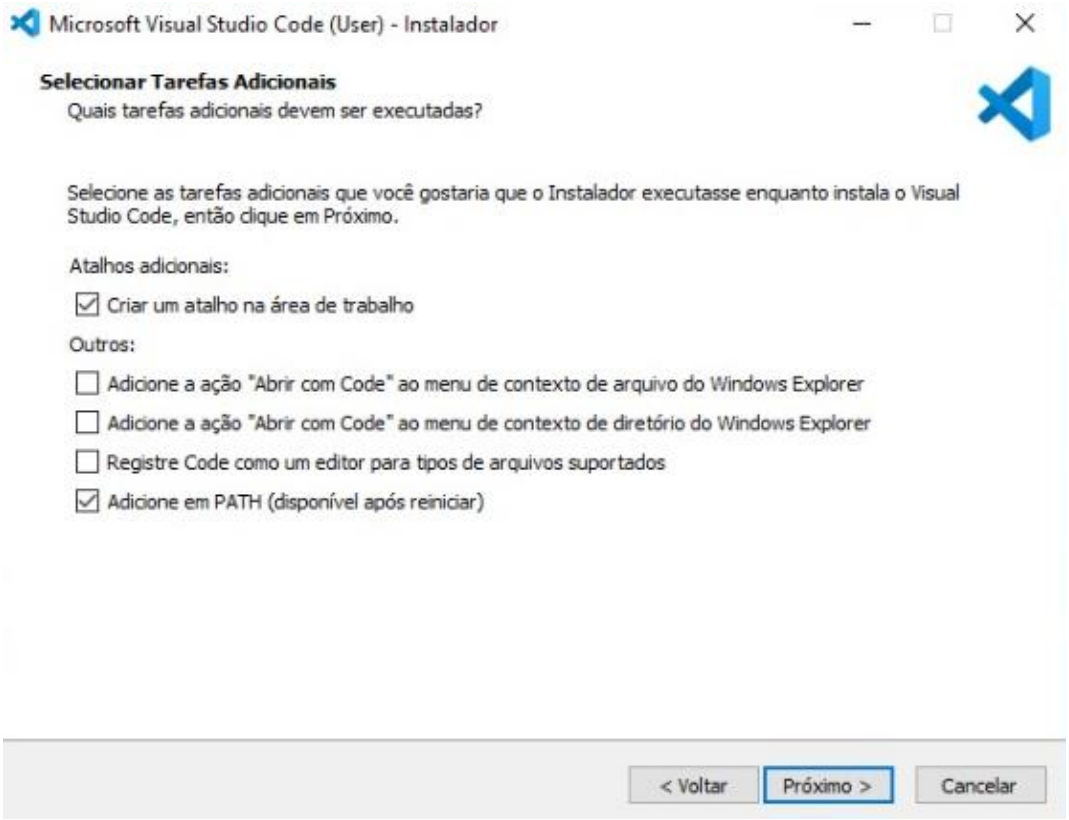
INSTALANDO O VISUAL STUDIO CODE

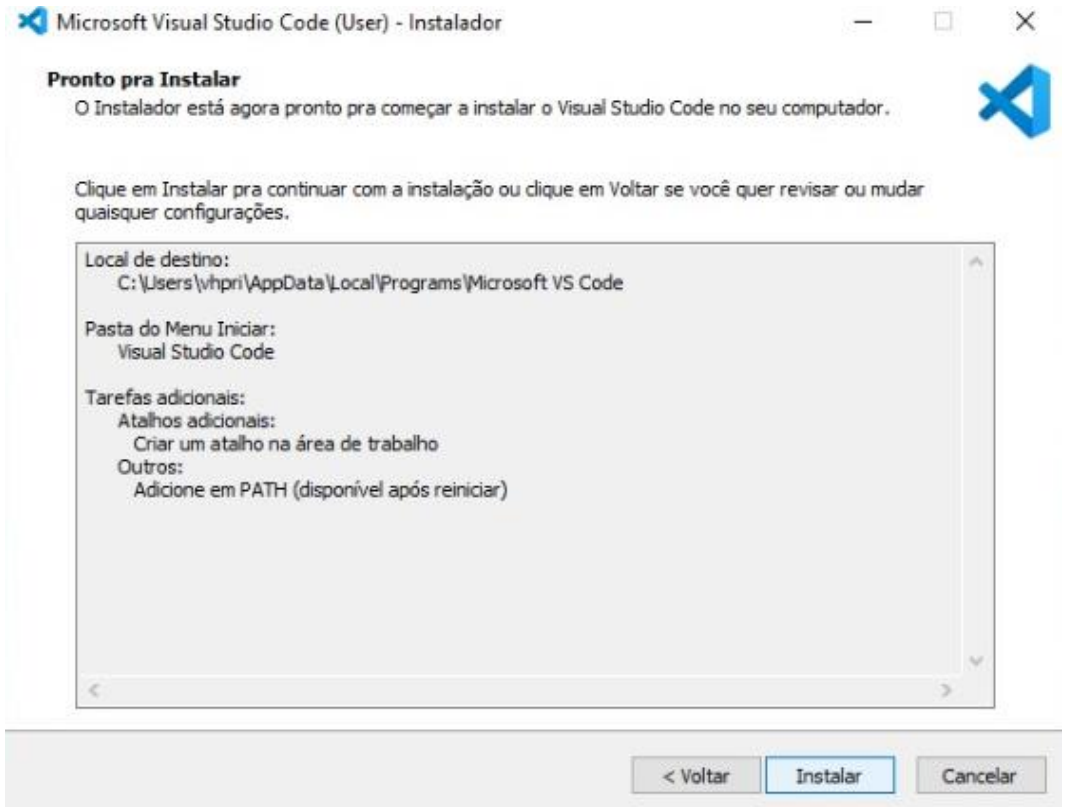
O visual studio code pode ser obtido no site da plataforma. Depois de feito o download, a instalação pode ser concluída seguindo os seguintes passos:

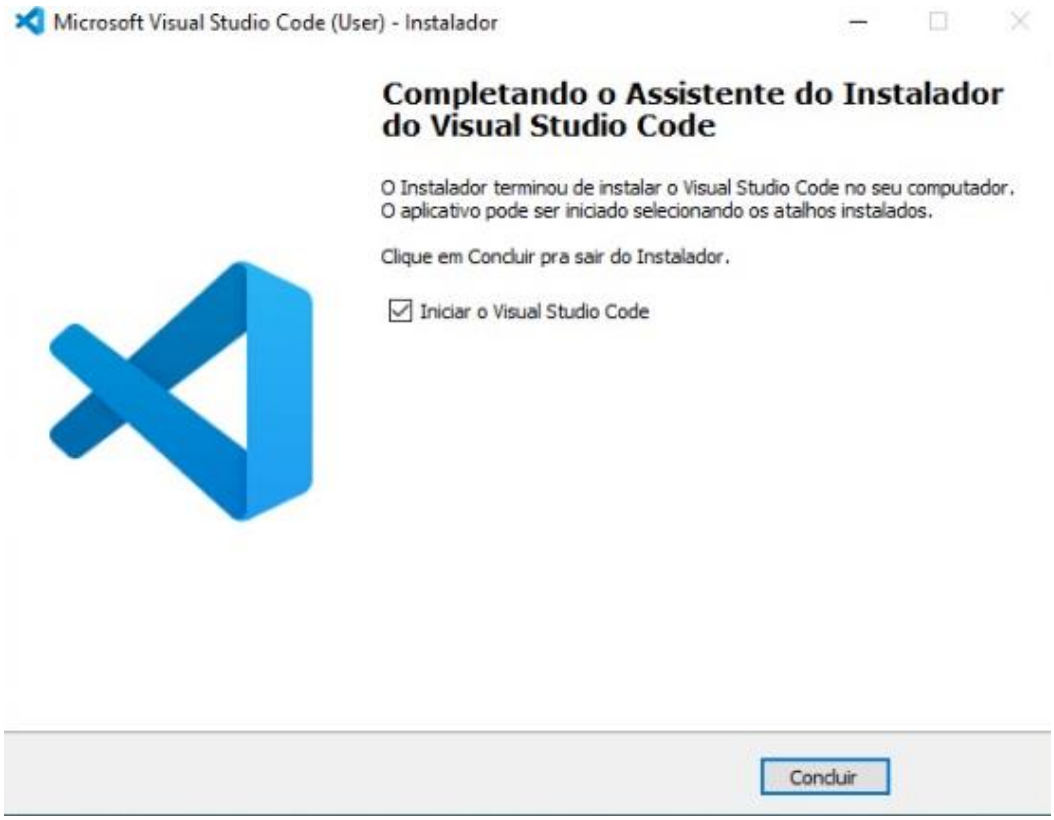



Microsoft Visual Studio Code (User) - Instalador
Selecione o Local de Destino
Aonde o Visual Studio Code deve ser instalado?
O Instalador instalará o Visual Studio Code na seguinte pasta.
Pra continuar clique em Próximo. Se você gostaria de selecionar uma pasta diferente, clique em Procurar.
C:\Users\pri\AppData\Local\Programs\Microsoft VS Code
Procurar...
Pelo menos 250,0 MBs de espaço livre em disco são requeridos.
< Voltar
Próximo >
Cancelar

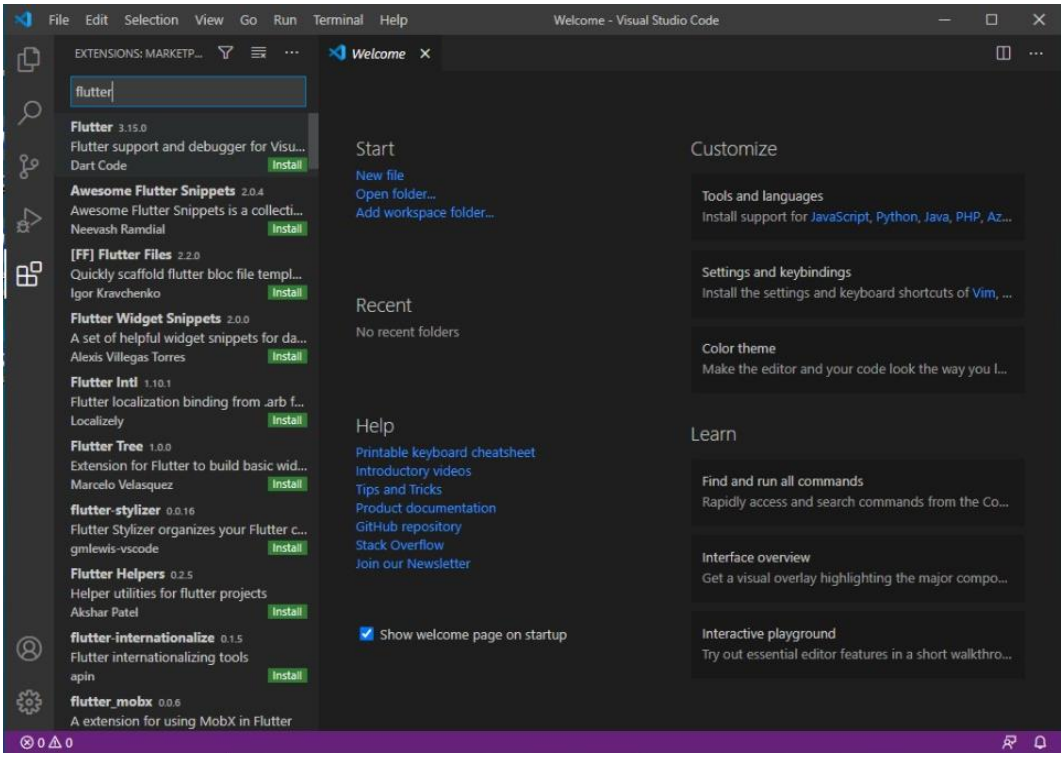

Microsoft Visual Studio Code (User) - Instalador
Selecionar a Pasta do Menu Iniciar
Aonde o Instalador deve colocar os atalhos do programa?
O Instalador criará os atalhos do programa na seguinte pasta do Menu Iniciar.
Pra continuar clique em Próximo. Se você gostaria de selecionar uma pasta diferente, clique em Procurar.
Visual Studio Code
Procurar...
Não criar uma pasta no Menu Iniciar
< Voltar
Próximo >
Cancelar

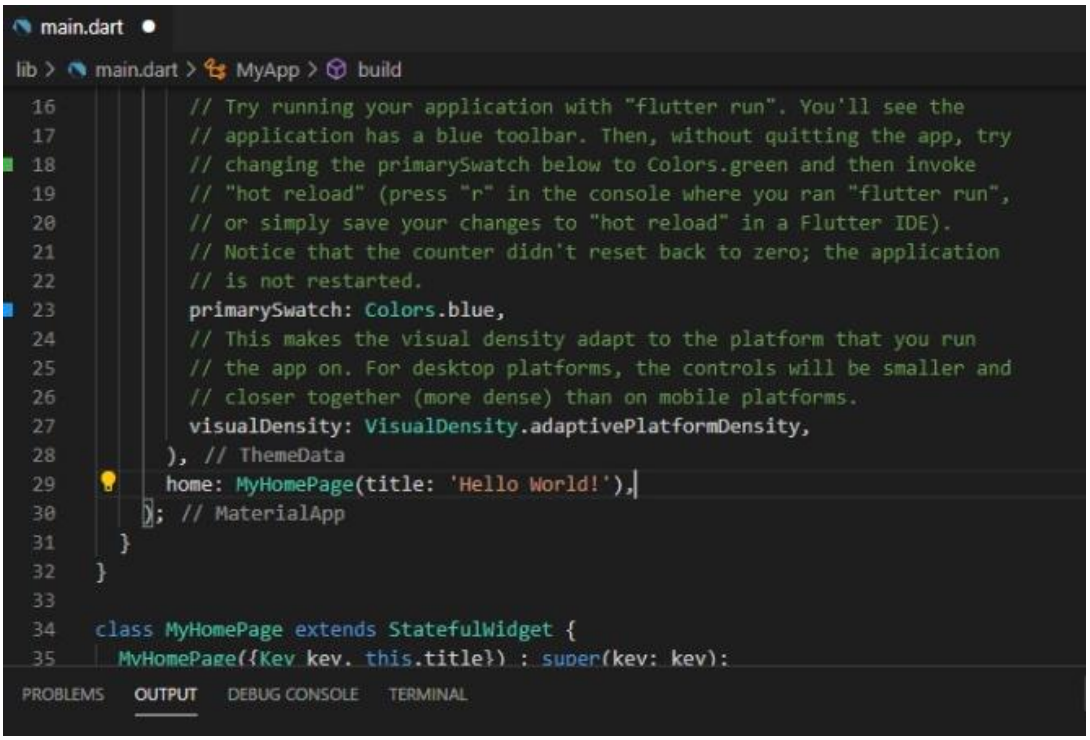

Microsoft Visual Studio Code (User) - Instalador
Selecionar Tarefas Adicionais
Quais tarefas adicionais devem ser executadas?
Selecione as tarefas adicionais que você gostaria que o Instalador executasse enquanto instala o Visual Studio Code, então clique em Próximo.
Atalhos adicionais:
Criar um atalho na área de trabalho
Outros:
Adicione a ação "Abrir com Code" ao menu de contexto de arquivo do Windows Explorer
Adicione a ação "Abrir com Code" ao menu de contexto de diretório do Windows Explorer
Registre Code como um editor para tipos de arquivos suportados
Adicione em PATH (disponível após reiniciar)
< Voltar
Próximo >
Cancelar


Microsoft Visual Studio Code (User) - Instalador
Pronto pra Instalar
O Instalador está agora pronto pra começar a instalar o Visual Studio Code no seu computador.
Clique em Instalar pra continuar com a instalação ou clique em Voltar se você quer revisar ou mudar quaisquer configurações.
Local de destino:
C:\Users\vhpri\AppData\Local\Programs\Microsoft VS Code
Pasta do Menu Iniciar:
Visual Studio Code
Tarefas adicionais:
Atalhos adicionais:
Criar um atalho na área de trabalho
Outros:
Adicione em PATH (disponível após reiniciar)
< Voltar
Instalar
Cancelar



INSTALANDO O FLUTTER E A LINGUAGEM DART NO VISUAL STUDIO E REALIZANDO O PRIMEIRO HELLO WORLD





Utilizamos a tool bar superior para visualizar o título “Hello World!”.

REFERÊNCIAS:

Documentação do flutter -

https://flutter.dev/docs

IDEs para o flutter -

https://www.dunebook.com/best-ide-for-flutter-development/

Guia de flutter -

https://www.devmedia.com.br/guia/flutter/40713

Hello world para flutter -

https://flutter.dev/docs/get-started/codelab

Guia de instalação do plugin -

https://flutter.dev/docs/development/tools/vs-code

INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE SÃO PAULO CAMPUS CUBATÃO

FLUTTER - PRIMEIRA APLICAÇÃO MOBILE

ALICIA OLIVEIRA DE AGUIAR
ALVARO PRIMITZ
ALLYSON WESLLEY
CAIQUE PEREIRA
DANTON MELLO
GUILHERME FARIAS
LUCAS GOMES
PEDRO DE FRANÇA
RICARDO FERNANDEZ
TIAGO AMARO
VITOR VIEIRA

Cubatão,
Novembro de 2020

DESENVOLVIMENTO DE APLICAÇÕES MOBILE NO FLUTTER

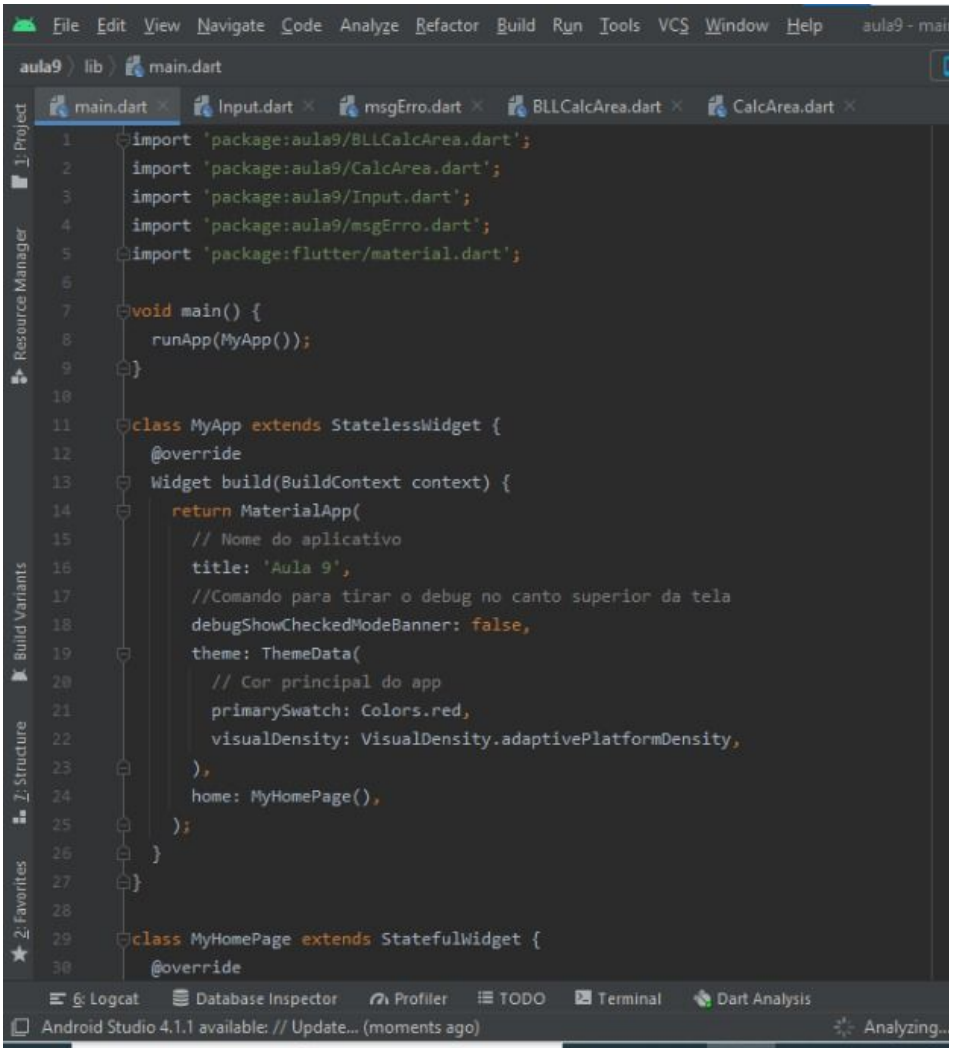
ÁREA DO TRIÂNGULO

Para o desenvolvimento da primeira aplicação mobile em Flutter, inicialmente foi necessário a instalação do emulador Android Studio para rodar o aplicativo num dispositivo Android.

Logo após, foi possível começar a desenvolver o aplicativo para calcular a área do triângulo cuja base e altura devem ser inseridas pelo usuário.

1. Layout do aplicativo

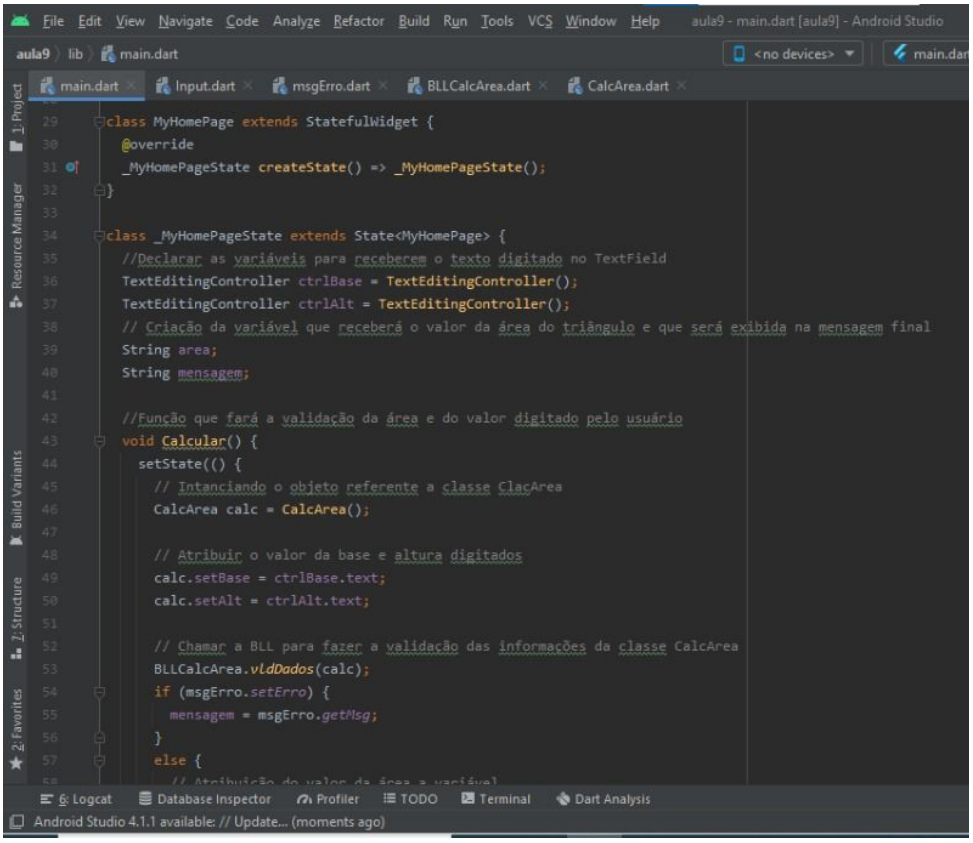
Na classe MyAPP é exibido código referente ao layout do aplicativo, com o título do projeto, as cores principais e os códigos do projeto. Segue abaixo o printscreen com o código do layout:

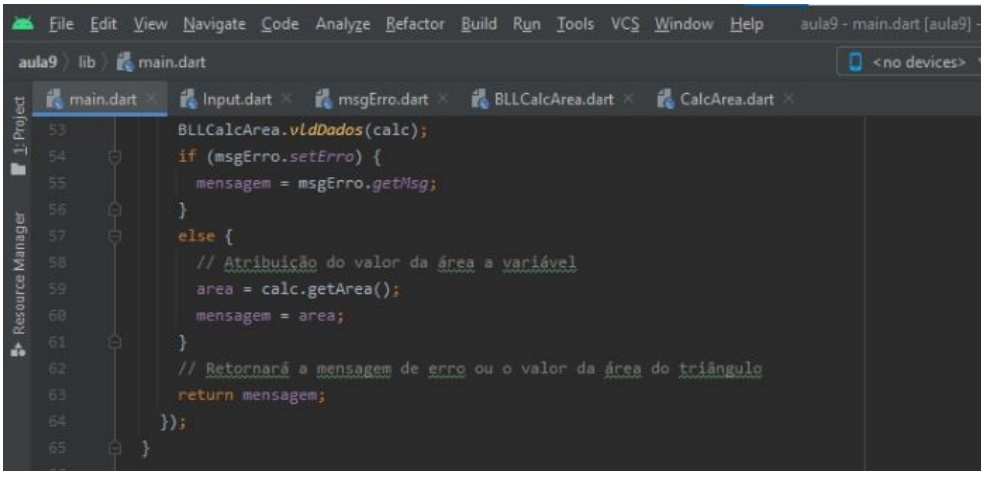


Printscreen 1: CÓDIGO DO LAYOUT DO PROJETO

2. Código principal do aplicativo

Nos códigos abaixo, pertencentes à classe MyHomePageState, é registrado a declaração de variáveis, pelo comando TextEditingController, e a criação das variáveis que receberão os valores digitados pelo usuário a base e a altura do triângulo. Além disso, é nessa classe que ocorre a convocação da classe BLL para a validação dos valores, bem como a exibição dos resultados.





Printscreen 2 e 3: Classe MyPageState

3. Código do Front-End do Aplicativo

Nas linhas abaixo é mostrado o bloco responsável por imprimir na tela as classes e os widgets da tela, como a caixa de digitação do texto, o botão para executar a operação matemática, e a execução da mensagem do programa, tanto de erro (caso os valores sejam inválidos) ou resultado obtido. Ademais, o design do aplicativo, exemplo: cor da fonte, tamanho, etc., é definido através dos códigos:

title: Código que exibe o título na barra superior do aplicativo;

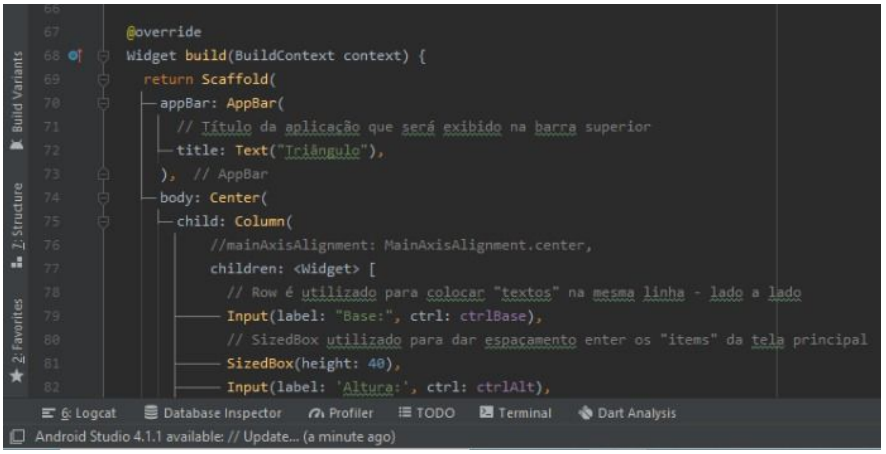
Widget: comando que exibe o conteúdo na tela;

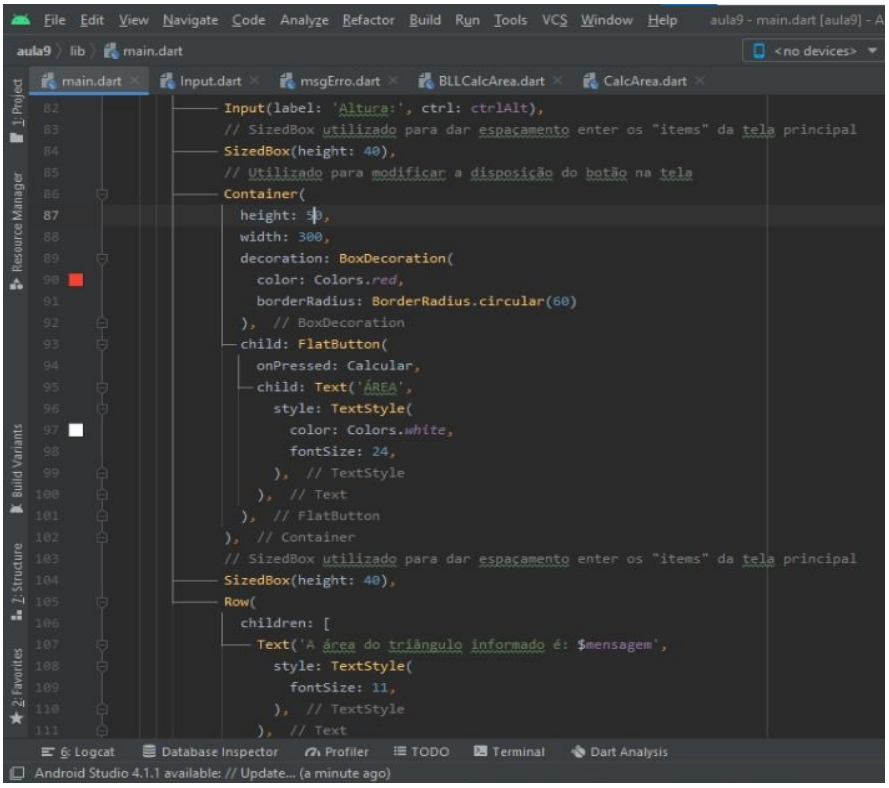
Row: código referente ao alinhamento de textos, lado a lado.

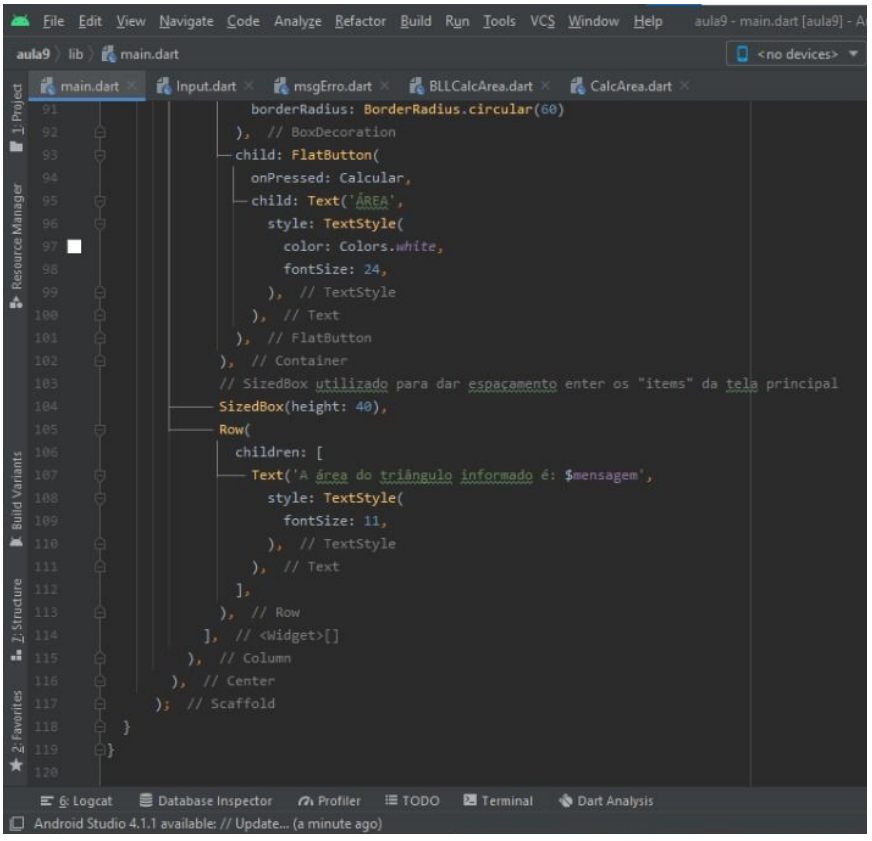
SizeBox: espaçamento entre os itens da tela principal;

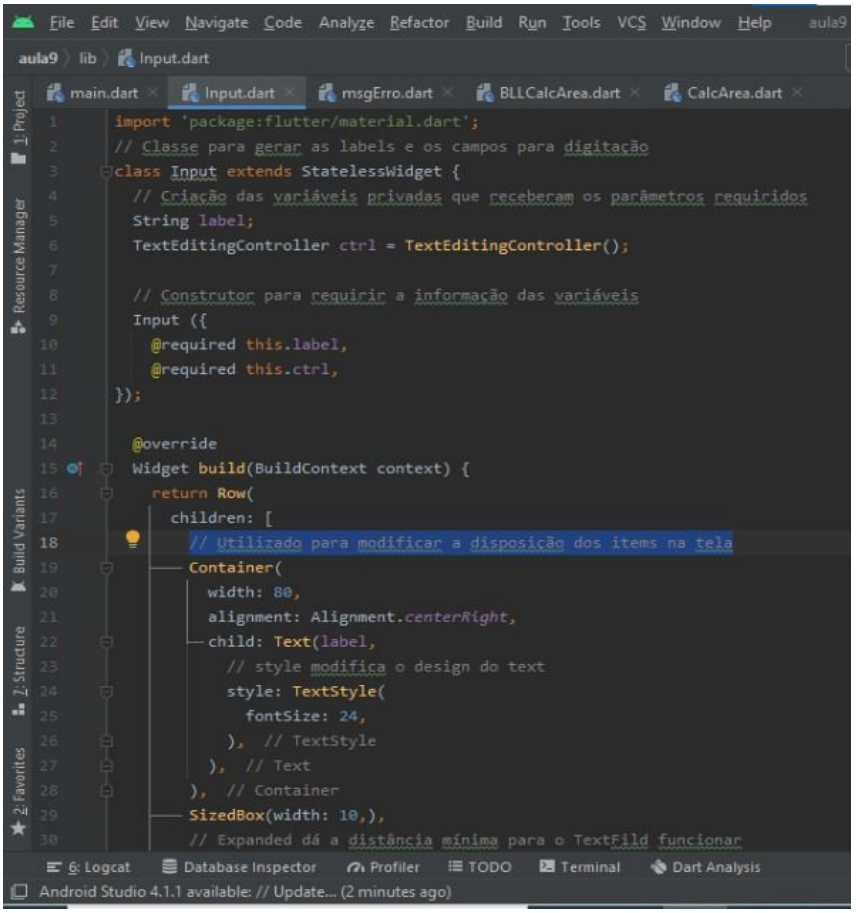
Input label: código que exibe a caixa para digitação dos valores;

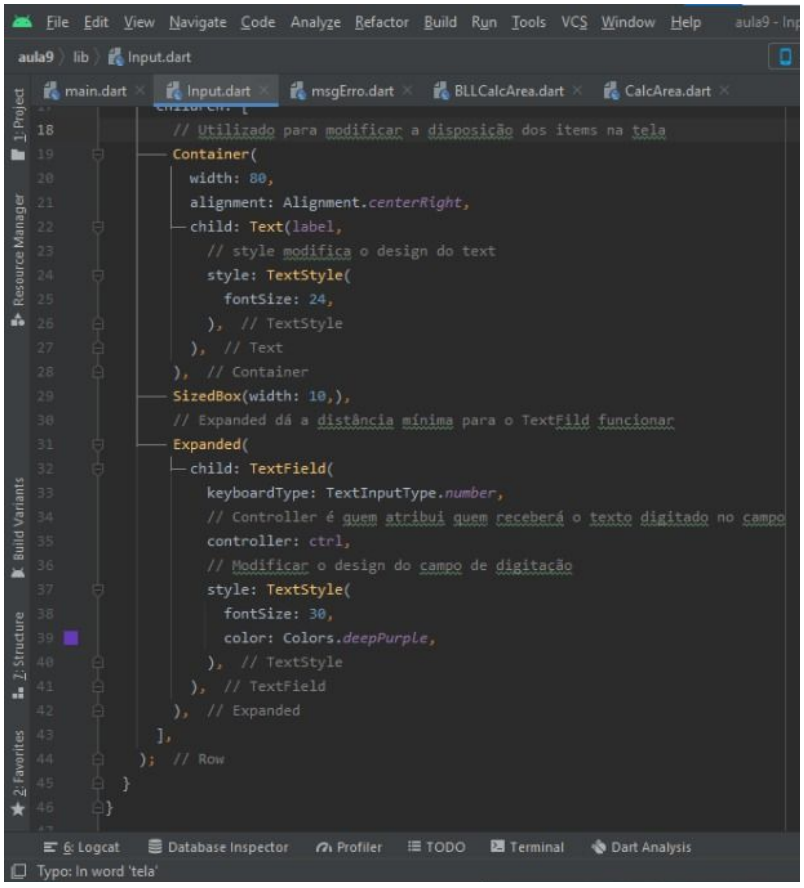
Style: modificação do design do programa.




File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
aula9 - main.dart [aula9] - A
aula9 lib main.dart
<no devices>
main.dart Input.dart msgErro.dart BLLCalcArea.dart CalcArea.dart
Input(label: 'Altura:', ctrl: ctrlAlt),
// SizedBox utilizado para dar espaçamento enter os "items" da tela principal
SizedBox(height: 40),
// Utilizado para modificar a disposição do botão na tela
Container(
height: 50,
width: 300,
decoration: BoxDecoration(
color: Colors.red,
borderRadius: BorderRadius.circular(60)
), // BoxDecoration
child: FlatButton(
onPressed: Calcular,
child: Text('ÁREA',
style: TextStyle(
color: Colors.white,
fontSize: 24,
), // TextStyle
), // Text
), // FlatButton
), // Container
// SizedBox utilizado para dar espaçamento enter os "items" da tela principal
SizedBox(height: 40),
Row(
children: [
Text('A área do triângulo informado é: $mensagem',
style: TextStyle(
fontSize: 11,
), // TextStyle
), // Text
6: Logcat Database Inspector Profiler TODO Terminal Dart Analysis
Android Studio 4.1.1 available: // Update... (a minute ago)


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
aula9 - main.dart [aula9] - A
aula9 lib main.dart
<no devices>
main.dart Input.dart msgErro.dart BLLCalcArea.dart CalcArea.dart
borderRadius: BorderRadius.circular(60)
), // BoxDecoration
child: FlatButton(
onPressed: Calcular,
child: Text('ÁREA',
style: TextStyle(
color: Colors.white,
fontSize: 24,
), // TextStyle
), // Text
), // FlatButton
), // Container
// SizedBox utilizado para dar espaçamento enter os "items" da tela principal
SizedBox(height: 40),
Row(
children: [
Text('A área do triângulo informado é: $mensagem',
style: TextStyle(
fontSize: 11,
), // TextStyle
), // Text
],
), // Row
], // <Widget>[]
), // Column
), // Center
); // Scaffold
}
}
6: Logcat Database Inspector Profiler TODO Terminal Dart Analysis
Android Studio 4.1.1 available: // Update... (a minute ago)

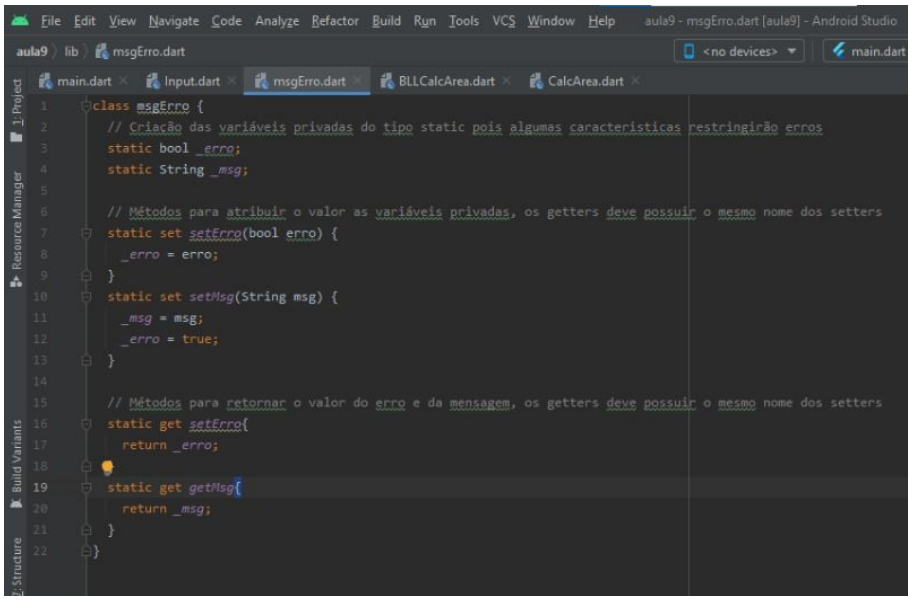




Printscreen 4, 5, 6, 7 e 8: Widgets e design do programa.

4. Classe Erro

A classe erro é a responsável pelo retorno da mensagem de erro, caso os valores sejam inválidos.

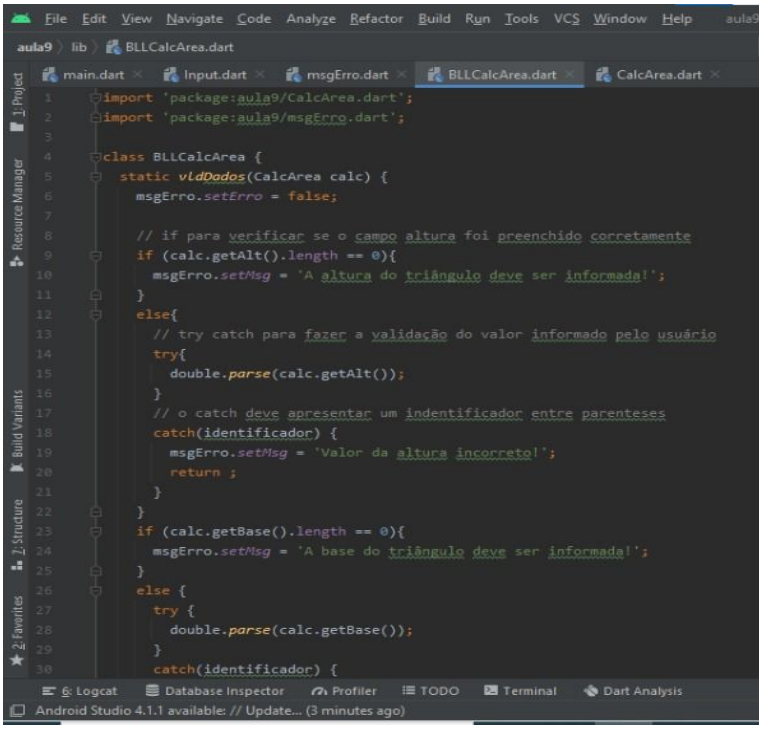


```dart
class msgErro {
  // Criação das variáveis privadas do tipo static pois algumas caracteristicas restringirão erros
  static bool _erro;
  static String _msg;

  // Métodos para atribuir o valor as variáveis privadas, os getters deve possuir o mesmo nome dos setters
  static set setErro(bool erro) {
    _erro = erro;
  }
  static set setMsg(String msg) {
    _msg = msg;
    _erro = true;
  }

  // Métodos para retornar o valor do erro e da mensagem, os getters deve possuir o mesmo nome dos setters
  static get setErro{
    return _erro;
  }
  static get getMsg{
    return _msg;
  }
}
```

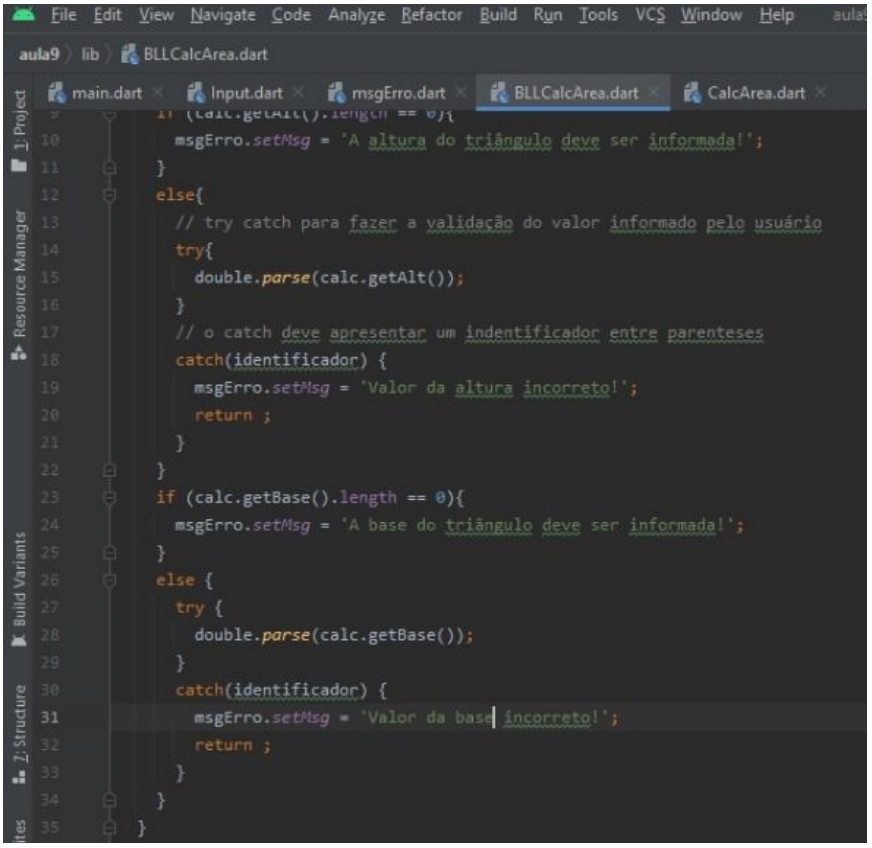
Printscreen 9: Classe Erro.

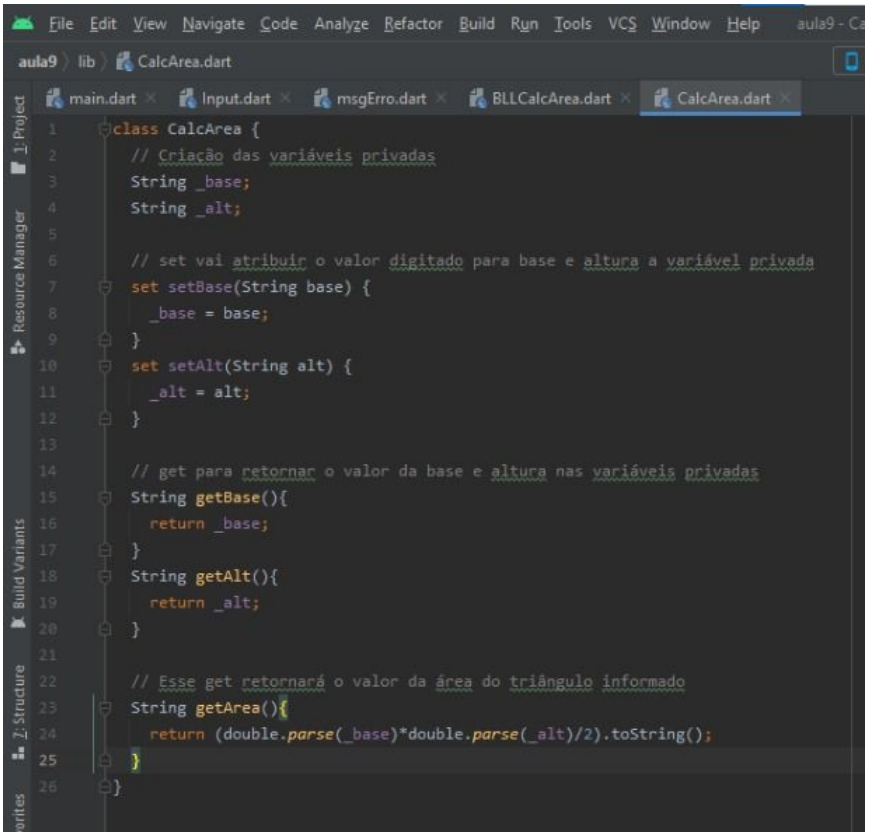
5. Classe BLLCalcArea

Essa classe recebe os valores digitados pelos usuários e realiza a verificação dos valores digitados pelo usuário. Caso os valores sejam inválidos, é exibido uma mensagem de erro ao usuário. Caso contrário, valores válidos, é feito o cálculo da área e exibido os resultados na tela.



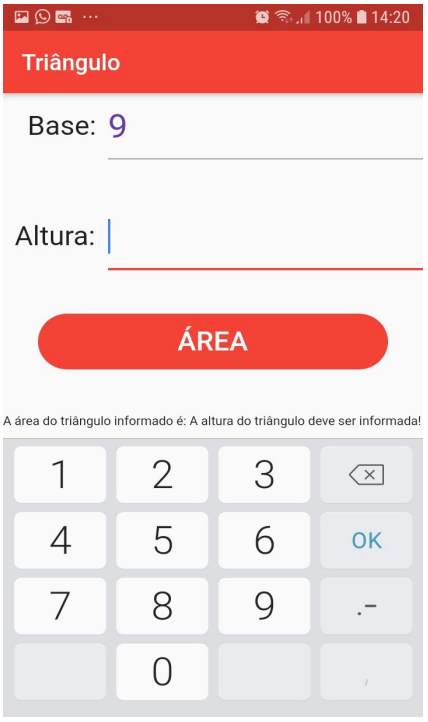
```dart
import 'package:aula9/CalcArea.dart';
import 'package:aula9/msgErro.dart';

class BLLCalcArea {
  static vldDados(CalcArea calc) {
    msgErro.setErro = false;

    // if para verificar se o campo altura foi preenchido corretamente
    if (calc.getAlt().length == 0){
      msgErro.setMsg = 'A altura do triângulo deve ser informada!';
    }
    else{
      // try catch para fazer a validação do valor informado pelo usuário
      try{
        double.parse(calc.getAlt());
      }
      // o catch deve apresentar um indentificador entre parenteses
      catch(identificador) {
        msgErro.setMsg = 'Valor da altura incorreto!';
        return ;
      }
    }
    if (calc.getBase().length == 0){
      msgErro.setMsg = 'A base do triângulo deve ser informada!';
    }
    else {
      try {
        double.parse(calc.getBase());
      }
      catch(identificador) {
```


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
aula9 lib BLLCalcArea.dart
main.dart Input.dart msgErro.dart BLLCalcArea.dart CalcArea.dart
if (calc.getAlt().length == 0){
msgErro.setMsg = 'A altura do triângulo deve ser informada!';
}
else{
// try catch para fazer a validação do valor informado pelo usuário
try{
double.parse(calc.getAlt());
}
// o catch deve apresentar um indentificador entre parenteses
catch(identificador) {
msgErro.setMsg = 'Valor da altura incorreto!';
return ;
}
}
if (calc.getBase().length == 0){
msgErro.setMsg = 'A base do triângulo deve ser informada!';
}
else {
try {
double.parse(calc.getBase());
}
catch(identificador) {
msgErro.setMsg = 'Valor da base incorreto!';
return ;
}
}
}


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
aula9 lib CalcArea.dart
main.dart Input.dart msgErro.dart BLLCalcArea.dart CalcArea.dart
class CalcArea {
// Criação das variáveis privadas
String _base;
String _alt;
// set vai atribuir o valor digitado para base e altura a variável privada
set setBase(String base) {
_base = base;
}
set setAlt(String alt) {
_alt = alt;
}
// get para retornar o valor da base e altura nas variáveis privadas
String getBase(){
return _base;
}
String getAlt(){
return _alt;
}
// Esse get retornará o valor da área do triângulo informado
String getArea(){
return (double.parse(_base)*double.parse(_alt)/2).toString();
}
}

6. Emulador

Após a realização de todos esses procedimentos, a aplicação mostrada pelo emulador, criado através do Android Studio, será:



INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CAMPUS CUBATÃO

FLUTTER - SEGUNDA APLICAÇÃO MOBILE

ALICIA OLIVEIRA DE AGUIAR
ALVARO PRIMITZ
ALLYSON WESLLEY
CAIQUE PEREIRA
DANTON MELLO
GUILHERME FARIAS
LUCAS GOMES
PEDRO DE FRANÇA
RICARDO FERNANDEZ
TIAGO AMARO
VITOR VIEIRA

Cubatão,
Novembro de 2020

DESENVOLVIMENTO DE APLICAÇÕES MOBILE NO FLUTTER

Na nossa segunda aplicação mobile em Flutter, foi solicitado o desenvolvimento de um aplicativo com um botão que, quando pressionado, acesse o GPS do celular, localize as coordenadas do dispositivo e as exiba na tela, através de ponto plotado no Google Maps. Para isso, foi necessário a instalação do emulador Android Studio para rodar o aplicativo num dispositivo android e acessar os recursos de hardware nativos do celular, como o GPS ou o Google Maps.

Diante disso, seguimos as seguintes etapas para o desenvolvimento da aplicação:

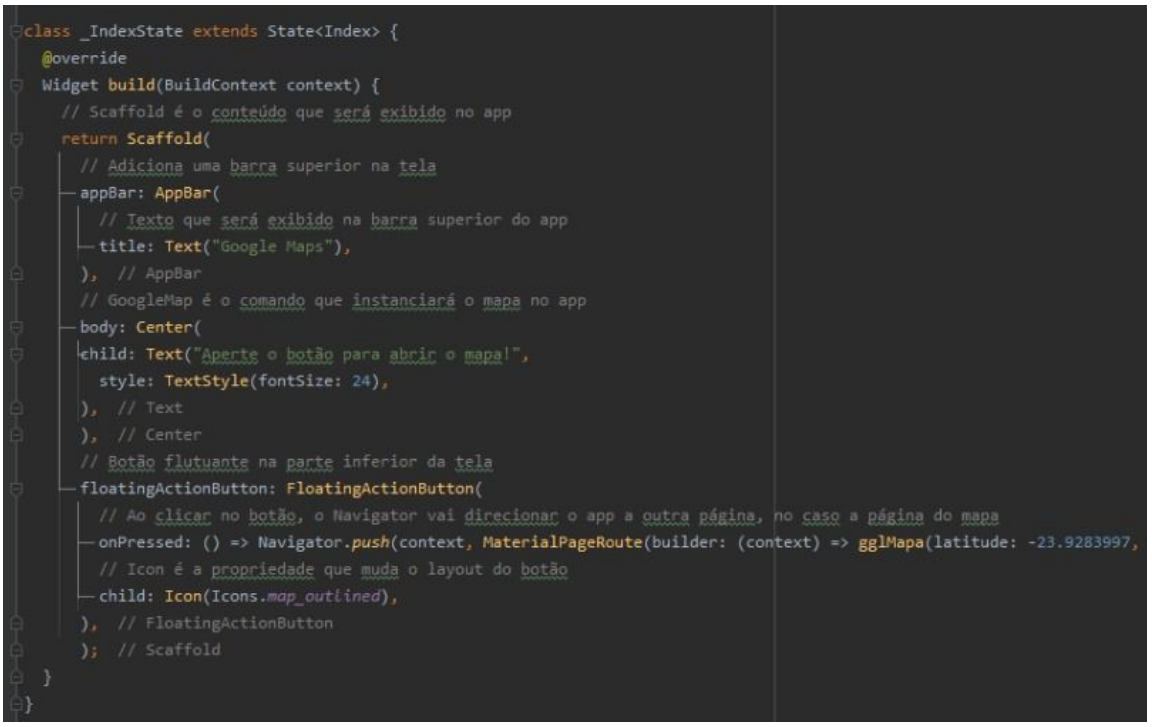
1. CÓDIGO INICIAL

Na classe MyAPP é exibido o código referente ao layout do aplicativo, com o título do projeto, as cores principais e os demais códigos. Abaixo é possível observar o printscreen da classe:

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      // Nome do app
      title: 'Aula 14',
      // Retirar o debug no canto superior esquerdo
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.green,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),  // ThemeData
      home: Index(),
    );  // MaterialApp
  }
}
```

2. LAYOUT DO APLICATIVO

Nos códigos abaixo, pertencentes à classe_IndexState, é responsável pela exibição do layout da página inicial, a criação de uma barra superior, inserir um texto e outros aspectos estéticos. Nessa classe também é introduzido um método que, ao clicar no botão responsável por abrir o mapa, direciona o mesmo para uma coordenada determinada.



```dart
class _IndexState extends State<Index> {
  @override
  Widget build(BuildContext context) {
    // Scaffold é o conteúdo que será exibido no app
    return Scaffold(
      // Adiciona uma barra superior na tela
      appBar: AppBar(
        // Texto que será exibido na barra superior do app
        title: Text("Google Maps"),
      ),  // AppBar
      // GoogleMap é o comando que instanciará o mapa no app
      body: Center(
      child: Text("Aperte o botão para abrir o mapa!",
        style: TextStyle(fontSize: 24),
      ),  // Text
      ),  // Center
      // Botão flutuante na parte inferior da tela
      floatingActionButton: FloatingActionButton(
        // Ao clicar no botão, o Navigator vai direcionar o app a outra página, no caso a página do mapa
        onPressed: () => Navigator.push(context, MaterialPageRoute(builder: (context) => gglMapa(latitude: -23.9283997,
        // Icon é a propriedade que muda o layout do botão
        child: Icon(Icons.map_outlined),
      ),  // FloatingActionButton
      );  // Scaffold
  }
}
```

3. ARMAZENAMENTO DOS VALORES DAS COORDENADAS

A classe gglMapa é responsável por receber e armazenar os valores das coordenadas. Através do controlador do mapa, será localizada a posição geográfica e, posteriormente, registrado os valores da longitude e da latitude, do ponto plotado no mapa, nas variáveis longitude e latitude, respectivamente.

Essa classe também será responsável realizar a ligação entre o Google Maps com o controlador e por adicionar um marcador no mapa com a identificação da sua localização, e os seus aspectos estéticos.

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

class gglMapa extends StatefulWidget {
  // Variáveis globais para receber os valores das coordenadas
  double latitude;
  double longitude;

  // Construtor que pedirá que informe a latitude e longitude ao chamar essa classe
  gglMapa({
    Key key, @required this.latitude,
    @required this.longitude,
  }): super(key: key);

  @override
  _gglMapaState createState() => _gglMapaState(latitude,longitude);
  }

class _gglMapaState extends State<gglMapa> {
  // controlador do mapa, para controlar onde o mapa estará apontando
  GoogleMapController ctrlMapa;

  //variáveis que receberão as ordenadas no mapa
  double latitude;
  double longitude;
```

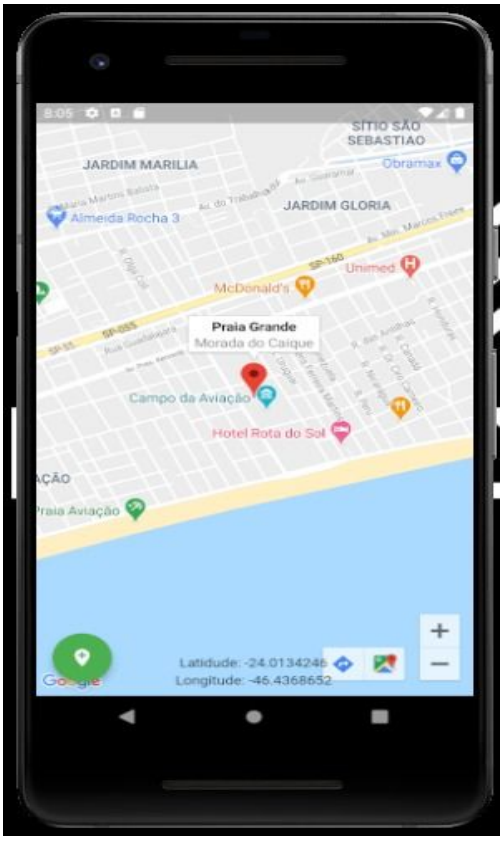
```dart
// Trazer os valores do statefull de volta
_gglMapaState(this.latitude,this.longitude);

// variavel para setar um marcador no mapa (aquele treco vermelho que tem no google maps)
Set<Marker> indicador = new Set<Marker>();

// função para atribuir a ordenada ao controlador, fazendo uma ligação entre o GoogleMap e o controlador
void _Mapinha (GoogleMapController ctrl){
  ctrlMapa = ctrl;

  setState(() {
    // Adicionando o marcador ao mapa
    indicador.add(Marker(
      markerId: MarkerId("0"),
      position: LatLng(-24.0134246,-46.4368652),
      // InfoWindow será onde poderemos apresentar um texto, por exemplo, ao clicar no marcador
      infoWindow: InfoWindow(
        title: "Praia Grande",
        snippet: "Morada do Caique",
      ),  // InfoWindow
    ),  // Marker
    );
```

```dart
// Função para setar uma nova coordenada no mapa
static final CameraPosition meuCEP = CameraPosition(
  target: LatLng(-24.0134246, -46.4368652),
  zoom: 15,
);  // CameraPosition

// Função para mudar a posição do mapa
void _meuCEP() {
  setState(() {
    latitude = -24.0134246;
    longitude = -46.4368652;
  });

  final GoogleMapController controlador = ctrlMapa;
  controlador.animateCamera(CameraUpdate.newCameraPosition(meuCEP));
}

@override
Widget build(BuildContext context) {
  // Scaffold é o conteúdo que será exibido no app
  return Scaffold(
    // Center irá centralizar o conteudo da tela
    body: Stack(
      // Container para redimensionar o tamanho do conteúdo na tela
      children: <Widget> [
        Container(
          width: 440,
          height: 690,
          // GoogleMap é o comando que inicia o mapa no app
          child: GoogleMap(
            // onMapCreated irá chamar os valores da função _Mapinha
            onMapCreated: _Mapinha,
            // Configuração da posição inicial da câmera ao abrir o mapa
            initialCameraPosition: CameraPosition(
              target: LatLng(latitude,longitude),
              // zoom inicial do mapa
              zoom: 15,
            ),  // CameraPosition
            // este comando irá colocar um ponteiro inicial no mapa
            markers: indicador,
          ),  // GoogleMap
        ),  // Container
        Container(
          alignment: Alignment.bottomCenter,
          padding: EdgeInsets.fromLTRB(0, 0, 0, 30),
          child: Text("Latidude: $latitude"),
        ),  // Container
        Container(
          alignment: Alignment.bottomCenter,
          padding: EdgeInsets.fromLTRB(0, 0, 0, 10),
          child: Text("Longitude: $longitude"),
        ),  // Container
      ]  // <Widget>[]
    ),  // Stack
    // Botão flutuante na parte inferior da tela
    floatingActionButton: FloatingActionButton(
      // Quando clicar no botão, irá chamar a função _meuCEP
      onPressed: _meuCEP,
```

```
            markers: indicador,
          ),  // GoogleMap
        ),  // Container
        Container(
          alignment: Alignment.bottomCenter,
          padding: EdgeInsets.fromLTRB(0, 0, 0, 30),
          child: Text("Latidude: $latitude"),
        ),  // Container
        Container(
          alignment: Alignment.bottomCenter,
          padding: EdgeInsets.fromLTRB(0, 0, 0, 10),
          child: Text("Longitude: $longitude"),
        ),  // Container
      ]  // <Widget>[]
    ),  // Stack
    // Botão flutuante na parte inferior da tela
    floatingActionButton: FloatingActionButton(
      // Quando clicar no botão, irá chamar a função _meuCEP
      onPressed: _meuCEP,
      // Icon é a propriedade que muda o layout do botão
      child: Icon(Icons.add_location),
    ),  // FloatingActionButton
    // Location predefine em qual parte da tela o botão irá aparecer
    floatingActionButtonLocation: FloatingActionButtonLocation.startFloat,
  );  // Scaffold
 }
}
```

4. TELA DO APLICATIVO

Por fim, após todas as etapas de programação detalhadas acima, o aplicativo ficou com o seguinte design:



INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CAMPUS CUBATÃO

FLUTTER - TERCEIRA APLICAÇÃO MOBILE

ALICIA OLIVEIRA DE AGUIAR
ALVARO PRIMITZ
ALLYSON WESLLEY
CAIQUE PEREIRA
DANTON MELLO
GUILHERME FARIAS
LUCAS GOMES
PEDRO DE FRANÇA
RICARDO FERNANDEZ
TIAGO AMARO
VITOR VIEIRA

Cubatão,
Novembro de 2020

DESENVOLVIMENTO DE APLICAÇÕES MOBILE NO FLUTTER

Foi solicitado pelo docente o desenvolvimento de aplicação mobile em Flutter (App03) que localize o contato de uma pessoa na lista de contatos do celular, através da digitação e correspondência do nome com o seu respectivo número de telefone, e, após encontrado, o programa deverá exibir na tela os seus dados. Através disso, objetivamos conhecer os recursos de software nativos do celular, nativos do sistema Android, no caso a lista de contatos.

Diante disso, seguimos as seguintes etapas para o desenvolvimento da aplicação:

1. CRIAÇÃO DO APLICATIVO

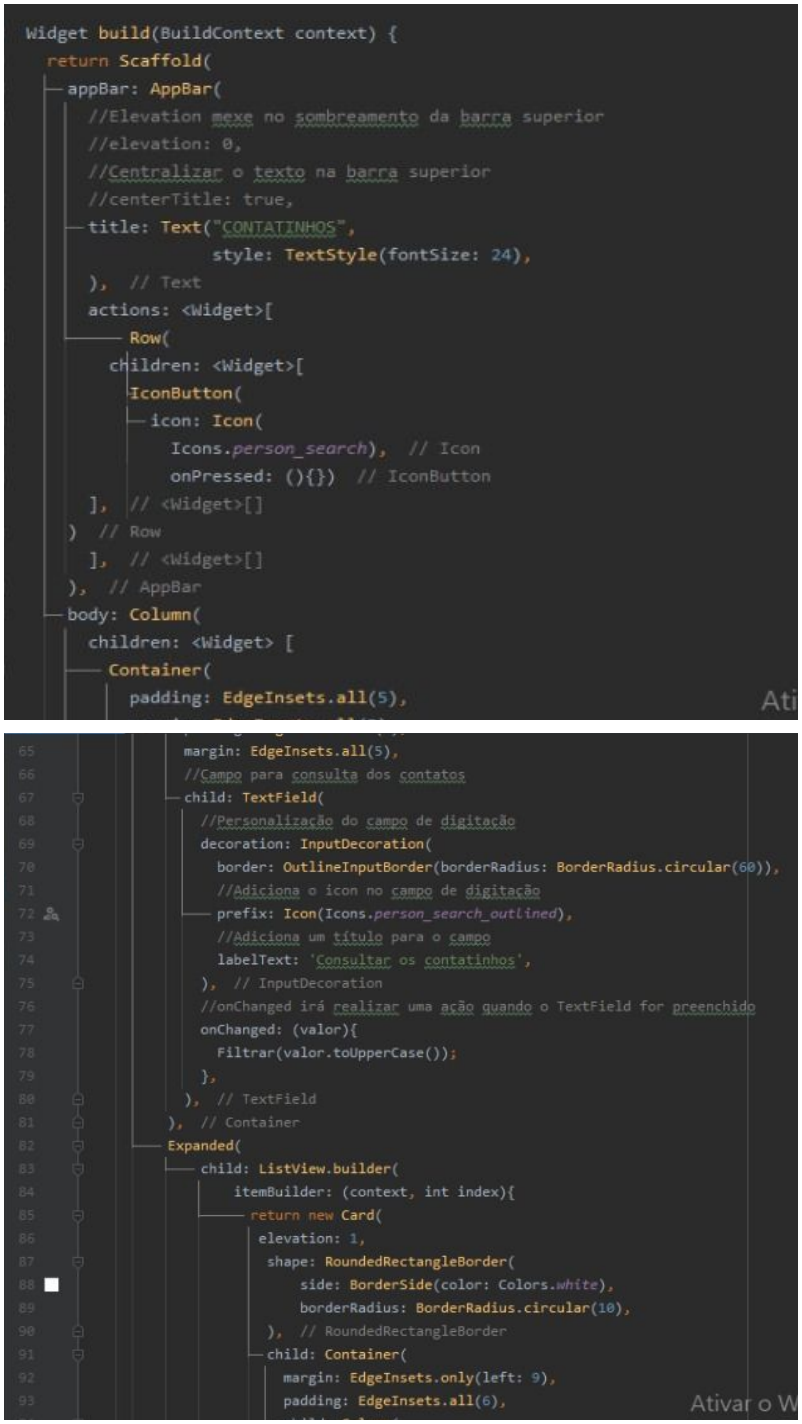
Para iniciar o desenvolvimento da aplicação, foi criado o programa “Lista de Contatos IFSP” e definido as classes (Index, MyApp e _IndexState), instanciado os objetos (ThemeData e MaterialApp) e declarado a variável Contatos, que guardará os dados telefônicos da pessoa.

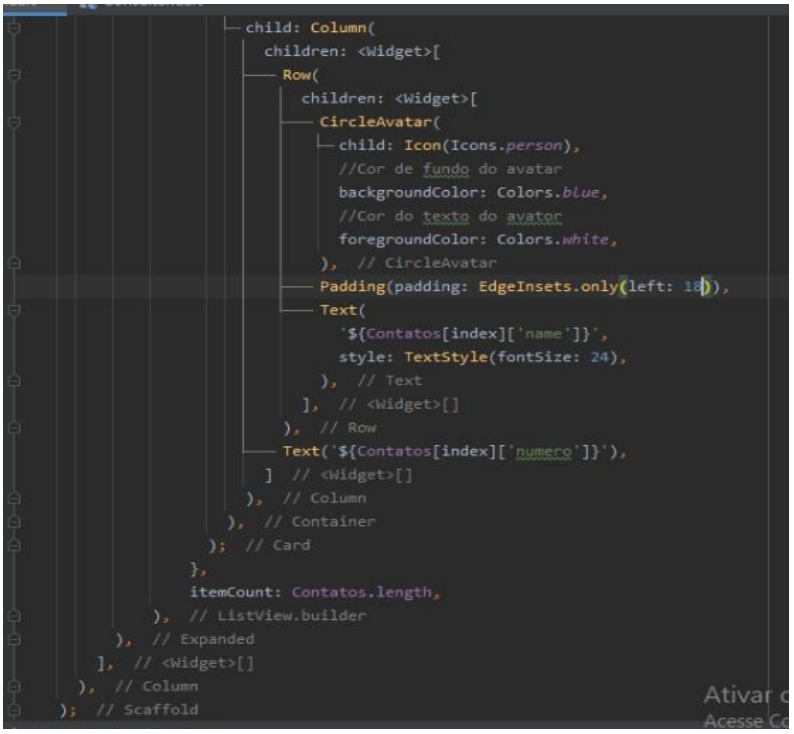
```
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Lista de Contatos IFSP',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),  // ThemeData
      home: Index(),
    );  // MaterialApp
  }
}

class Index extends StatefulWidget {
  @override
  _IndexState createState() => _IndexState();
}

class _IndexState extends State<Index> {
  //Criar a variável que receberá os contatos
  var Contatos = [];
```

```
    //Adicionar os contatos à lista de contatos
    Contatos.addAll(Consultar);
  }
```

2. CÓDIGO INICIAL

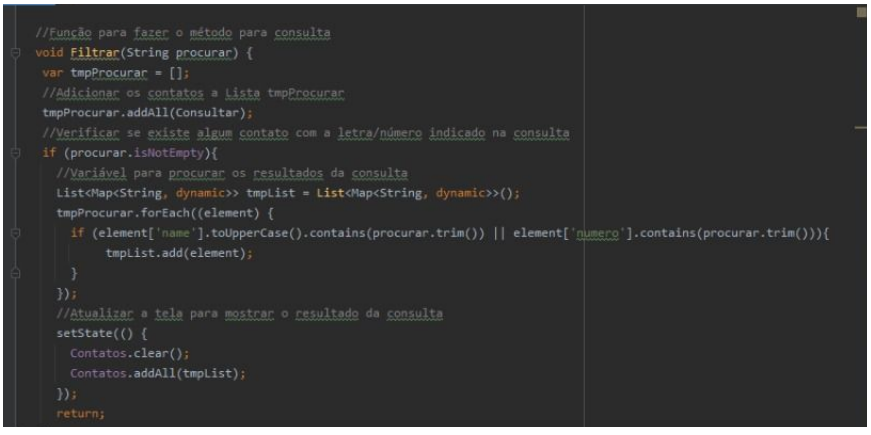
Na classe MyAPP é exibido o código referente ao layout do aplicativo, com o título do projeto, as cores principais e os demais códigos. Nessa classe, foi estabelecido o aplicativo “CONTATINHOS” e definido a posição dos widgets, com a localização da barra de pesquisa, a cor dos ícones e botões, e também, introduzido o método que, ao clicar no botão, pesquise e localize o contato do nome digitado. Abaixo o print screen é apresentado todos os comandos da classe:



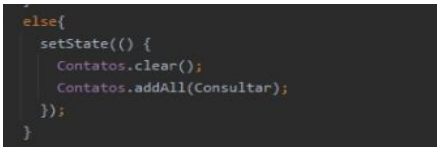


3. CONSULTA E LOCALIZAÇÃO DOS CONTATOS

O método Void, primeiramente, irá retornar a String Filtrar para consultar se o nome digitado no campo digitação, TextField, encontra-se na lista de contatos. Caso encontrado, o contato será adicionado à variável “tmpProcurar” e averiguado a existência de algum outro tipo de informação similar (letra, número ou ícone) contido na lista de números, e, posteriormente, exibido os resultados encontrados na tela.

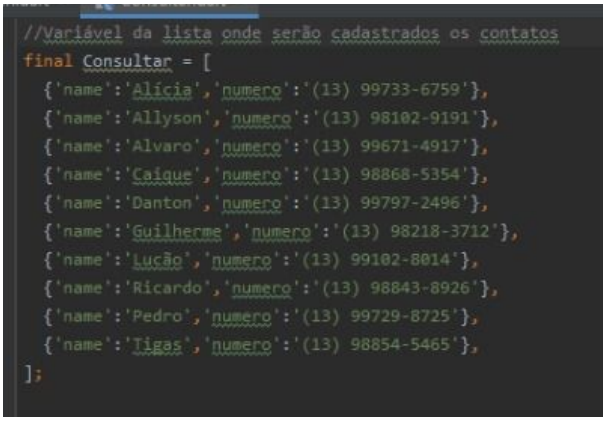


Observação: caso o contato não seja encontrado, o programa irá limpar a tela do aplicativo.



4. GRAVAÇÃO DOS DADOS

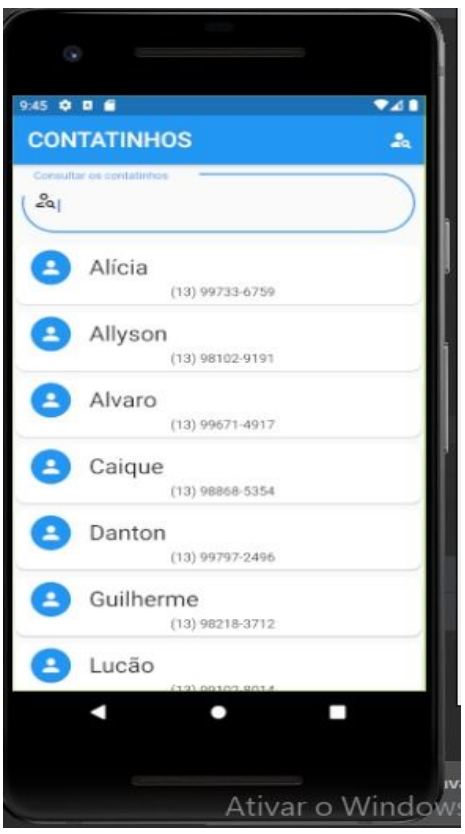
Após a pesquisa do contato dos indivíduos, o programa gravará os dados relativos a eles, como nome e número de telefone, na variável “Consultar”, permitindo a listagem e o encontro das informações.

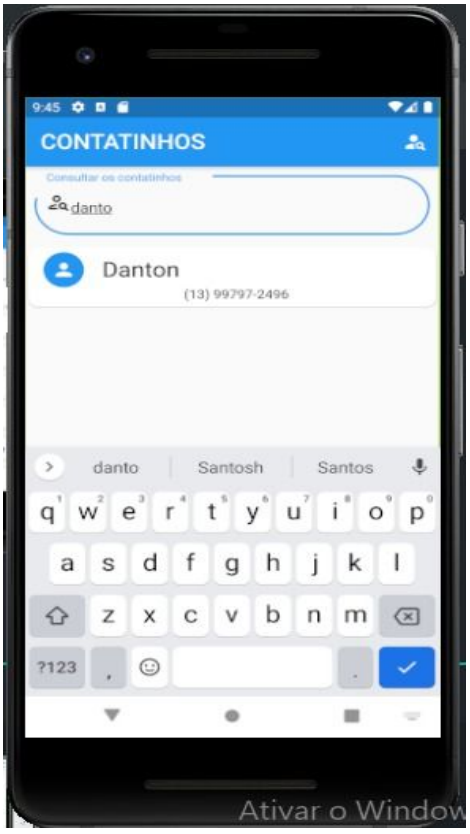


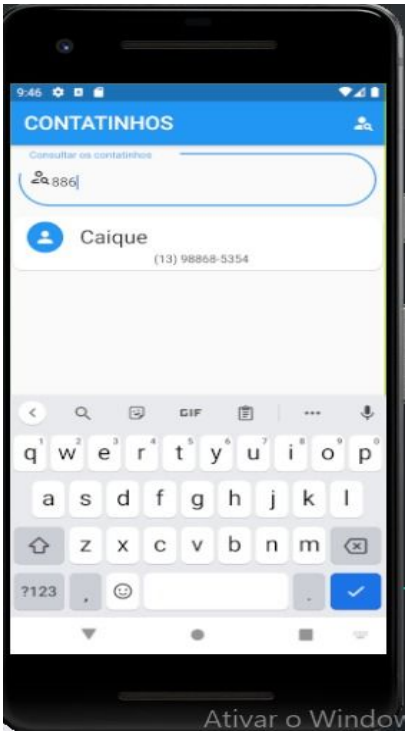
```
//Variável da lista onde serão cadastrados os contatos
final Consultar = [
  {'name':'Alicia','numero':'(13) 99733-6759'},
  {'name':'Allyson','numero':'(13) 98102-9191'},
  {'name':'Alvaro','numero':'(13) 99671-4917'},
  {'name':'Caique','numero':'(13) 98868-5354'},
  {'name':'Danton','numero':'(13) 99797-2496'},
  {'name':'Guilherme','numero':'(13) 98218-3712'},
  {'name':'Lucão','numero':'(13) 99102-8014'},
  {'name':'Ricardo','numero':'(13) 98843-8926'},
  {'name':'Pedro','numero':'(13) 99729-8725'},
  {'name':'Tigas','numero':'(13) 98854-5465'},
];
```

5. TELA DO APLICATIVO:

Após a realização de todos esses procedimentos, a aplicação ficou com o seguinte design, sendo possível realizar a busca dos contatos através dos nomes e também dos números:







INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CAMPUS CUBATÃO

FLUTTER - QUARTA APLICAÇÃO MOBILE

ALICIA OLIVEIRA DE AGUIAR
ALVARO PRIMITZ
ALLYSON WESLLEY
CAIQUE PEREIRA
DANTON MELLO
GUILHERME FARIAS
LUCAS GOMES
PEDRO DE FRANÇA
RICARDO FERNANDEZ
TIAGO AMARO
VITOR VIEIRA

Cubatão,
Janeiro de 2021

DESENVOLVIMENTO DE APLICAÇÕES MOBILE NO FLUTTER

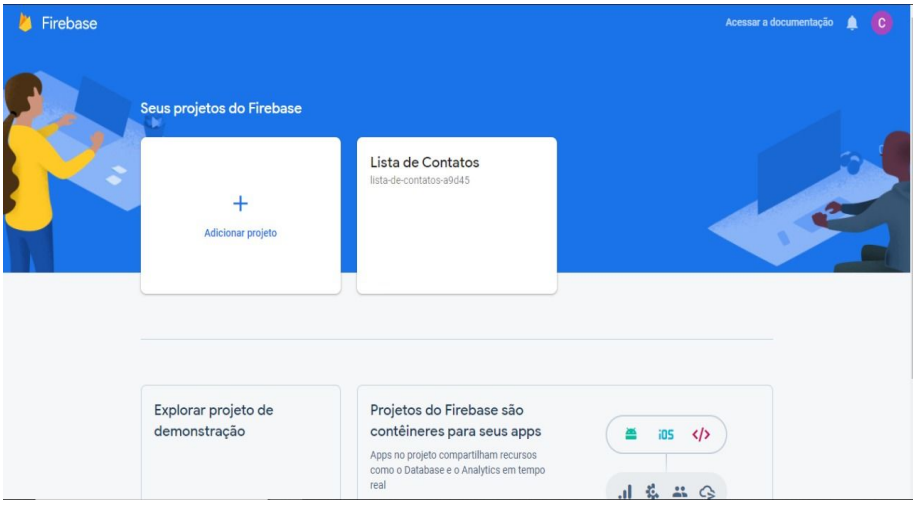
Foi solicitado pelo docente o desenvolvimento de aplicação mobile em Flutter (App03) que deve nos permitir salvar, consultar, deletar e alterar dados em um banco de dados, que também é conhecido como CRUD. Através disso utilizamos como parâmetro uma lista de contatos, com isso objetivamos conhecer os recursos de software nativos do celular, o sistema Android e juntamente com uma base de dados, tornando-se possível realizarmos diversos comandos na listagem..

Diante disso, seguimos as seguintes etapas para o desenvolvimento da aplicação:

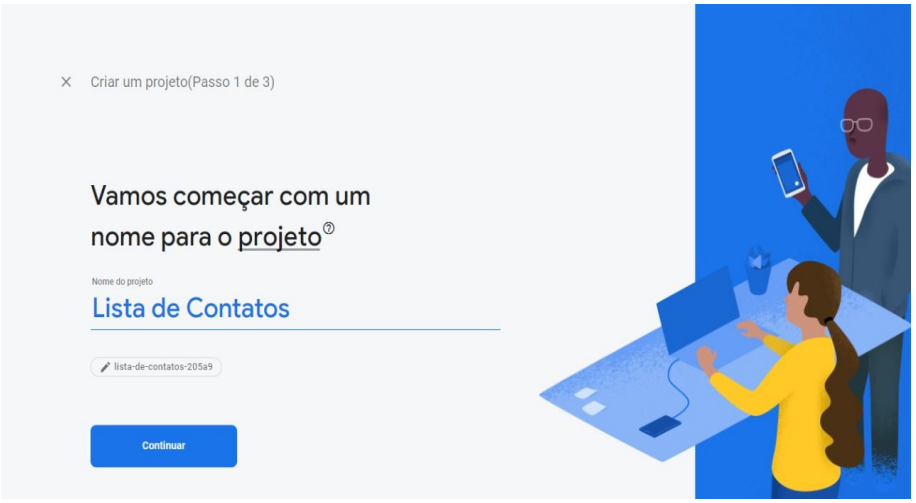
1. CRIAÇÃO DE BANCO DE DADOS NO FIREBASE

Para a criação do banco de dados, foi escolhida a plataforma “Firebase” da Google. Sendo assim, foram realizados alguns passos que permitiram a obtenção de um banco de dados, os quais são

Na tela inicial, pressionou-se o botão “adicionar projeto” para iniciar a criação de um novo banco de dados.

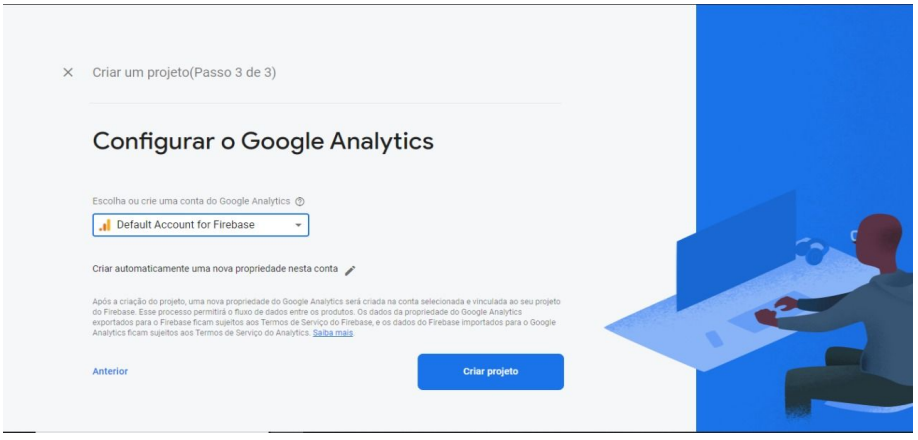


Após isso, inseriu-se o nome do novo projeto, intitulado como “Lista de Contatos”.

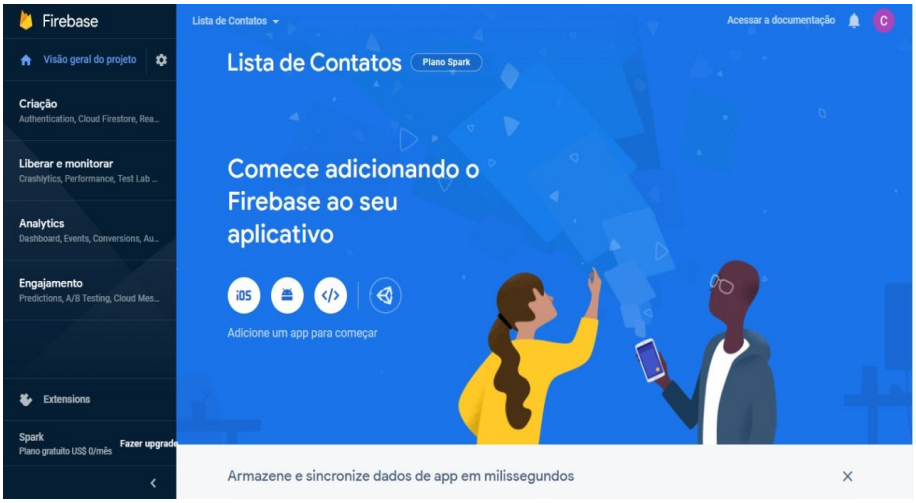


Posteriormente, foi realizada a ativação de um recurso do Google Firebase, que auxiliará no projeto.

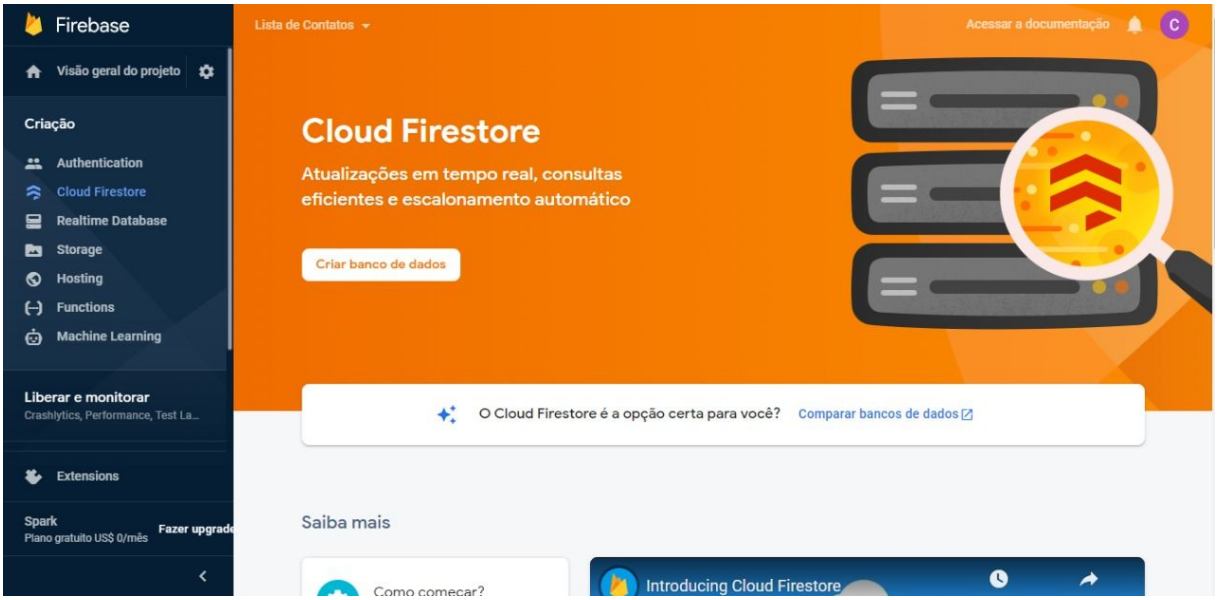
Novamente em relação ao Google Analytics, foi feita a configuração do recurso, selecionando a opção de “Default Account for Firebase”.



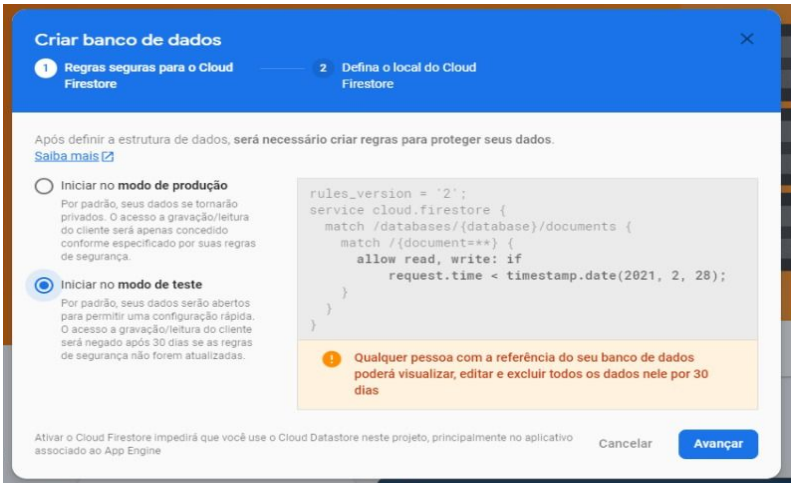
Após isso, na interface principal da plataforma, clicou-se em “Criação” e em seguida em “Cloud Firebase”



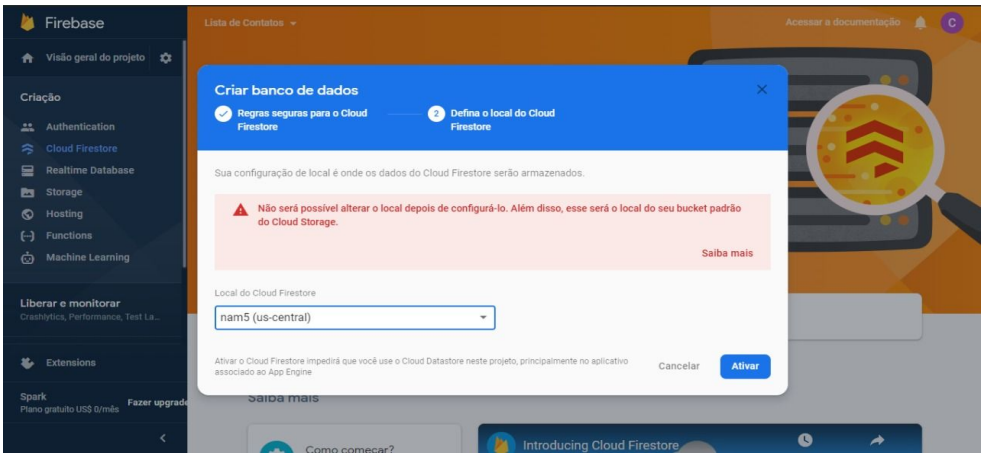
Na próxima janela, pressionou-se o botão “Criar banco de dados”, que como o nome sugere, permitirá a criação do banco de dados.



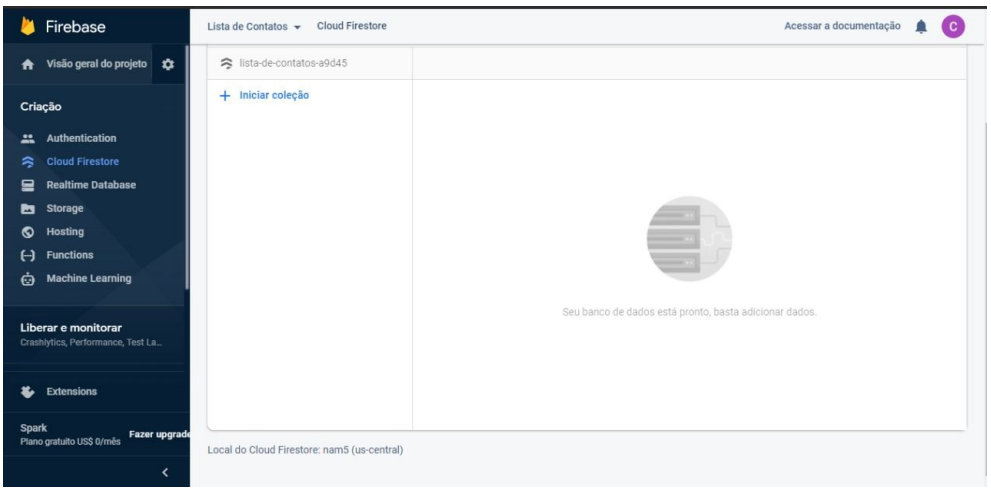
Então, foi aberta uma nova aba em que preferiu-se a opção “Iniciar no modo teste” e em seguida clicou-se no botão “Avançar”.



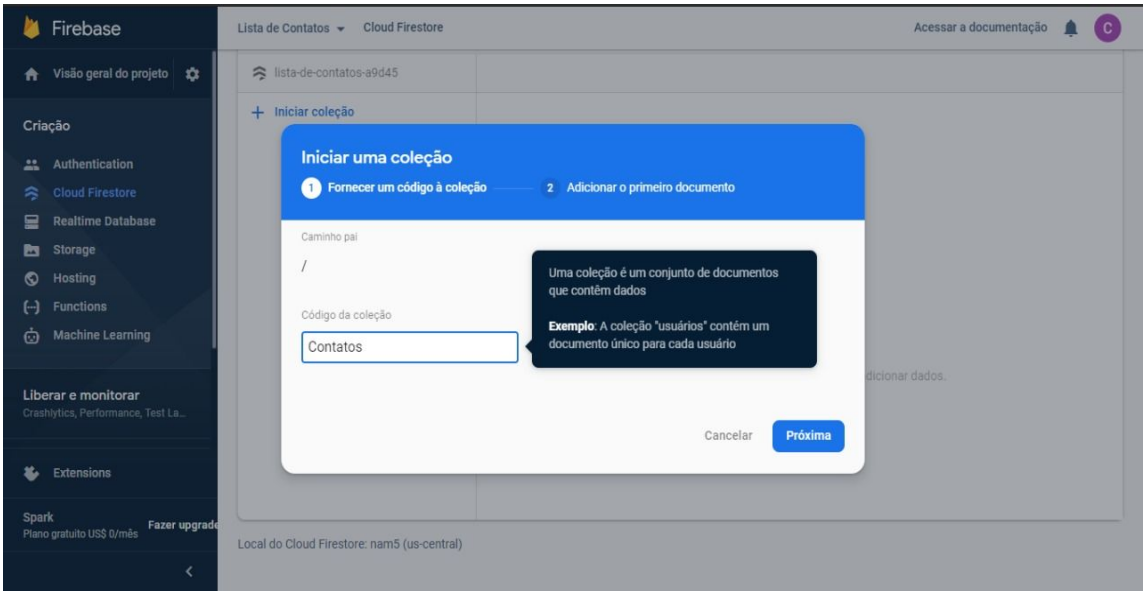
Posteriormente, fora definido o local do Cloud Firestone, escolhendo por utilizar a opção “nam5 (us-central)” e em seguida apertando o botão “Ativar”.



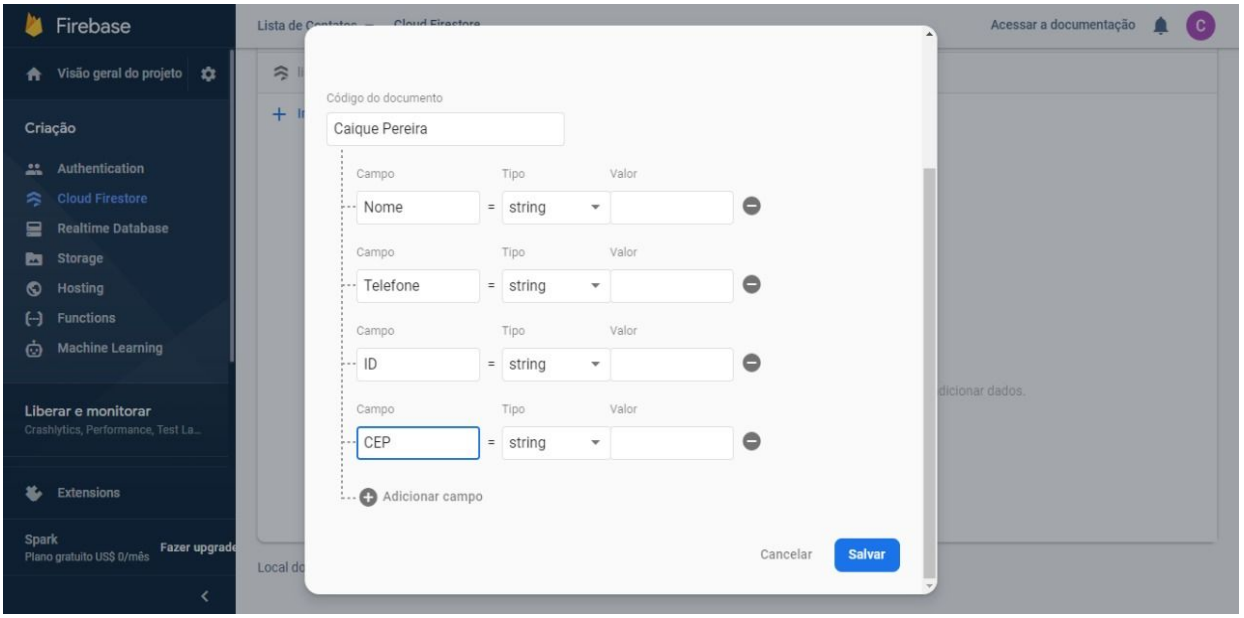
Após feito isso, utilizou-se o botão “Iniciar coleção” para ser criado uma coleção que permitirá a alocação dos dados do APP.



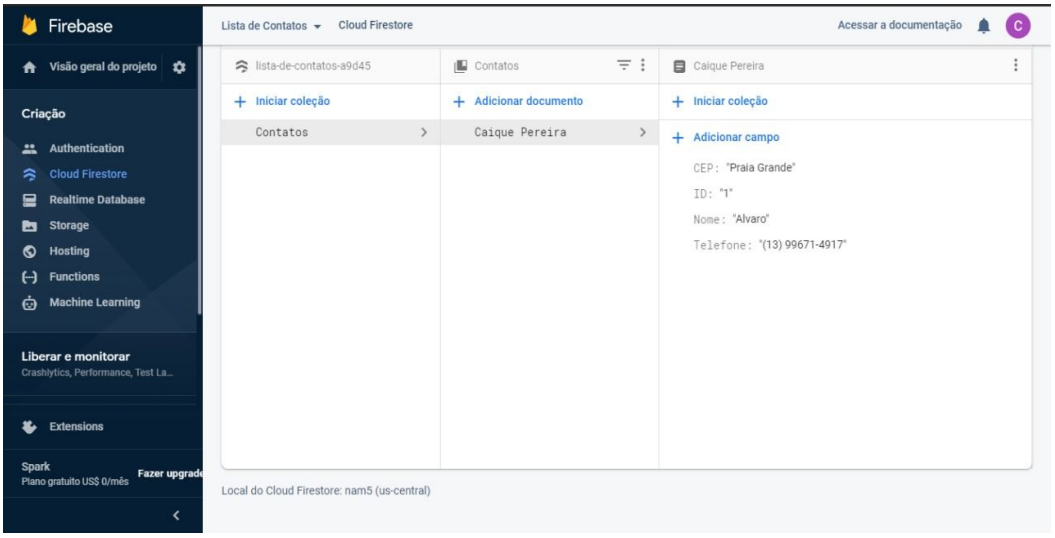
Com a abertura da nova aba, foi inserido o código da coleção, nomeado como “Contatos”. Em seguida, pressionou-se o botão “Próximo”.



Em seguida, foram declarados os campos e suas tipagens que serão utilizados posteriormente.



Logo em seguida, no layout da coleção “Contatos”, foi realizado um teste atribuindo valores a esses campos.



Feito todos os passos, realizou-se a criação do banco de dados através da plataforma Google Firebase, que será utilizada posteriormente para a criação do APP.

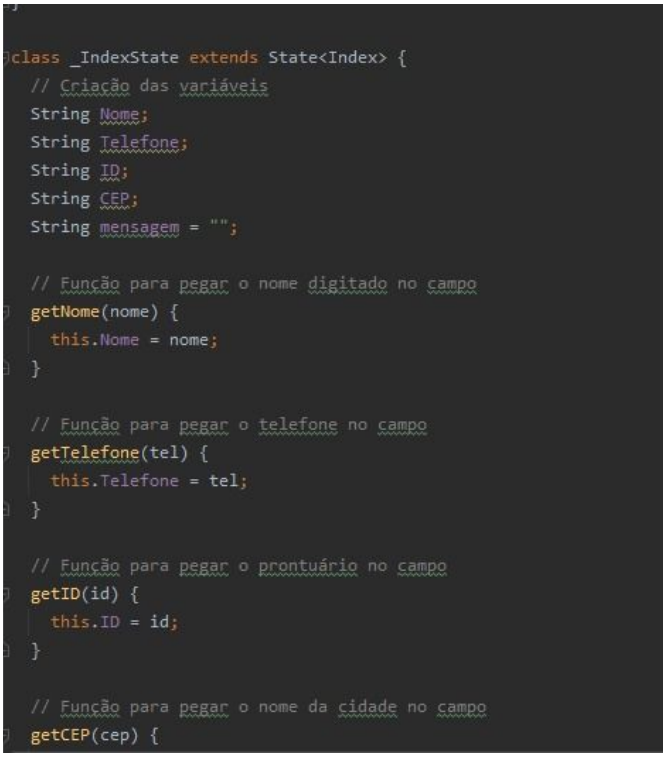
2. CRIAÇÃO DO APLICATIVO:

Para iniciar o desenvolvimento da aplicação, foi necessária a importação da pasta do banco de dados, e também na função *void main()* foram precisos a instanciação de dois métodos “WidgetFlutterBinding” e “Firebase” para que o app consiga se comunicar com o banco de dados “Firestore”. Além disso, foi definida a classe “MyApp” retornando “MaterialApp”.

```dart
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
// Importação da pasta do banco de dados
import 'package:cloud_firestore/cloud_firestore.dart';

void main() {
  // Esses dois métodos WidgetFlutterBinding e Firebase deverão ser instanciados para que o app consiga se comunicar
  // com o banco de dados Firestore
  WidgetsFlutterBinding.ensureInitialized();
  Firebase.initializeApp();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Aula 19',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.purple,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),  // ThemeData
      home: Index(),
    );  // MaterialApp
  }
}

class Index extends StatefulWidget {
  @override
```

3. CÓDIGO INICIAL:

Primeiramente foi gerada uma classe chamada “_IndexState**”,** em que foram criadas as variáveis para caracterização dos integrantes da lista de contatos. Nesta classe, foram feitas funções com objetivo de atribuir valor às variáveis criadas anteriormente. Abaixo o print screen é apresentada as atribuições:



```
class _IndexState extends State<Index> {
  // Criação das variáveis
  String Nome;
  String Telefone;
  String ID;
  String CEP;
  String mensagem = "";

  // Função para pegar o nome digitado no campo
  getNome(nome) {
    this.Nome = nome;
  }

  // Função para pegar o telefone no campo
  getTelefone(tel) {
    this.Telefone = tel;
  }

  // Função para pegar o prontuário no campo
  getID(id) {
    this.ID = id;
  }

  // Função para pegar o nome da cidade no campo
  getCEP(cep) {
```

4. COMANDOS DA LISTA:

Logo após à criação das variáveis, também na mesma classe foram criadas funções como “CadastrarDados()”, “ExcluirDados()”, “ConsultarDados()”, “AtualizarDados()”,

Esses comandos são onde foram registrados um novo contato no banco de dados, exclusão, consultas e alterações, utilizando-se métodos e referências para alocação do valor, além de exibições de mensagens para quando a operação ocorrer com êxito, evidenciando-se nas imagens abaixo:

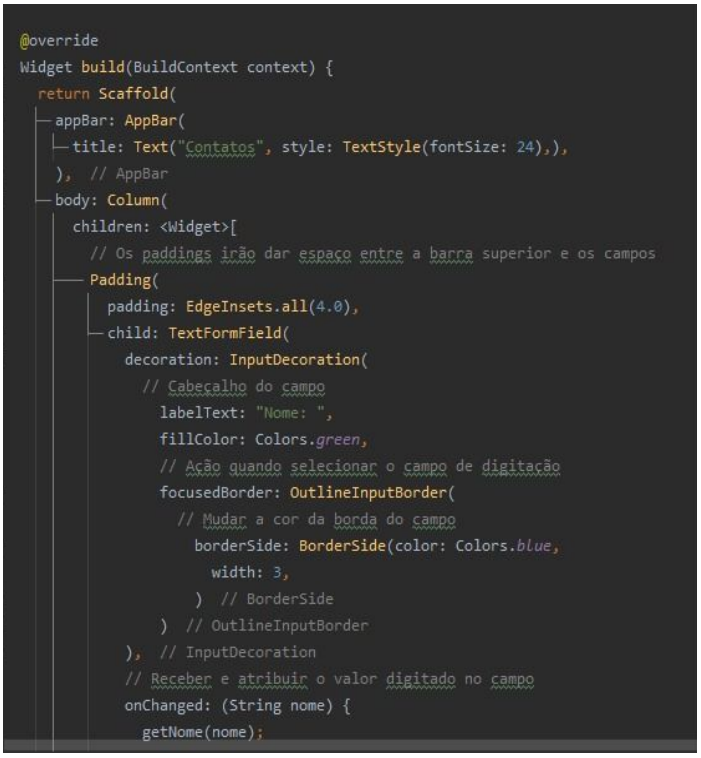
```dart
// Função para registrar o novo contato no banco de dados
CadastrarDados() {
  // Exibirá uma mensagem no terminal da IDE mostrando que a ação foi realizada
  print("Cadastrado");
  // Quando esse método for chamado, irá criar um contato no banco de dados Firebase
  DocumentReference documentReference = FirebaseFirestore.instance.collection("Contatos").doc(Nome);

  // Criar uma referencia para onde cada valor deverá ser alocado
  Map <String, dynamic> contatos = {
    "Nome": Nome,
    "Telefone": Telefone,
    "ID": ID,
    "CEP": CEP,
  };

  // Exibir uma mensagem de sucesso ao cadastrar um novo contato
  documentReference.set(contatos).whenComplete(() => print("$Nome Cadastrado!"));

  // Mostrar a alteração feita no status do contato
  setState(() {
    mensagem = "Cadastrado!";
  });
}
```

```dart
ExcluirDados(){
  // Exibirá uma mensagem no terminal da IDE mostrando que a ação foi realizada
  print("Dados excluídos");
  // Direcionar a "coleção" referente ao banco de dados utilizado
  DocumentReference documentReference = FirebaseFirestore.instance.collection("Contatos").doc(Nome);

  // Exibir uma mensagem indicando a deleção do contato indicado
  documentReference.delete().whenComplete(() => print("$Nome excluído!"));

  // Mostrar a alteração feita no status do contato
  setState(() {
    mensagem = "excluído!";
  });
}

ConsultarDados(){
  // Exibirá uma mensagem no terminal da IDE mostrando que a ação foi realizada
  print("Consulta dos dados");

  dynamic snapshot;

  // Direcionar a "coleção" referente ao banco de dados utilizado
  DocumentReference documentReference = FirebaseFirestore.instance.collection("Contatos").doc(Nome);

  // exibir os dados da consulta no terminal da IDE
  documentReference.get().then((datasnapshot) {
    print(datasnapshot.data());
  });
```

```dart
  });

  // Mostrar a alteração feita no status do contato
  setState(() {
    mensagem = "Consultado!";
  });
}

AtualizarDados(){
  // Exibirá uma mensagem no terminal da IDE mostrando que a ação foi realizada
  print("Dados atualizados");

  // Quando esse método for chamado, irá criar um contato no banco de dados Firebase
  DocumentReference documentReference = FirebaseFirestore.instance.collection("Contatos").doc(Nome);

  // Criar uma referencia para onde cada valor deverá ser alocado
  Map <String, dynamic> contatos = {
    "Nome": Nome,
    "Telefone": Telefone,
    "ID": ID,
    "CEP": CEP,
  };

  // Exibir uma mensagem de sucesso ao cadastrar um novo contato
  documentReference.set(contatos).whenComplete(() => print("$Nome atualizado!"));

  // Mostrar a alteração feita no status do contato
  setState(() {
    mensagem = "atualizado!";
  });
```

5. EXIBIÇÃO:

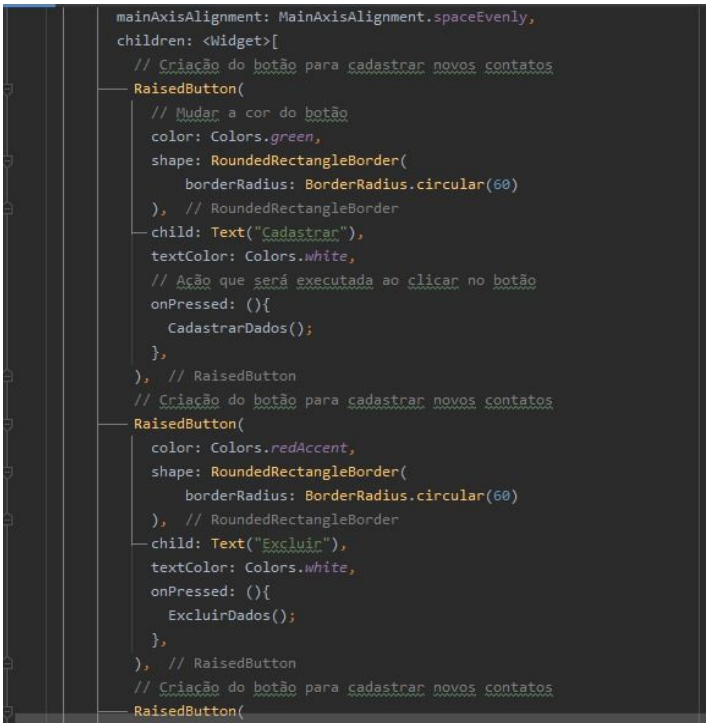
Nesta parte, é exibido o código referente ao layout do aplicativo, com o título do projeto, as cores principais, botões e os demais códigos com outras ações. Antes disso, é estabelecido o comando “@override” para sobrescrever um método já existente. Dentro da estrutura, também são exibidos os campos de digitações dos integrantes da lista de contatos, e juntamente com a atribuição dos valores digitados.



```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Contatos", style: TextStyle(fontSize: 24),),
    ),  // AppBar
    body: Column(
      children: <Widget>[
        // Os paddings irão dar espaço entre a barra superior e os campos
        Padding(
          padding: EdgeInsets.all(4.0),
          child: TextFormField(
            decoration: InputDecoration(
              // Cabecalho do campo
              labelText: "Nome: ",
              fillColor: Colors.green,
              // Ação quando selecionar o campo de digitação
              focusedBorder: OutlineInputBorder(
                // Mudar a cor da borda do campo
                borderSide: BorderSide(color: Colors.blue,
                  width: 3,
                )  // BorderSide
              )  // OutlineInputBorder
            ),  // InputDecoration
            // Receber e atribuir o valor digitado no campo
            onChanged: (String nome) {
              getNome(nome);
```

```
              getNome(nome);
            },
          ),  // TextFormField
        ),  // Padding
        Padding(
          padding: EdgeInsets.all(4.0),
          child: TextFormField(
            decoration: InputDecoration(
                labelText: "Prontuário: ",
                fillColor: Colors.green,
                focusedBorder: OutlineInputBorder(
                    borderSide: BorderSide(color: Colors.blue,
                      width: 3,
                    )  // BorderSide
                )  // OutlineInputBorder
            ),  // InputDecoration
            onChanged: (String id) {
              getID(id);
            },
          ),  // TextFormField
        ),  // Padding
        Padding(
          padding: EdgeInsets.all(4.0),
          child: TextFormField(
            decoration: InputDecoration(
                labelText: "Telefone: ",
                fillColor: Colors.green,
                focusedBorder: OutlineInputBorder(
                    borderSide: BorderSide(color: Colors.blue,
                      width: 3,
```

```
                      width: 3,
                    )  // BorderSide
                )  // OutlineInputBorder
            ),  // InputDecoration
            onChanged: (String tel) {
              getTelefone(tel);
            },
          ),  // TextFormField
        ),  // Padding
        Padding(
          padding: EdgeInsets.all(4.0),
          child: TextFormField(
            decoration: InputDecoration(
                labelText: "Cidade: ",
                fillColor: Colors.green,
                focusedBorder: OutlineInputBorder(
                    borderSide: BorderSide(color: Colors.blue,
                      width: 3,
                    )  // BorderSide
                )  // OutlineInputBorder
            ),  // InputDecoration
            onChanged: (String cep) {
              getCEP(cep);
            },
          ),  // TextFormField
        ),  // Padding
        // Comando para criar botões lado a lado
        Row(
          // Alinhamento dos botões na tela do app
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
```



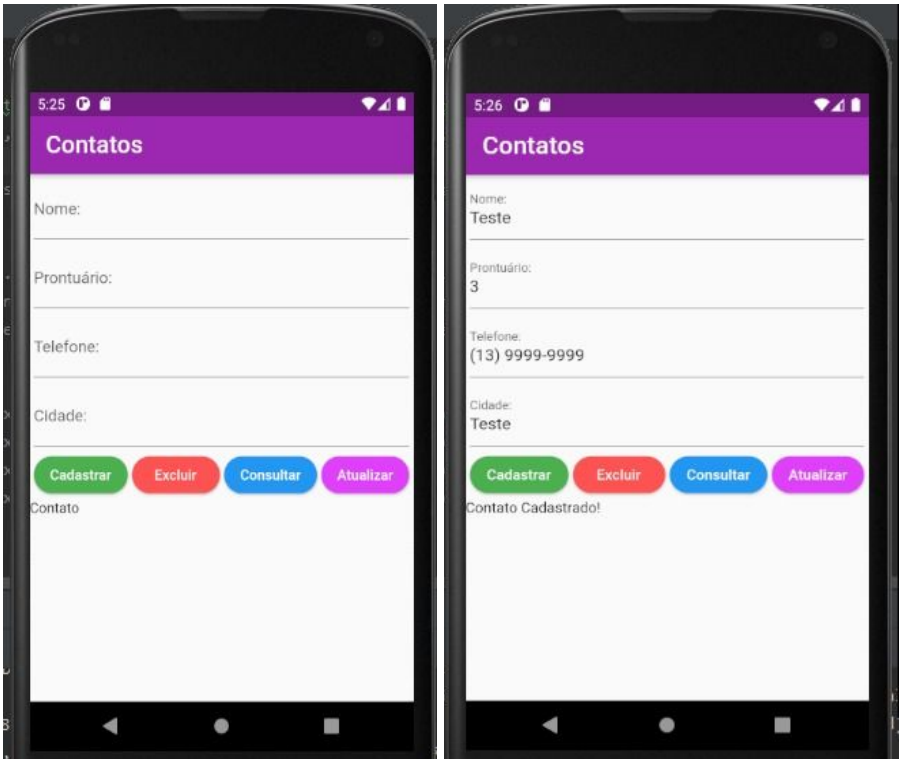
```
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: <Widget>[
            // Criação do botão para cadastrar novos contatos
            RaisedButton(
              // Mudar a cor do botão
              color: Colors.green,
              shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(60)
              ),  // RoundedRectangleBorder
              child: Text("Cadastrar"),
              textColor: Colors.white,
              // Ação que será executada ao clicar no botão
              onPressed: (){
                CadastrarDados();
              },
            ),  // RaisedButton
            // Criação do botão para cadastrar novos contatos
            RaisedButton(
              color: Colors.redAccent,
              shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(60)
              ),  // RoundedRectangleBorder
              child: Text("Excluir"),
              textColor: Colors.white,
              onPressed: (){
                ExcluirDados();
              },
            ),  // RaisedButton
            // Criação do botão para cadastrar novos contatos
            RaisedButton(
```

```
        // Criação do botão para cadastrar novos contatos
        RaisedButton(
          color: Colors.blue,
          shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(20)
          ),  // RoundedRectangleBorder
          child: Text("Consultar"),
          textColor: Colors.white,
          onPressed: (){
            ConsultarDados();
          },
        ),  // RaisedButton
        // Criação do botão para cadastrar novos contatos
        RaisedButton(
          color: Colors.purpleAccent,
          shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(60)
          ),  // RoundedRectangleBorder
          child: Text("Atualizar"),
          textColor: Colors.white,
          onPressed: (){
            AtualizarDados();
          },
        ),  // RaisedButton
      ],  // <Widget>[]
    ),  // Row
    Row(children: <Widget>[Text("Contato $mensagem")],),
    // Comando para exibição, no app, dos contatos existentes
    //StreamBuilder(
      //stream: FirebaseFirestore.instance.collection("Contatos").snapshots
```

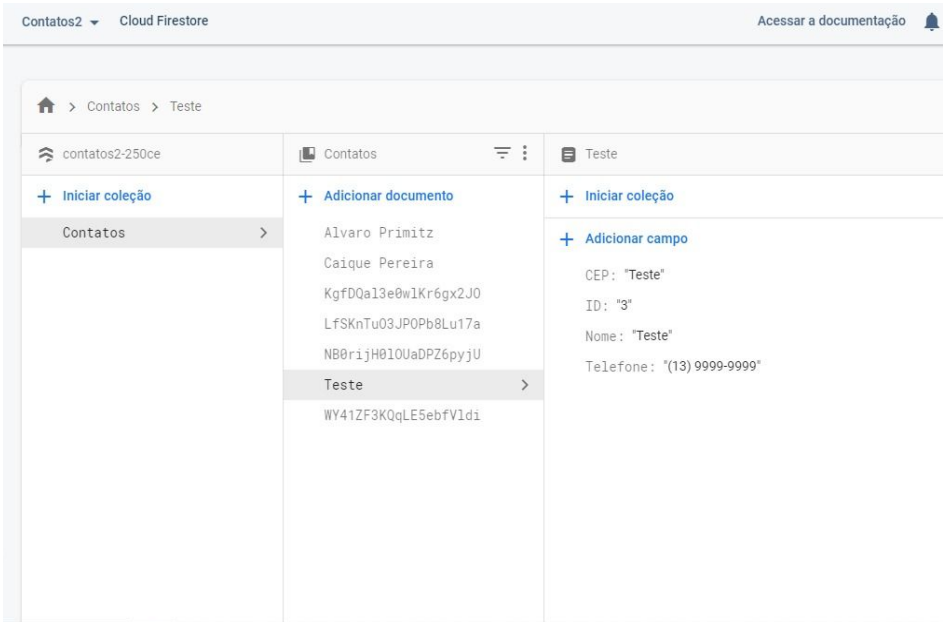
```
      ],  // <Widget>[]
    ),  // Row
    Row(children: <Widget>[Text("Contato $mensagem")],),
    // Comando para exibição, no app, dos contatos existentes
    //StreamBuilder(
      //stream: FirebaseFirestore.instance.collection("Contatos").snapshots()
      // builder: (context, snapshot) {
        //return ListView.builder(
        // itemCount: snapshot.data.toString().length,
         //itemBuilder: (context, index) {
          // DocumentSnapshot documentSnapshot = snapshot.data();
        //   return Row(
            //  children: <Widget>[
              //  Expanded(child: Text(documentSnapshot["Nome"]),),
                //Expanded(child: Text(documentSnapshot["ID"]),),
                //Expanded(child: Text(documentSnapshot["Telefone"]),),
                //Expanded(child: Text(documentSnapshot["CEP"]),)
              //],
            //);
          //},
        //);
      //}
    // Ultimo parenteses do ListView
    //)
  ],  // <Widget>[]
),  // Column
);  // Scaffold
}
}
```

6. TELA DO APLICATIVO:

Após a realização de todos esses procedimentos, a aplicação ficou com o seguinte design, sendo possível cadastrar, consultar, deletar e alterar os participantes de uma lista de contatos:



Posteriormente a digitação, os dados já vão diretamente para o banco de dados Firebase:



Diante disso, quando ocorre a exclusão, temos a mesma ação no banco de dados: