Desenvolvimento Mobile 1
Aula 08

Prof. Me Daniel Vieira

SENAI

Agenda

1- Flutter Widgets
2- Tipos de layout de App : Single Layout e Multi Layout
3-Exemplos
4 -Criando projeto Flutter no VSCode
5 - Criando emulador Android

Flutter Widgets

- **Widgets básicos**

Os widgets básicos são fundamentais para construir qualquer interface no Flutter

Container: Um contêiner que pode ter padding, margens, bordas e um fundo.

Row e Column: Alinhamento horizontal (Row) ou vertical (Column) de widgets.

Text: Exibe texto na tela.

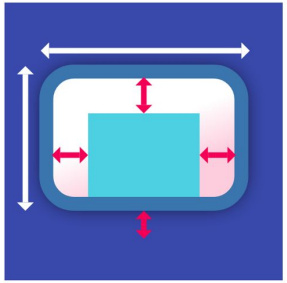
Image: Exibe imagens de várias fontes.

Stack: Empilha widgets uns sobre os outros

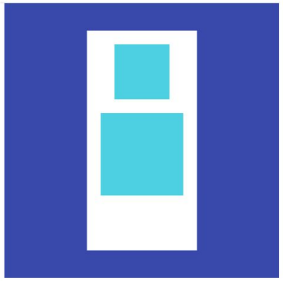
Scaffold: oferece áreas pré-definidas que facilitam a organização de widgets na tela.

Flutter Widgets

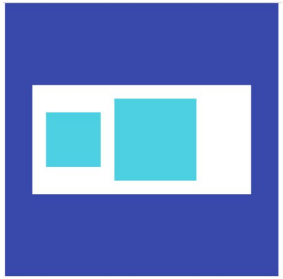
https://docs.flutter.dev/ui/widgets/basics
https://m3.material.io/components/icon-buttons/overview



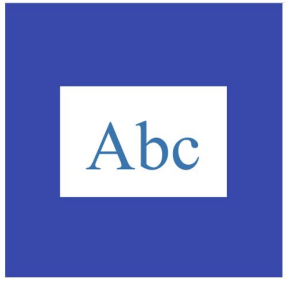
Container



Column

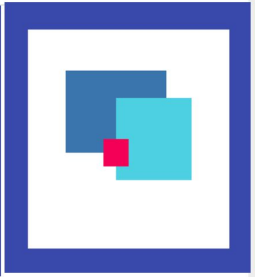


Row

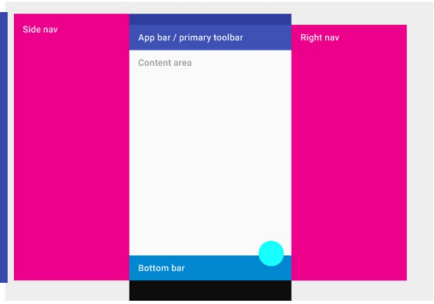


Text

Image



Stack



Scaffold

Flutter Widgets

Os códigos de cada widget pode ser consultado na própria documentação do Flutter conforme o link https://docs.flutter.dev/ui/widgets/basics

Flutter Widgets

- **Widgets de layout**

Os Widgets utilizados para layout são :

Align: Alinha um único widget dentro de um espaço.

Center: Centraliza um widget dentro do seu pai.

Expanded: Ajusta o tamanho de um widget dentro de um Row ou Column.

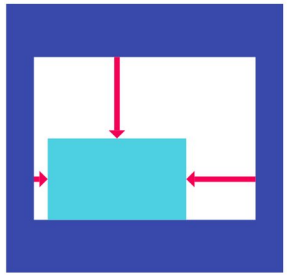
SizedBox: Define um tamanho fixo para um widget.

Padding: Adiciona espaço ao redor de um widget.

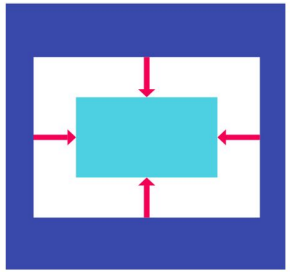
GridView: Exibe widgets em um layout de grade.

ListView: Exibe widgets em um formato de lista rolável.

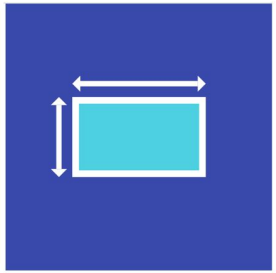
Flutter Widgets de layout



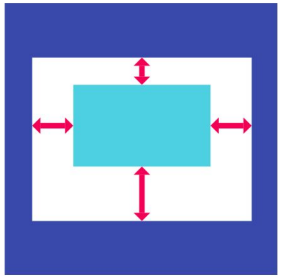
Align



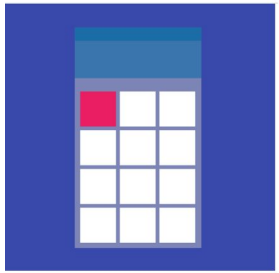
Center



SizeBox

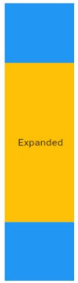


Padding



GridView

ListView



Expanded

Widgets interativos

Briefing: Figura para exibir os Widgets de layout

Para mais informações sobre os widgets de layout acessar o link

https://docs.flutter.dev/ui/widgets/layout

Widgets interativos

- **Widgets interativos**

Os widgets interativos são widgets que respondem a interações do usuário
Na Tabela 32 é possível visualizar o código exemplo para o ElevatedButton
Um botão elevado que reage ao clique do usuário, realizando uma ação. Tem um fundo com elevação (sombra), destacando-se do restante da interface.

Widgets interativos - Elevated Button

```
/*
onpressed: é  o evento que ocorre quando o botão é pressionado

child: é uma propriedade do ElevatedButton que permite inserir um texto no botão
*/

ElevatedButton(
  onPressed: () {
    print('ElevatedButton clicado!');
  },
  child: Text('Clique Aqui'),
)
```

Widgets interativos - Text Button

```
1. /*
2. onpressed: é  o evento que ocorre quando o botão é pressionado
3.
4. child: é uma propriedade do TextButton que permite inserir um texto no botão
5. */
6. TextButton(
7.   onPressed: () {
8.     print('TextButton clicado!');
9.   },
10.  child: Text('Clique Aqui'),
11. )
12.
```

Widgets interativos - Icon Button

```
1. /*
2. onpressed: é  o evento que ocorre quando o botão é pressionado
3.
4. child: é uma propriedade do IconButton que permite inserir um texto no botão
5. */
6. IconButton(
7.   onPressed: () {
8.     print('IconButton clicado!');
9.   },
10.  icon: Icon(Icons.settings),
11.
```

Widgets interativos - Floating Action Button

```
1. /*
2. onpressed: é  o evento que ocorre quando o botão é pressionado
3.
4. child: é uma propriedade do FloatingActionButton que permite inserir um texto no botão
5. */
6. FloatingActionButton(
7.   onPressed: () {
8.     print('FloatingActionButton clicado!');
9.   },
10.  child: Icon(Icons.add),
11. )
12.
```

Widgets interativos - Gesture Detector

```
1. /*
 2. onTap: Um callback que será acionado quando o usuário tocar no widget. Neste caso,
ele imprime "Widget tocado!" no console.
 3. child: O filho do GestureDetector, que neste caso é um Container com largura e
altura de 100 pixels e cor azul.
 4. */
 5.
 6. GestureDetector(
 7.   onTap: () {
 8.     print('Widget tocado!');
 9.   },
10.   child: Container(
11.     width: 100,
12.     height: 100,
13.     color: Colors.blue,
14.   ),)
```

Widgets interativos - switch

```
/*
bool isSwitched = false;

Representa o estado inicial do botão Switch.
O estado é false, indicando que o botão começa na posição desligada.
Switch:

value: Define o estado atual do Switch. Aqui ele usa a variável isSwitched.
onChanged: Callback acionado ao alternar o estado do botão. Atualiza o estado de isSwitched e imprime o novo valor no console.
*/

bool isSwitched = false;

Switch(
  value: isSwitched,
  onChanged: (value) {
    isSwitched = value;
    print('Switch: $isSwitched');
  },
)

```

Widgets interativos - slider

```
1. /*
 2. double sliderValue = 0.5;
 3.
 4. Representa o valor inicial do slider, posicionado no meio do intervalo, já que o valor mínimo é 0 e o máximo
é 1.
 5. Slider:
 6.
 7. value: Define o valor atual do slider (nesse caso, 0.5).
 8. min e max: Estabelecem os limites inferior e superior do intervalo do slider.
 9. onChanged: Callback que é acionado sempre que o usuário move o controle deslizante. Aqui, o valor atualizado
é atribuído à variável sliderValue e exibido no console.
10. */
11.
12. double sliderValue = 0.5;
13.
14. Slider(
15.   value: sliderValue,
16.   min: 0,
17.   max: 1,
18.   onChanged: (value) {
19.     sliderValue = value;
20.     print('Slider: $sliderValue');
21.   },
22. )
23.
```

Widgets interativos - checkbox

```
/*
bool isChecked = false;

Define o estado inicial do checkbox como desmarcado (false).
Checkbox:

value: Controla o estado atual do checkbox (true para marcado, false para desmarcado).
onChanged: Callback acionado quando o estado do checkbox é alterado. O novo estado é passado como parâmetro (value).
value!:

O operador ! é usado para indicar que o valor nunca será nulo (non-null assertion). Isso é necessário porque o parâmetro value pode ser null se o checkbox estiver desativado.
*/

bool isChecked = false;

Checkbox(
  value: isChecked,
  onChanged: (value) {
    isChecked = value!;
    print('Checkbox: $isChecked');
  },
)

```

Widgets interativos - Radio

```dart
/*
int? selectedOption = 1;:

A variável selectedOption armazena a opção selecionada. Inicialmente, ela é definida como 1, marcando o primeiro botão de rádio.
Radio:

value: O valor associado ao botão de rádio específico.
groupValue: O valor atualmente selecionado no grupo. Apenas o botão de rádio cujo value coincide com groupValue estará marcado.
onChanged: Callback acionado quando o botão de rádio é selecionado. Aqui, a variável selectedOption é atualizada para refletir a nova seleção.
*/
int? selectedOption = 1;

Column(
  children: [
    Radio<int>(
      value: 1,
      groupValue: selectedOption,
      onChanged: (value) {
        selectedOption = value;
        print('Radio: $selectedOption');
      },
    ),
    Radio<int>(
      value: 2,
      groupValue: selectedOption,
      onChanged: (value) {
        selectedOption = value;
        print('Radio: $selectedOption');
      },
    ),
  ],
)

```

Widgets interativos - TextField

```
/*
onChanged:

Callback acionado toda vez que o texto dentro do campo muda. O valor inserido pelo usuário é recebido como parâmetro (text) e, nesse caso, é exibido no console com o comando print.
decoration:

Personaliza a aparência do TextField. Aqui está sendo usado o widget InputDecoration com:
labelText: Exibe um rótulo acima do campo de texto quando ele está em foco ou preenchido.
border: Define um contorno ao redor do campo de texto usando OutlineInputBorder.
*/

TextField(
  onChanged: (text) {
    print('Texto inserido: $text');
  },
  decoration: InputDecoration(
    labelText: 'Digite algo',
    border: OutlineInputBorder(),
  ),
)
```

Scaffold

```
1. /*
  2.       import 'package:flutter/material.dart';
  3.
  4. Esse comando importa o pacote material.dart do Flutter, que contém todos os widgets, temas e
outros recursos do Material Design. O Material Design é um padrão visual desenvolvido pelo Google, que
fornece um estilo unificado para aplicativos.
  5.
  6. 2. void main() { runApp(MyApp()); }
  7. Essa função é o ponto de entrada do aplicativo Flutter. A função runApp() inicializa o aplicativo,
e o widget MyApp é passado como o widget raiz.
  8.
  9. 3. class MyApp extends StatelessWidget
 10. Aqui, estamos criando uma classe MyApp, que estende StatelessWidget. Um StatelessWidget é um
widget que não mantém estado interno (ou seja, seu conteúdo não muda dinamicamente).
 11. O MyApp é o widget principal do aplicativo.
 12.
 13. 4. @override Widget build(BuildContext context)
 14. O método build é obrigatório para qualquer widget e é onde a interface do usuário é construída.
Ele retorna um widget MaterialApp, que configura o aplicativo com as convenções de design do Material
Design.
 15.
 16. 5. MaterialApp
 17. MaterialApp é um widget que envolve o aplicativo e fornece configuração para temas, navegação e
outros aspectos globais do aplicativo.
 18. A propriedade home do MaterialApp define a tela inicial do aplicativo.
 19.
 20. 6. Scaffold
 21.
 22. Scaffold é um widget que fornece uma estrutura básica de layout para um aplicativo Material
Design. Ele cria a base da interface, onde podemos adicionar elementos como a barra de app (AppBar), o
corpo (body), rodapés (bottomNavigationBar), entre outros.
 23. Neste caso, o Scaffold define a estrutura para a tela inicial, que inclui o appBar e o body.
```

Scaffold

```
24.
25. 7. AppBar
26. AppBar é o widget que representa a barra superior do aplicativo. Ele é usado para exibir o título,
ícones de ação e ícones de navegação.
27. Aqui, o AppBar contém várias propriedades configuradas:
28.       title: Define o título da barra. Neste exemplo, title: Text("Meu App") exibe "Meu App" no
centro da AppBar.
29.       backgroundColor: Define a cor de fundo da AppBar, configurada para Colors.blue, o que torna
a barra azul.
30.       actions: É uma lista de widgets que aparecem no lado direito da AppBar. Geralmente, são
usados IconButtons para representar ações rápidas.
31.
32. 8. actions: [ ... ]
33. A propriedade actions é uma lista de widgets (no caso, IconButtons) que serão exibidos à direita
do título na AppBar.
34. IconButton: Cada IconButton é um botão interativo que contém um ícone e uma ação associada.
35. Icon(Icons.search): O primeiro botão exibe um ícone de lupa (search), que geralmente é usado para
uma funcionalidade de busca.
36. onPressed: Define a ação que será executada quando o botão for pressionado. Aqui, ele está vazio
(// Ação de busca), mas você poderia adicionar um código específico para iniciar uma busca.
37. Icon(Icons.more_vert): O segundo botão exibe um ícone de três pontos verticais (more_vert), que
geralmente representa um menu ou opções adicionais.
38. onPressed: O callback para este botão também está vazio (// Outra ação), mas aqui você pode
adicionar a funcionalidade para abrir um menu, por exemplo.
39.
```

Scaffold

```
40. 9. body: Center(...)
 41.
 42. A propriedade body define o conteúdo principal da tela. Neste exemplo, usamos um widget Center
para centralizar seu conteúdo filho.
 43. Dentro do Center, há um widget Text que exibe o texto "Conteúdo principal". Esse texto aparecerá
centralizado na tela.
 44.
 45. 10. child: Text("Conteúdo principal")
 46. Text é um widget que exibe texto na interface.
 47. "Conteúdo principal" é o texto que será exibido no centro da tela, como definido pelo widget
Center.
 48.
 49.
 50. */
 51.
```

Scaffold

```dart
// Pacote importado para criar a interface utilizando o Material Design
import 'package:flutter/material.dart';

// Função principal do aplicativo, ponto de entrada da aplicação
void main() {
  // Inicializa o aplicativo e chama a classe MyApp, que define a interface do app
  runApp(MyApp());
}

```

Scaffold

```
// Classe principal do aplicativo, do tipo StatelessWidget
// StatelessWidget significa que esta classe não possui estado dinâmico (imutável)
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Método build constrói a interface do aplicativo
    return MaterialApp(
      // Define a tela inicial do aplicativo
      home: Scaffold(
        // AppBar é a barra superior da tela
        appBar: AppBar(
          // Define o título exibido na AppBar
          title: Text("Meu App"),
          // Define a cor de fundo da AppBar
          backgroundColor: Colors.blue,
          // Define os ícones de ação na AppBar
          actions: [
            IconButton(
              // Ícone de busca
              icon: Icon(Icons.search),
              // Define a ação que será executada ao pressionar o botão
              onPressed: () {
                // Ação de busca (a ser implementada)
                print('Busca acionada');
              },
            ),
```

Scaffold

```
IconButton(
 88.               // Ícone de "mais opções"
 89.               icon: Icon(Icons.more_vert),
 90.               // Define a ação que será executada ao pressionar o botão
 91.               onPressed: () {
 92.                 // Outra ação (a ser implementada)
 93.                 print('Mais opções acionadas');
 94.               },
 95.             ),
 96.           ],
 97.         ),
 98.         // Corpo principal da interface do aplicativo
 99.         body: Center(
100.           // Define o widget centralizado na tela
101.           child: Text("Conteúdo principal"), // Exibe um texto simples no centro
102.         ),
103.       ),
104.     );
105.   }
106. }
107.
```

Tipos de layout

- **Single-Child Layout**: Contém um único widget filho, como Container e Center.

Tipos de layout

```
/*
1. class MyApp extends
StatelessWidget

MyApp é uma classe que estende StatelessWidget, que significa que ela não possui estado interno e não muda ao longo do tempo.
Esta classe é a estrutura principal do aplicativo e define a interface básica.

2. @override Widget build(BuildContext context)
O método build é obrigatório para qualquer widget no Flutter. É onde a interface é construída. Aqui, retornamos um widget MaterialApp, que é a base para qualquer aplicativo Flutter que use o design de Material.

3. MaterialApp
MaterialApp é um widget que configura o aplicativo para seguir o estilo Material Design, fornecendo temas, navegação, e outras funcionalidades globais.
A propriedade home define a tela inicial do aplicativo. Aqui, estamos passando um Scaffold como a tela inicial.

4. Scaffold
Scaffold é um widget de layout básico que fornece uma estrutura de página padrão com suporte para AppBar, Drawer, FloatingActionButton, entre outros.
Aqui, estamos usando Scaffold para estruturar a tela com uma barra de aplicativo (AppBar) e um corpo (body).
```

Tipos de layout

18.
19. 5. AppBar
20. AppBar é a barra superior da aplicação, geralmente usada para exibir o título da página e ações rápidas.
21. title: Define o título exibido na AppBar. Aqui, usamos um widget Text com o texto "Exemplo de Single-Child Layout".
22.
23. 6. body: Center(...)
24. A propriedade body define o conteúdo principal da tela. Usamos um widget Center, que é um Single-Child Layout, para centralizar seu conteúdo filho.
25. O Center aceita apenas um widget filho e posiciona esse widget no centro da tela.
26.
27. 7. Container
28. Container é o único filho do Center e é usado para estilizar e alinhar o conteúdo da interface.
29. width e height: Define a largura e a altura do Container para 200x200 pixels, criando um quadrado centralizado.
30. alignment: Define o alinhamento do conteúdo dentro do Container. Aqui, usamos Alignment.center para centralizar o texto dentro do Container.
31. decoration: Usamos BoxDecoration para adicionar estilo ao Container.
32. color: Define a cor de fundo do Container como azul (Colors.blue).
33. borderRadius: Define bordas arredondadas com um raio de 12 pixels, arredondando os cantos do Container.

Tipos de layout

```
34. 8. Text
35. Text é o conteúdo do Container. Ele exibe o texto "Conteúdo centralizado".
36. style: Usamos TextStyle para estilizar o texto.
37.
38.  color: Define a cor do texto como branco (Colors.white).
39.
40. fontSize: Define o tamanho da fonte como 16 pixels.
41. textAlign: Define o alinhamento do texto dentro do Text como TextAlign.center, centralizando o
texto no Container.
42.
43. */
44. // Pacote importado para criar a interface utilizando o Material Design
```

Tipos de layout

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

// Função principal do aplicativo, ponto de entrada da aplicação
void main() {
  // Inicializa o aplicativo e chama a classe MyApp, que define a interface do app
  runApp(MyApp());
}

// Classe principal do aplicativo, do tipo StatelessWidget
// StatelessWidget significa que esta classe não possui estado dinâmico (imutável)
class MyApp extends StatelessWidget {
```

Tipos de layout

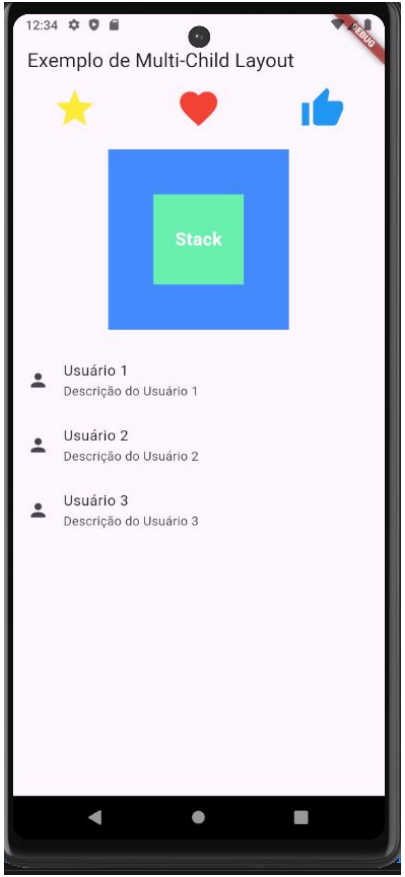
```dart
@override
 Widget build(BuildContext context) {
   // Método build constrói a interface do aplicativo
   return MaterialApp(
     // Define a tela inicial do aplicativo
     home: Scaffold(
       // AppBar é a barra superior da tela
       appBar: AppBar(
         // Define o título exibido na AppBar
         title: Text("Meu App"),
         // Define a cor de fundo da AppBar
         backgroundColor: Colors.blue,
         // Define os ícones de ação na AppBar
         actions: [
           IconButton(
             // Ícone de busca
             icon: Icon(Icons.search),
             // Define a ação que será executada ao pressionar o botão
             onPressed: () {
               // Ação de busca (a ser implementada)
               print('Busca acionada');
             },
           ),
```

Tipos de layout

```
79.             IconButton(
80.                // Ícone de "mais opções"
81.                icon: Icon(Icons.more_vert),
82.                // Define a ação que será executada ao pressionar o botão
83.                onPressed: () {
84.                  // Outra ação (a ser implementada)
85.                  print('Mais opções acionadas');
86.                },
87.              ),
88.            ],
89.          ),
90.          // Corpo principal da interface do aplicativo
91.          body: Center(
92.            // Define o widget centralizado na tela
93.            child: Text("Conteúdo principal"), // Exibe um texto simples no centro
94.          ),
95.        ),
96.      );
97.    }
98. }
99.
100.
```

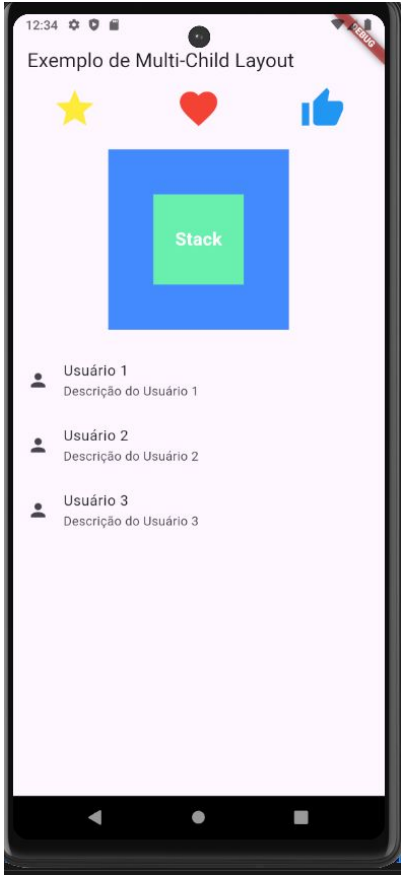
Tipos de layout

- **Multi-Child Layout**: Contém múltiplos widgets filhos, como Column, Row, Stack, e ListView.



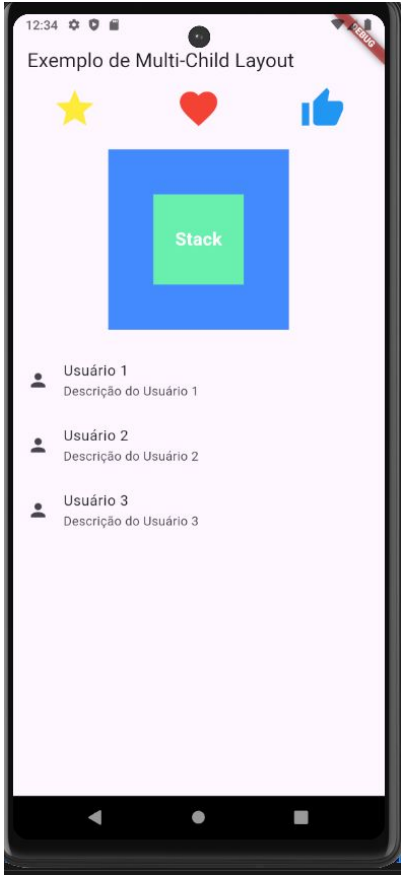
Tipos de layout

```
/*
AppBar:

Barra superior com o título "Exemplo de Multi-Child Layout".
Column:

Organiza os widgets verticalmente.
Propriedades:
crossAxisAlignment: Define o alinhamento horizontal dos filhos.
children: Contém os widgets filhos (Row, Stack, ListView, etc.).
Row:

Organiza os widgets horizontalmente.
Propriedades:
mainAxisAlignment: Espaça os widgets igualmente.
Contém três ícones (star, favorite, thumb_up).
SizedBox:

Espaçamento entre widgets.
Stack:
```



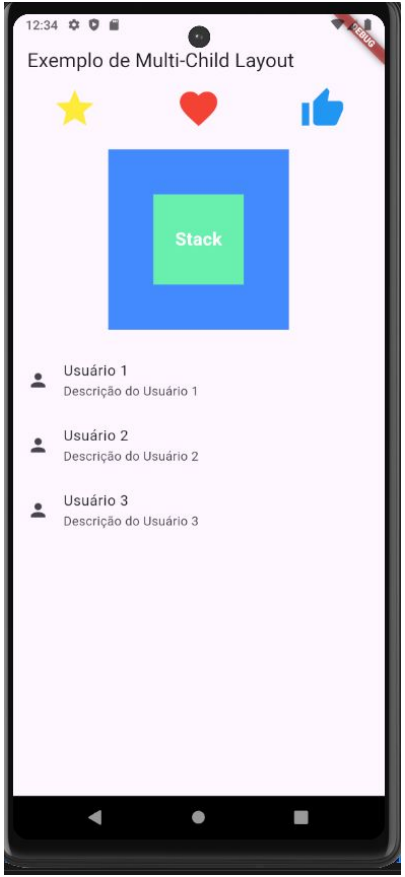
Tipos de layout

```
21.
 22. Sobrepõe widgets uns aos outros.
 23. Contém dois Containers e um Text centralizado.
 24. ListView:
 25.
 26. Um widget rolável que lista elementos.
 27. Contém vários ListTiles representando usuários fictícios.
 28. Cada ListTile tem um ícone (leading), título (title) e descrição (subtitle).
 29. Expanded:
 30.
 31. Permite que o ListView preencha o espaço disponível na tela.
 32. */
```



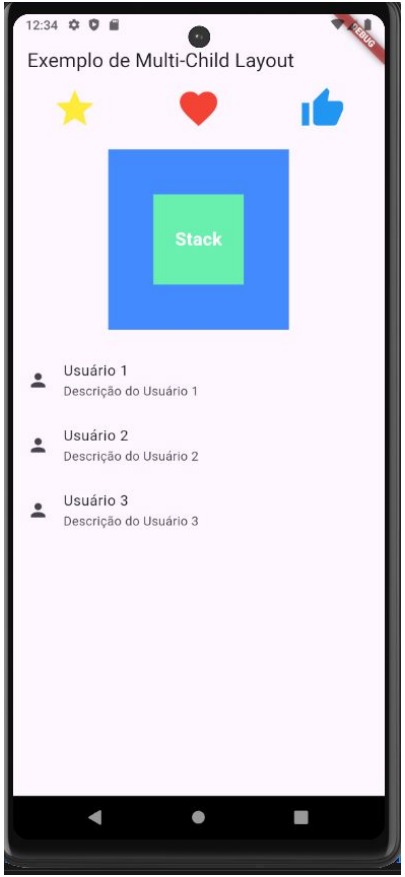
Tipos de layout

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        // AppBar:
        // Barra superior com o título "Exemplo de Multi-Child Layout".
        appBar: AppBar(
          title: Text("Exemplo de Multi-Child Layout"),
        ),
```



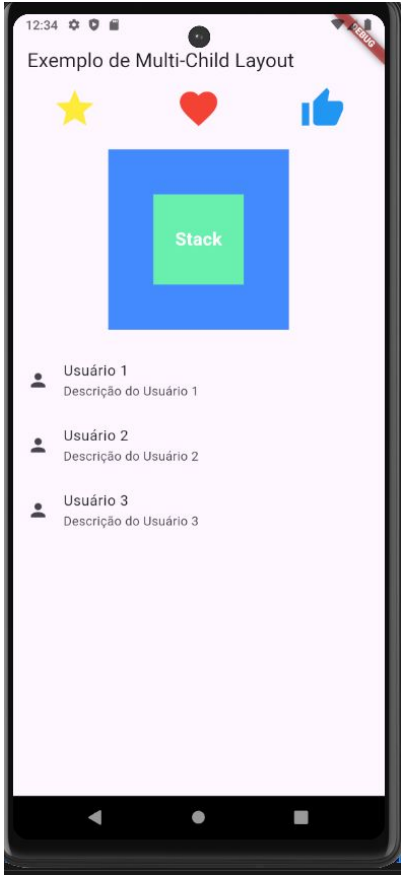
Tipos de layout

```
49.         body: Column(
50.           // Column:
51.           // Organiza os widgets verticalmente.
52.           // Propriedades:
53.           // - crossAxisAlignment: Define o alinhamento horizontal dos filhos.
54.           // - children: Contém os widgets filhos (Row, Stack, ListView, etc.).
55.           crossAxisAlignment: CrossAxisAlignment.stretch,
56.           children: [
57.             // Row:
58.             // Organiza os widgets horizontalmente.
59.             // Propriedades:
60.             // - mainAxisAlignment: Espaça os widgets igualmente.
61.             // Contém três ícones (star, favorite, thumb_up).
62.             Row(
63.               mainAxisAlignment: MainAxisAlignment.spaceAround,
64.               children: [
65.                 Icon(Icons.star, color: Colors.yellow, size: 50),
66.                 Icon(Icons.favorite, color: Colors.red, size: 50),
67.                 Icon(Icons.thumb_up, color: Colors.blue, size: 50),
68.               ],
69.             ),
```



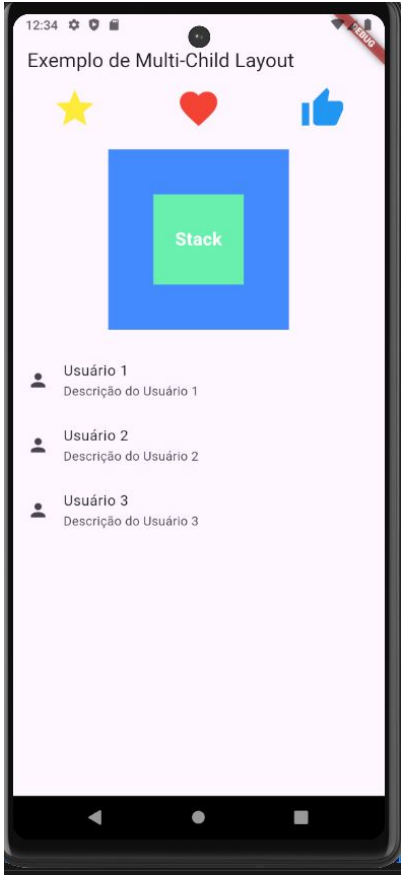
Tipos de layout

```
70.             // SizedBox:
71.              // Espaçamento entre widgets.
72.              SizedBox(height: 20),
73.              // Stack:
74.              // Sobrepõe widgets uns aos outros.
75.              // Contém dois Containers e um Text centralizado.
76.              Stack(
77.                alignment: Alignment.center,
78.                children: [
79.                  Container(
80.                    width: 200,
81.                    height: 200,
82.                    color: Colors.blueAccent,
83.                  ),
84.                  Container(
85.                    width: 100,
86.                    height: 100,
87.                    color: Colors.greenAccent,
88.                  ),
```



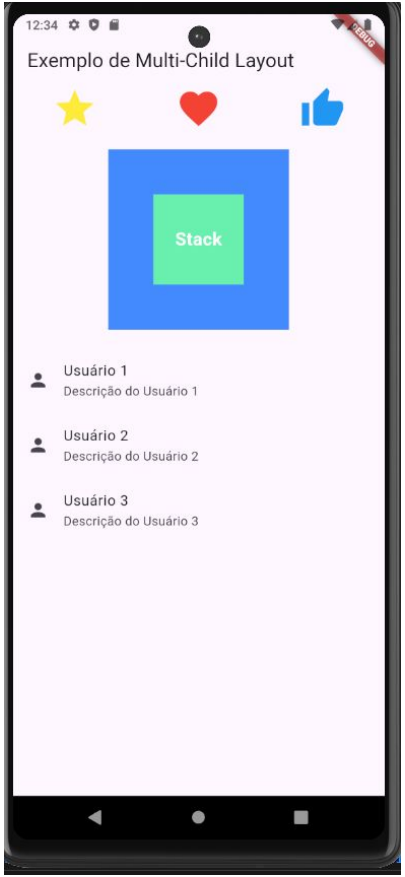
Tipos de layout

```
                Text(
                  "Stack",
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: 20,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ],
            ),
            // SizedBox:
            // Espaçamento entre widgets.
            SizedBox(height: 20),
            // Expanded:
            // Permite que o ListView preencha o
```



Tipos de layout

```
104.             Expanded(
105.               // ListView:
106.               // Um widget rolável que lista elementos.
107.               // Contém vários ListTiles representando usuários fictícios.
108.               // Cada ListTile tem um ícone (leading), título (title) e descrição (subtitle).
109.               child: ListView(
110.                 children: [
111.                   ListTile(
112.                     leading: Icon(Icons.person),
113.                     title: Text("Usuário 1"),
114.                     subtitle: Text("Descrição do Usuário 1"),
115.                   ),
116.                   ListTile(
117.                     leading: Icon(Icons.person),
118.                     title: Text("Usuário 2"),
119.                     subtitle: Text("Descrição do Usuário 2"),
120.                   ),
121.                   ListTile(
122.                     leading: Icon(Icons.person),
123.                     title: Text("Usuário 3"),
124.                     subtitle: Text("Descrição do Usuário 3"),
125.                   ),
126.                 ],
127.               ),
128.             ),
129.           ],
130.         ),
131.       ),
132.     );
133.   }
134. }
135.
136.
```



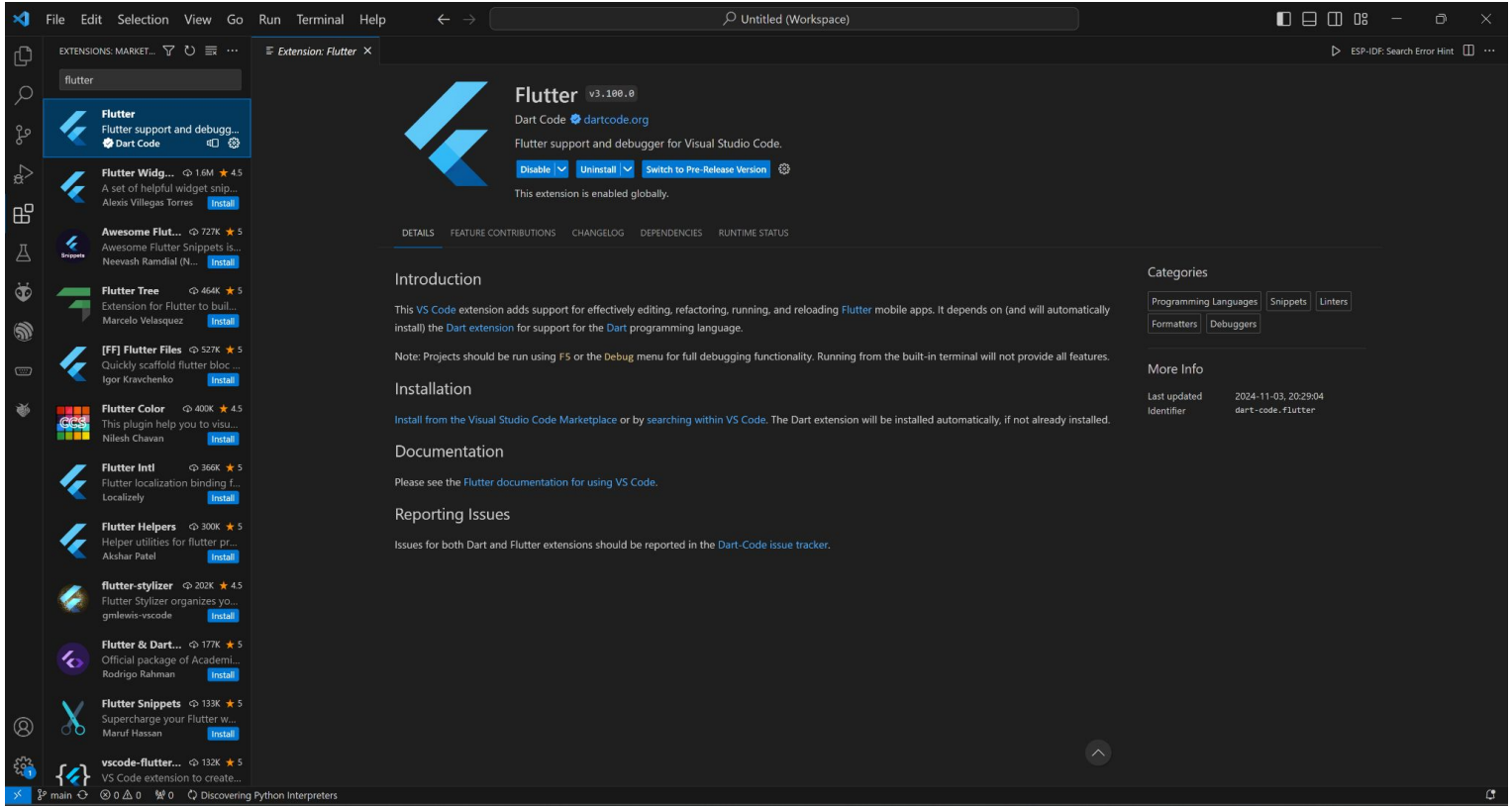
Criando projeto no VSCODE

Para criar um projeto de aplicativo mobile utilizando o framework Flutter com a IDE VSCode é necessário seguir os passos listados a seguir:

1º Abrir o VSCode

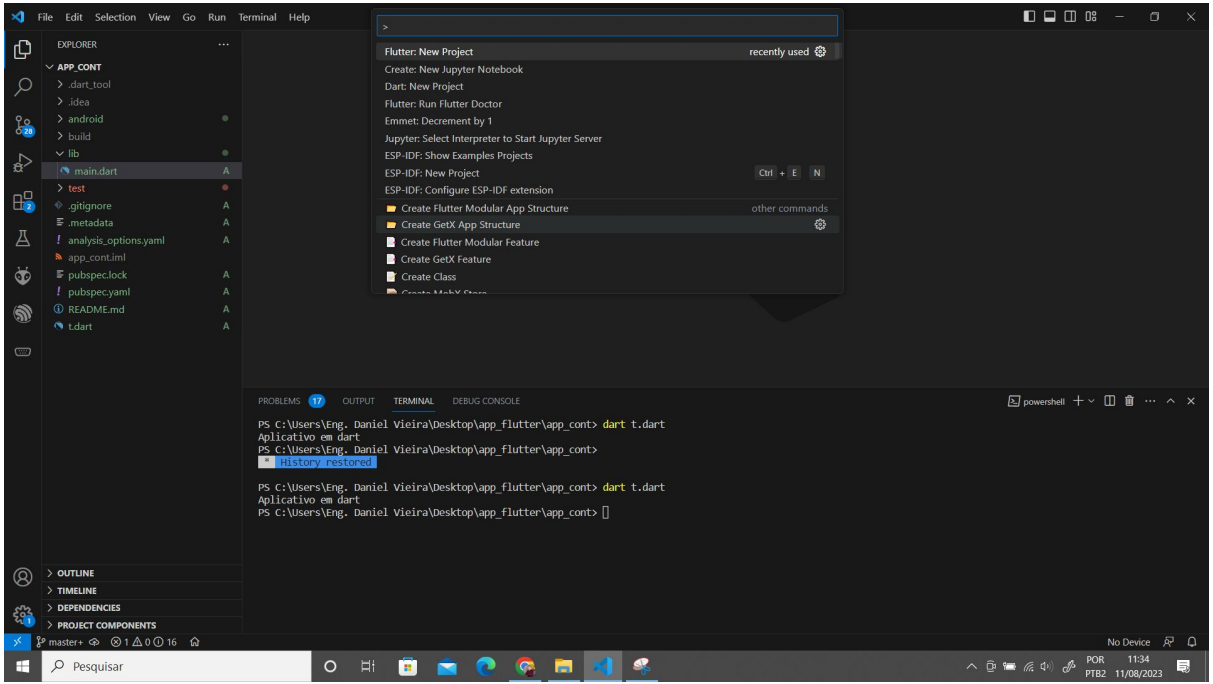
2º Instalar os plugins.

Criando projeto no VSCODE



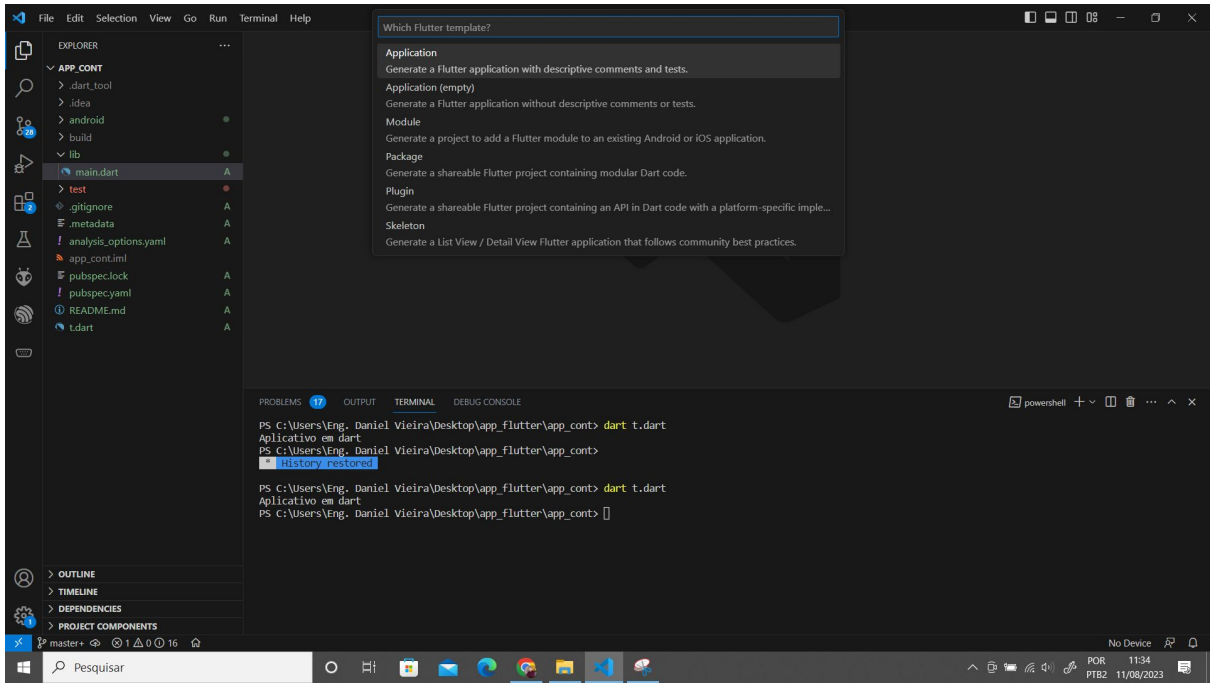
Criando projeto no VSCODE

3º Apertar a tecla F1, após apertar em F1 clicar em Flutter New Project conforme



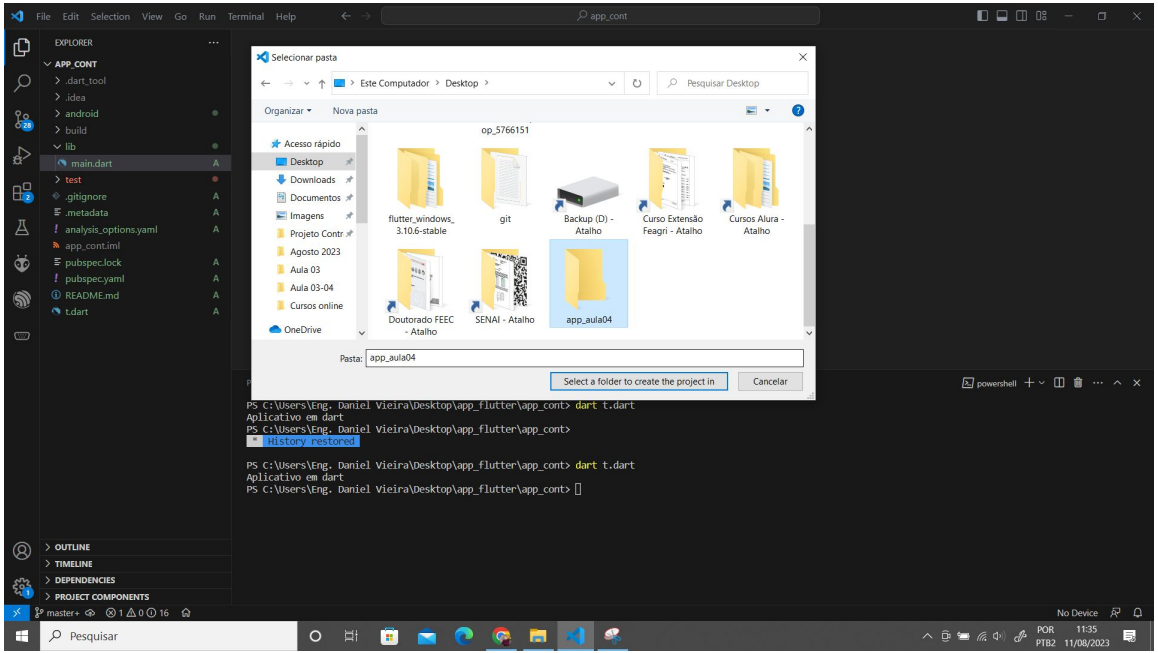
Criando projeto no VSCODE

4º Selecionar a opção Application conforme a figura abaixo



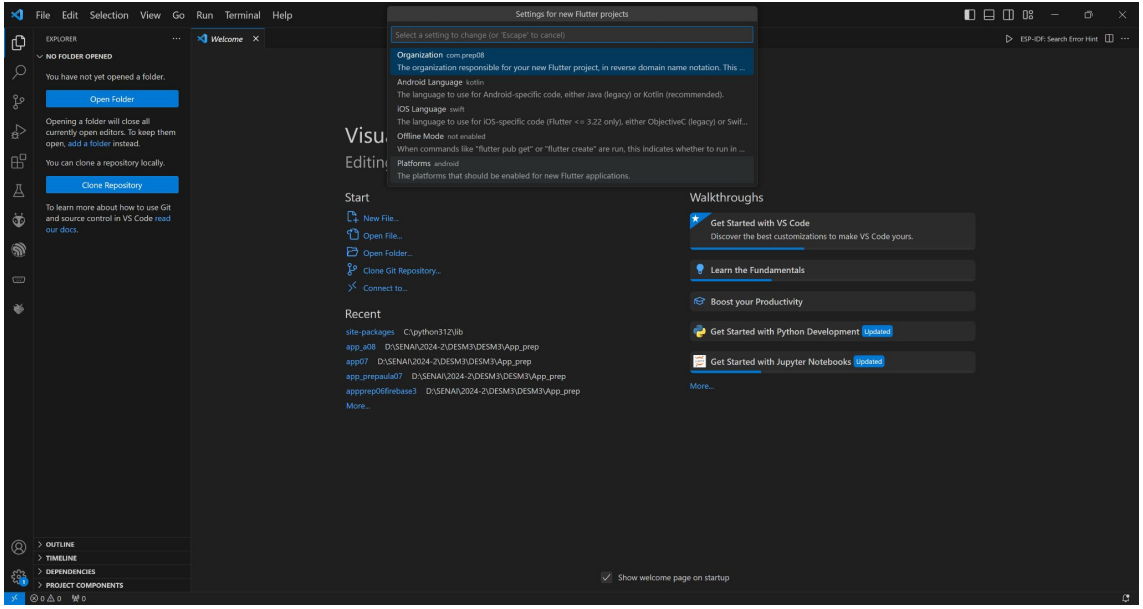
Criando projeto no VSCODE

5º Selecionar uma pasta para salvar o projeto conforme a Figura. A pasta não deve começar com letras maiúsculas, números, não pode ter espaço no nome da pasta



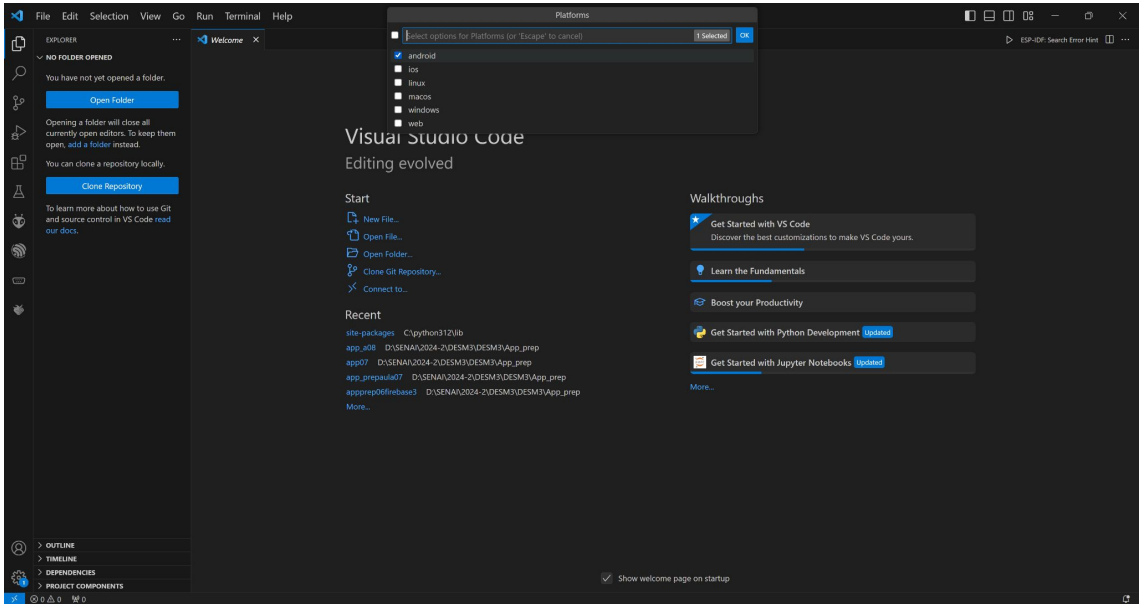
Criando projeto no VSCODE

6º Clicar na engrenagem para configurar as plataformas mobile que o projeto será executado conforme a Figura



Criando projeto no VSCODE

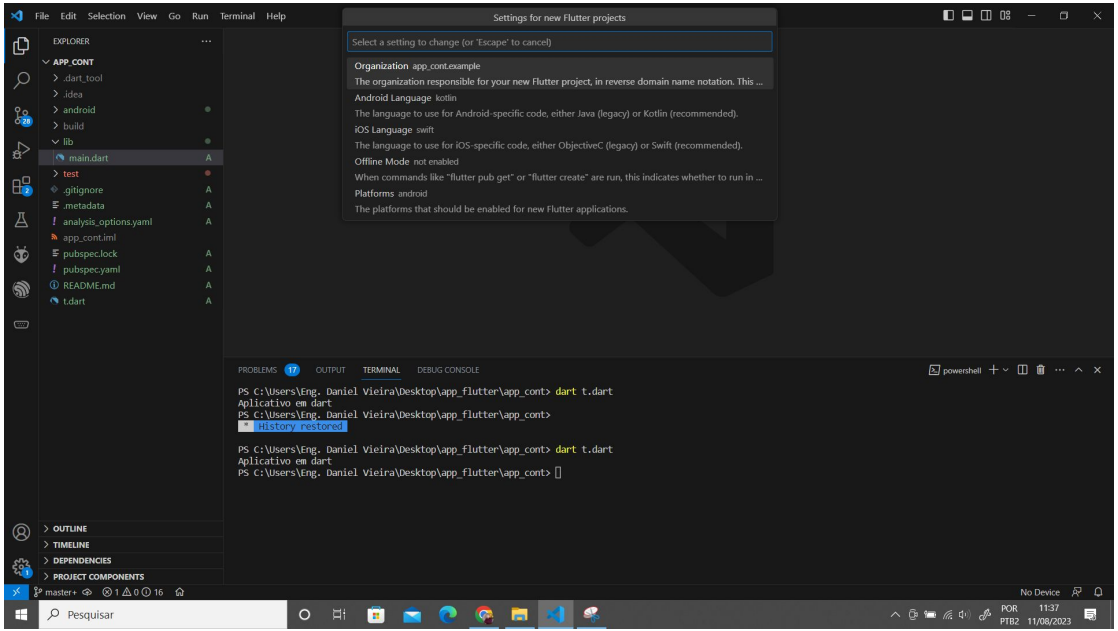
7º Selecionar a opção Android, realizada essa opção apertar enter conforme a Figura



Criando projeto no VSCODE

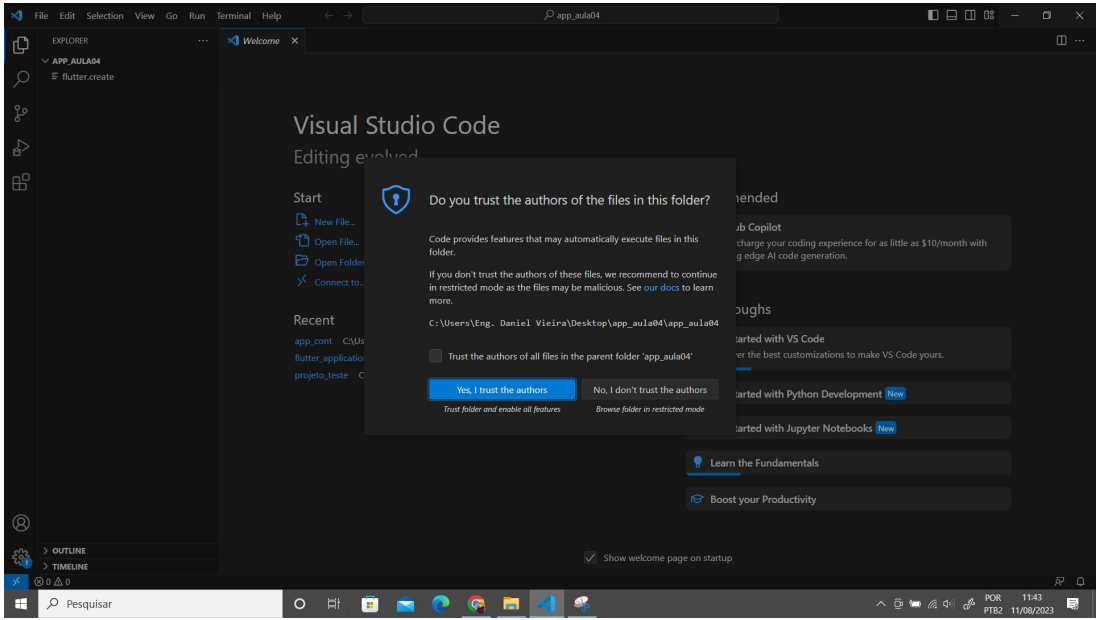
8º Passo Clicar em Organization conforme a Figura 48

Organization é o parâmetro que indica o domínio da empresa que desenvolveu o app. É um parâmetro importante para publicação do APP em lojas virtuais.



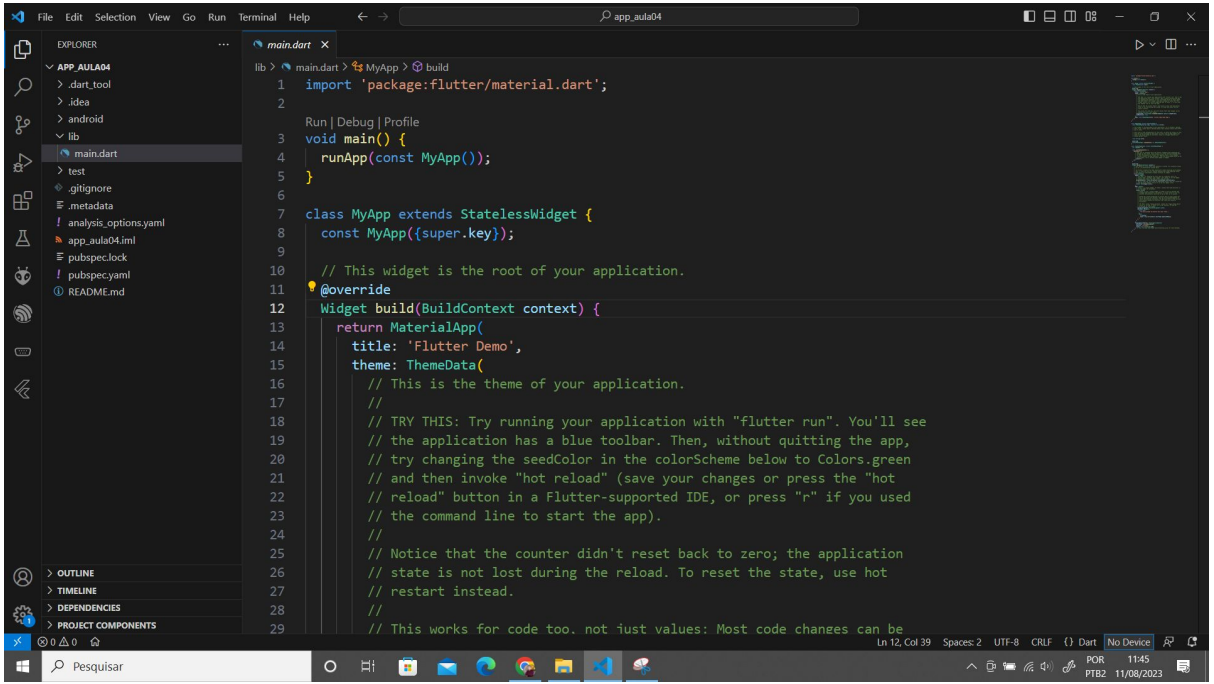
Criando projeto no VSCODE

Após realizar as configurações, apertar Esc e enter
O projeto Flutter será criado Marcar a opção Yes, I Trust the authors conforme a Figura.



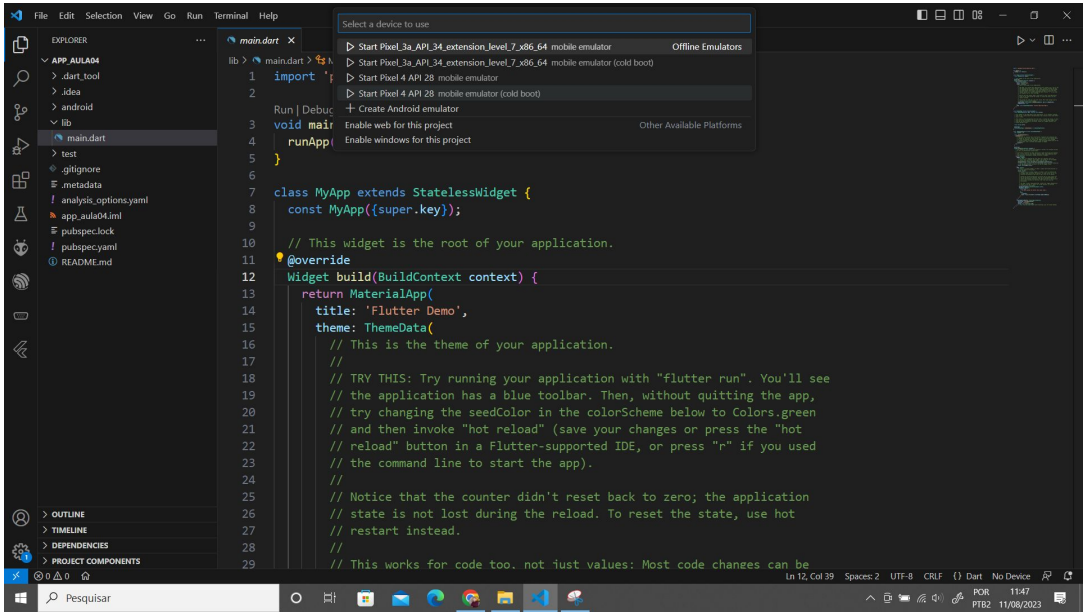
Criando projeto no VSCODE

Para executar o aplicativo no emulador o primeiro passo é clicar em iremos escolher o device para emular nosso telefone, clicar em No Device e selecionar o emulador desejado conforme a Figura.



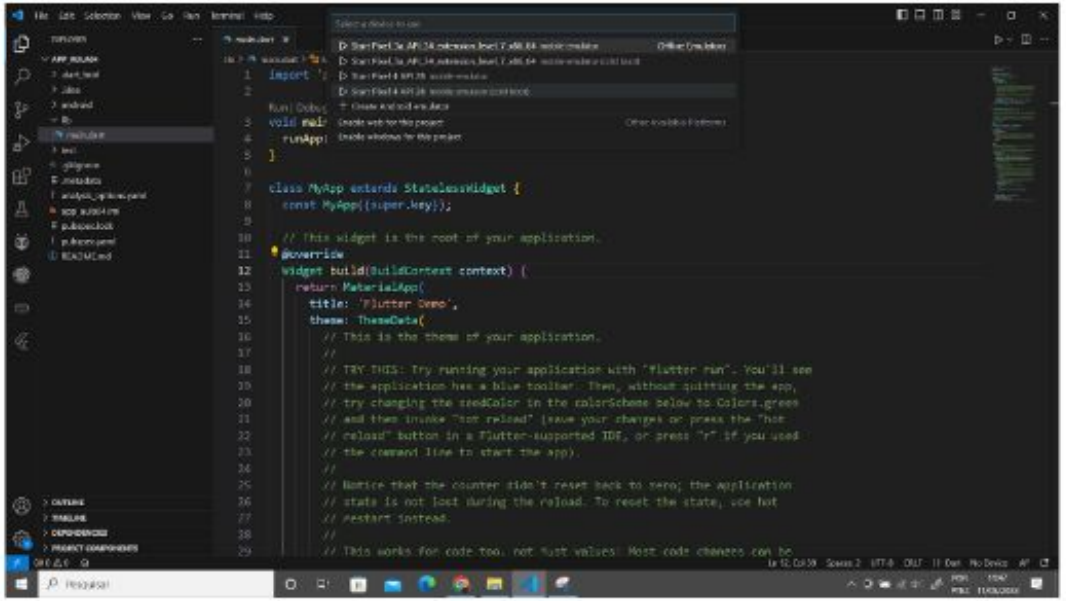
Criando projeto no VSCODE

Após clicar em No Device no editor VSCode irá aparecer para escolher o emulador para executar o aplicativo conforme a Figura, caso não apareça nenhum emulador, então é necessário criá-lo.



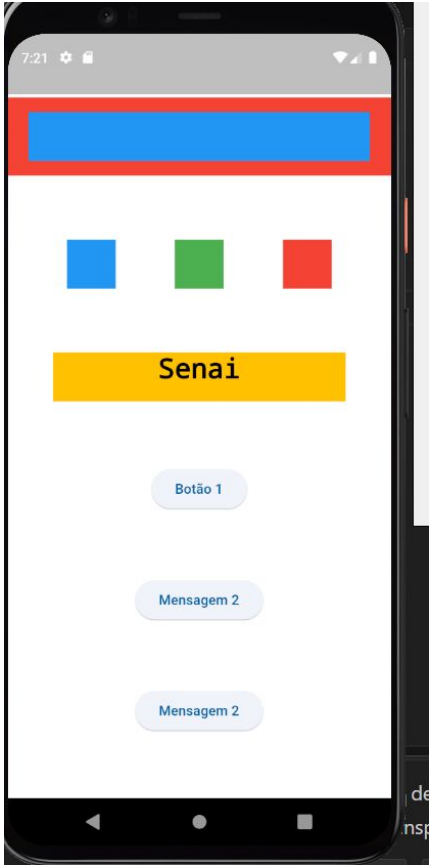
Criando projeto no VSCODE

Na Figura, após selecionar o device ele irá aparecer na tela



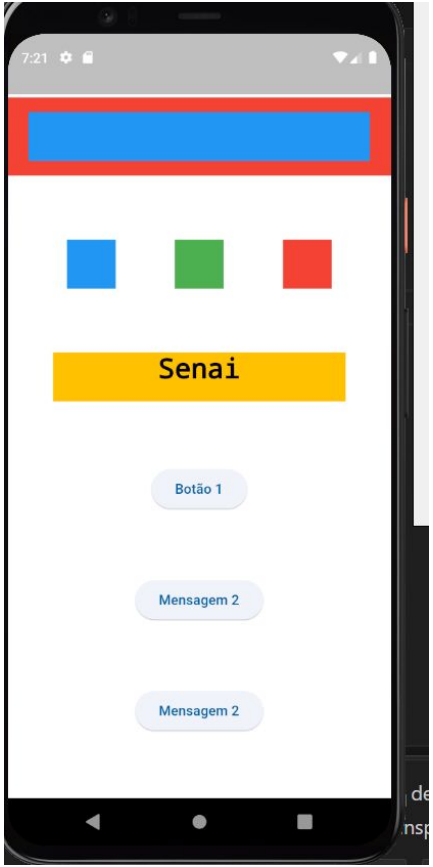
Aplicativo

```
1. /*
2. Importa biblioteca material que permite criar a interface do aplicativo com os widgets do Flutter
3. Column, Row, Elevated Button
4. */
5. import 'package:flutter/material.dart';
6. /*
7. A variável cont é usada como um contador global para ser incrementada ou decrementada nas funções.
8.
9. Funções:
10.
11. _msg() → Incrementa o valor de cont e imprime mensagens no console.
12. _msg2() → Decrementa o valor de cont e imprime mensagens no console.
13.
14. */
15. int cont =0;
16. void _msg()
17. {
18.         cont = cont+1;
19.         print("Desenvolvimento Mobile 1");
20.         print("Contagem $cont");
21. }
```



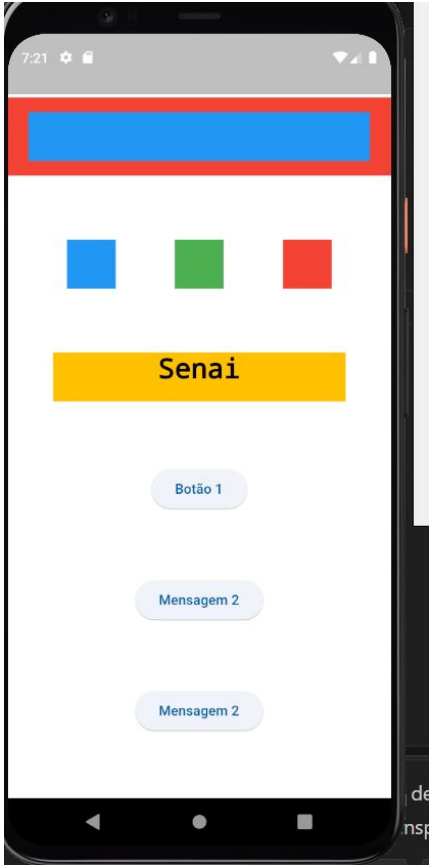
Aplicativo

```
22. void _msg2()
 23. {
 24.         cont = cont-1;
 25.         print("Contagem $cont");
 26.         print("Senai");
 27. }
 28.
 29. void main() {
 30.   runApp(const MyApp());
 31.    // remove a faixa debug do app
 32. }

34. Classe MyApp
 35. A classe MyApp é a raiz da aplicação Flutter e estende StatelessWidget.
 36.
 37. */
 38. /*
 39. Configuração do MaterialApp:
 40.
 41. debugShowCheckedModeBanner: false: Remove a faixa "debug" do aplicativo.
 42. Define um tema com Material 3 e cores baseadas em um "seed color".
 43.
 44. Widget Principal (Container)
 45. A tela principal usa um Container branco que organiza os elementos internos usando um
Column.
 46.
 47. Estrutura Geral dos Widgets:
 48. Stack:
 49.
```



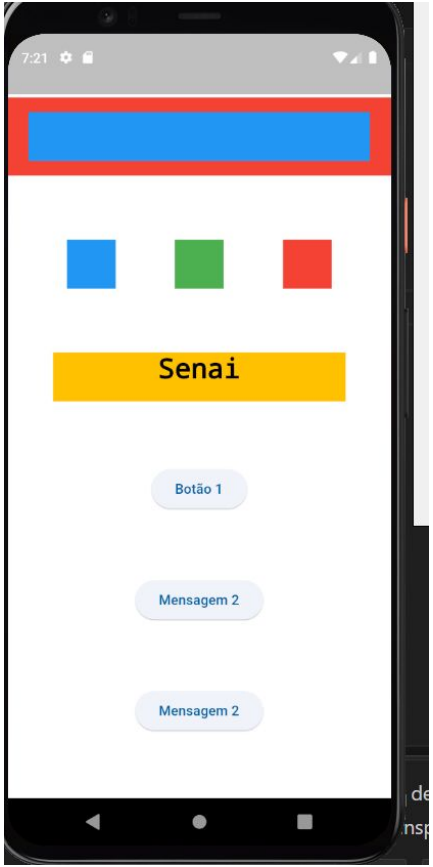
Aplicativo

```
50. Um Stack é usado para sobrepor dois contêineres.
 51. O Container vermelho é o fundo (mais amplo), e o azul é sobreposto no centro
 52.
 53. Row:
 54.
 55. Um Row alinha três quadrados (contêineres) de cores diferentes horizontalmente e com
espaçamento uniforme.
 56.
 57. Text:
 58.
 59. O texto "Senai" é centralizado dentro de um Container amarelo com tamanho fixo.
 60.
 61. Botões:
 62.
 63. Existem três botões criados com ElevatedButton:
 64. O primeiro botão imprime uma mensagem fixa.
 65. O segundo botão chama a função _msg para incrementar o contador.
 66. O terceiro botão chama a função _msg2 para decrementar o contador.
 67. dart
 68.
 69. */
```



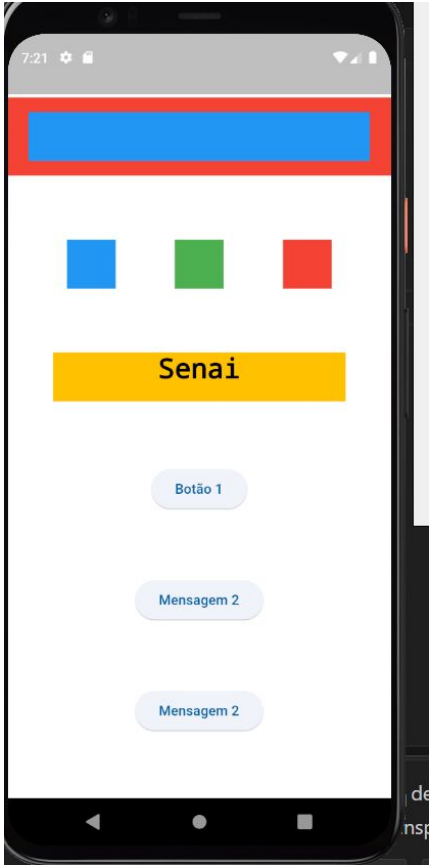
Aplicativo

```
70. class MyApp extends StatelessWidget {
 71.   const MyApp({super.key});
 72.
 73.   // This widget is the root of your application.
 74.   @override
 75.   Widget build(BuildContext context) {
 76.     return MaterialApp(
 77.       debugShowCheckedModeBanner: false, // Remove a faixa debug do app
 78.       title: 'Flutter Demo',
 79.       theme: ThemeData(
 80.                colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue),
 81.         useMaterial3: true,
 82.       ),
 83.       home: Container(
 84.         color:Colors.white,
 85.         child:Column(
 86.          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
 87.          children: [
 88.           Stack(
 89.             alignment: AlignmentDirectional.center,
 90.             children: [
 91.               Container(color:Colors.red,width:400,height:80),
 92.               Container(color:Colors.blue,width:350,height:50,),
 93.
 94.             ],
 95.           ),
```



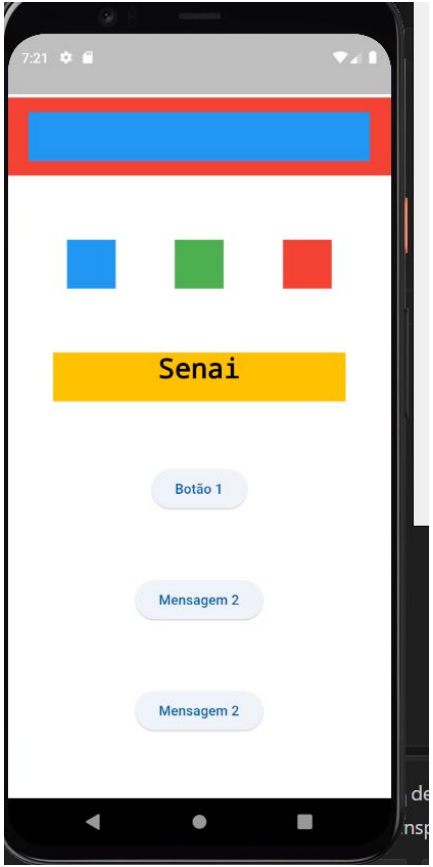
Aplicativo

```
96. Row(
 97.             mainAxisAlignment: MainAxisAlignment.spaceEvenly,
 98.             crossAxisAlignment: CrossAxisAlignment.end,
 99.             children: [
100.              Container(color:Colors.blue,height:50, width:50),
101.              Container(color:Colors.green,height:50, width:50),
102.              Container(color:Colors.red,height:50, width:50),
103.             ],
104.           ),
105.           Container(color:Colors.amber,height:50,width:300,
106.           child:Text("Senai",style:TextStyle(color:Colors.black,fontSize:28,
107.           decoration:TextDecoration.none),textAlign: TextAlign.center,),
108.         ),
109. ElevatedButton(
110.           onPressed: (){
111.             print("Você apertou o botao1");
112.           },
113.           child: Text("Botão 1"),),
```



Aplicativo

```
114.           ElevatedButton(
115.             onPressed: _msg,
116.             child: Text("Mensagem 2")),
117.
118.             ElevatedButton(
119.             onPressed: _msg2,
120.             child: Text("Mensagem 2")),
121.         ],
122.         ),
123.       ),
124.
125.     );
126.   }
127.   }
128.
129.
130.
131.
132.
133.
```

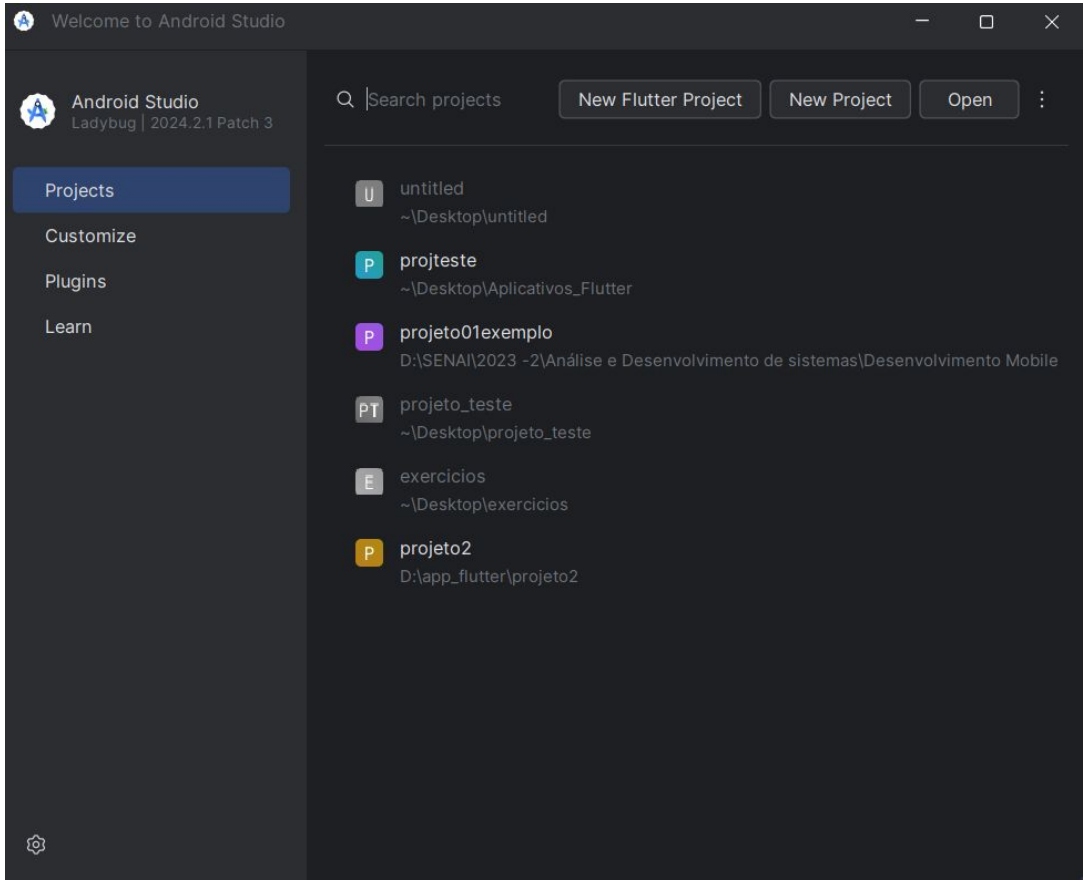


Criando emulador Android

Uma forma muito eficiente e rápida de testar os aplicativos desenvolvidos é utilizando o emulador do Android Studio, para isso é necessário criá-lo no Android Studio conforme os passos a seguir :

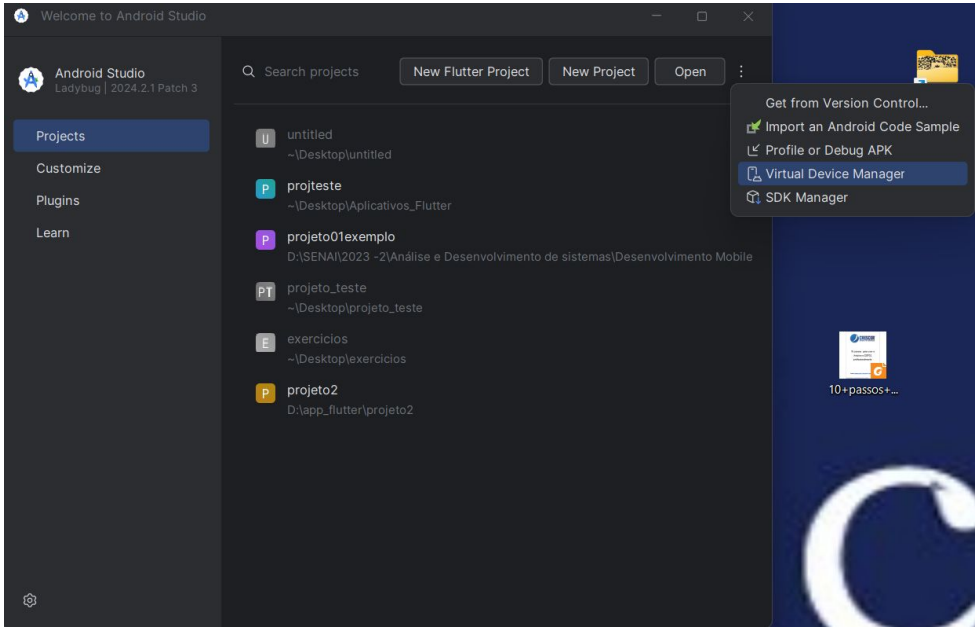
1º Abrir o Android Studio conforme a Figura 54 e clicar nos três pontos ao lado do botão Open

Criando emulador Android



Criando emulador Android

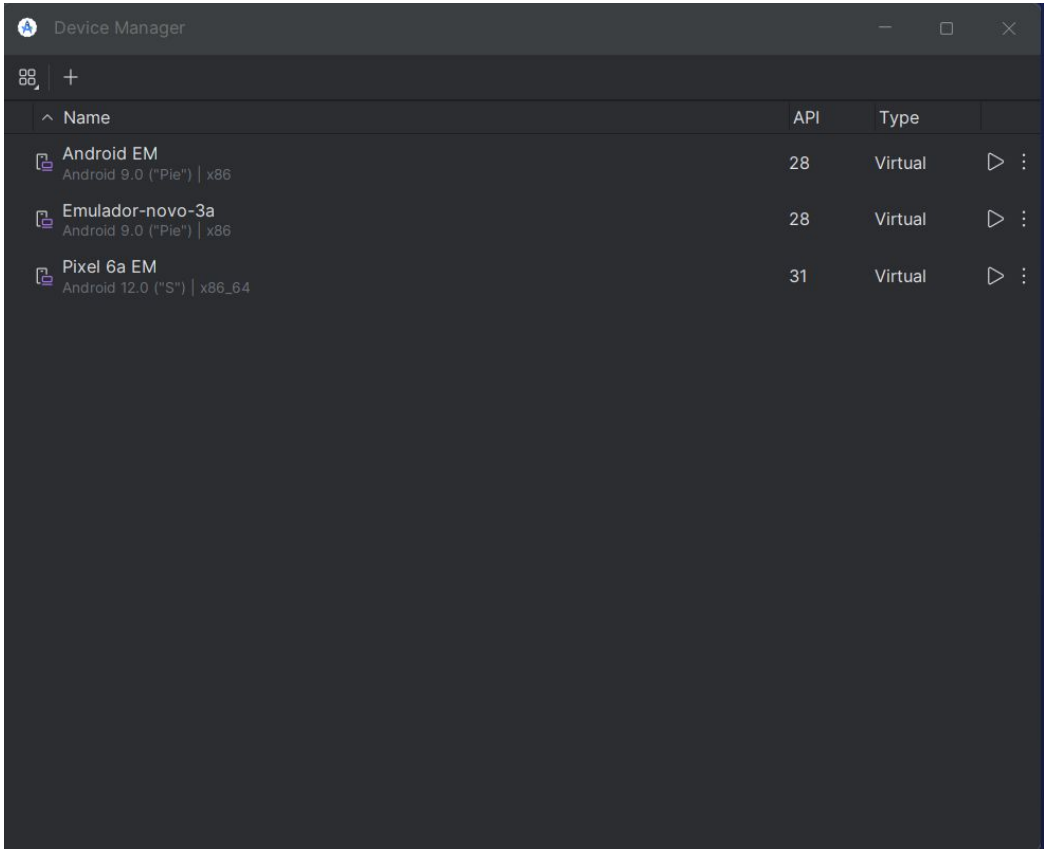
Na Figura após clicar nos três pontos ao lado do botão open, clicamos em Virtual Device Manager



Criando emulador Android

Após clicar em Virtual Device Manager, a tela com os emuladores criados irá aparecer conforme a Figura.

Criando emulador Android

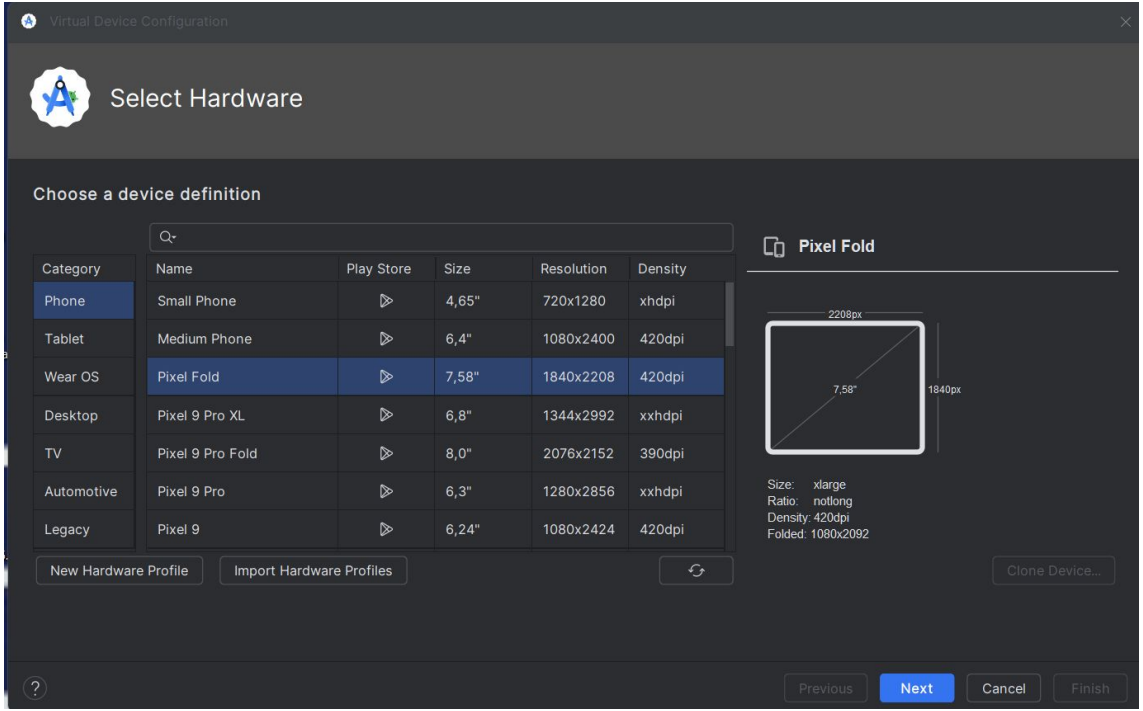


Criando emulador Android

Para criar um novo emulador é necessário clicar no ícone +

Após clicar no ícone + para criar um novo emulador irá aparecer a lista dos dispositivos emuladores disponíveis para serem escolhidos conforme a Figura

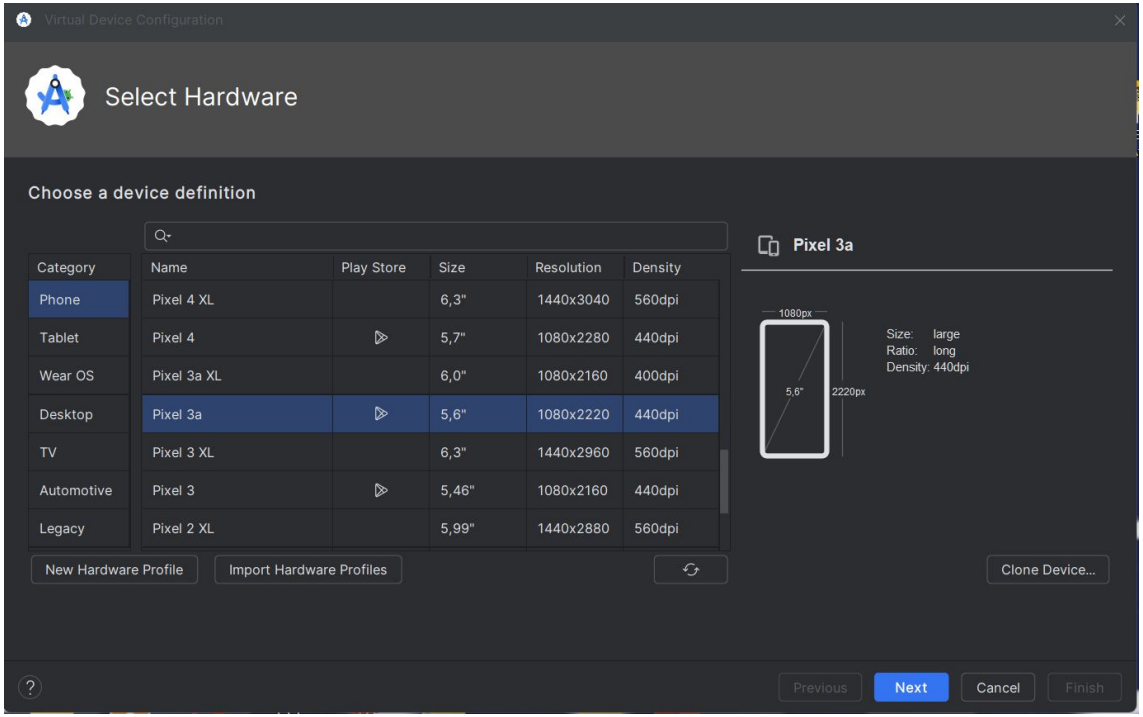
Criando emulador Android



Criando emulador Android

Iremos escolher o emulador Pixel 3a conforme a Figura, ou qualquer outro emulador que esteja com o ícone da Play Store ativado, para caso nós desejemos hospedar o aplicativo na Play Store.

Criando emulador Android

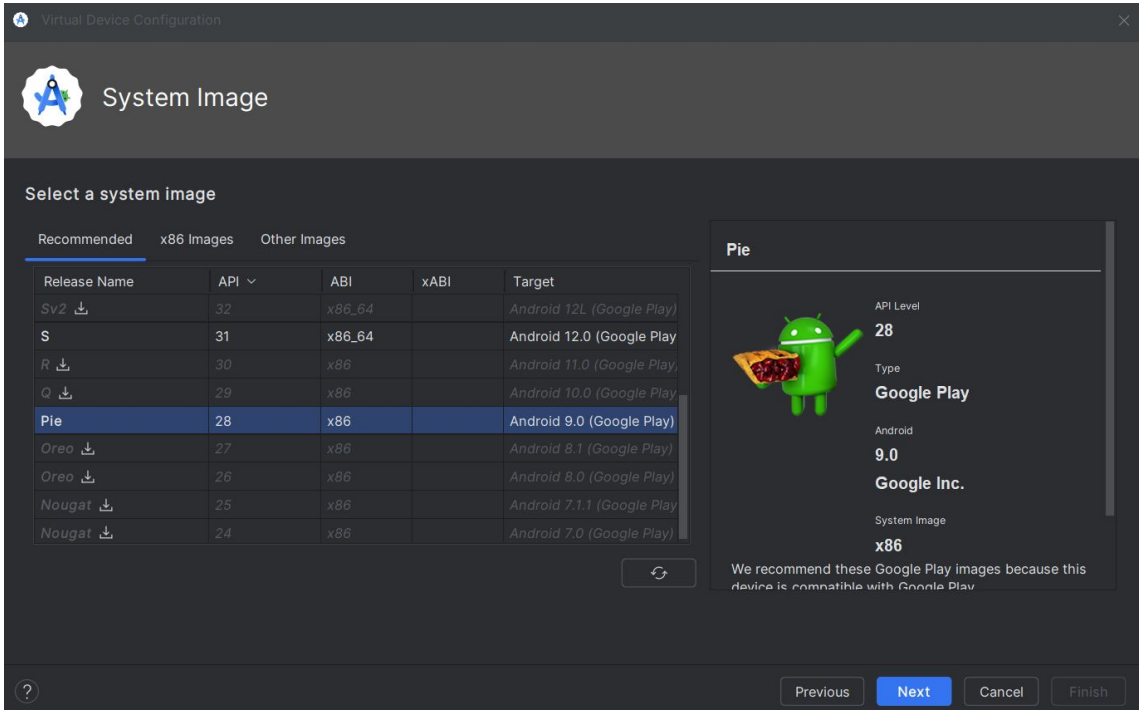


Criando emulador Android

Com o emulador Pixel 3a escolhido, o próximo passo será a escolha da versão do sistema operacional conforme a Figura.

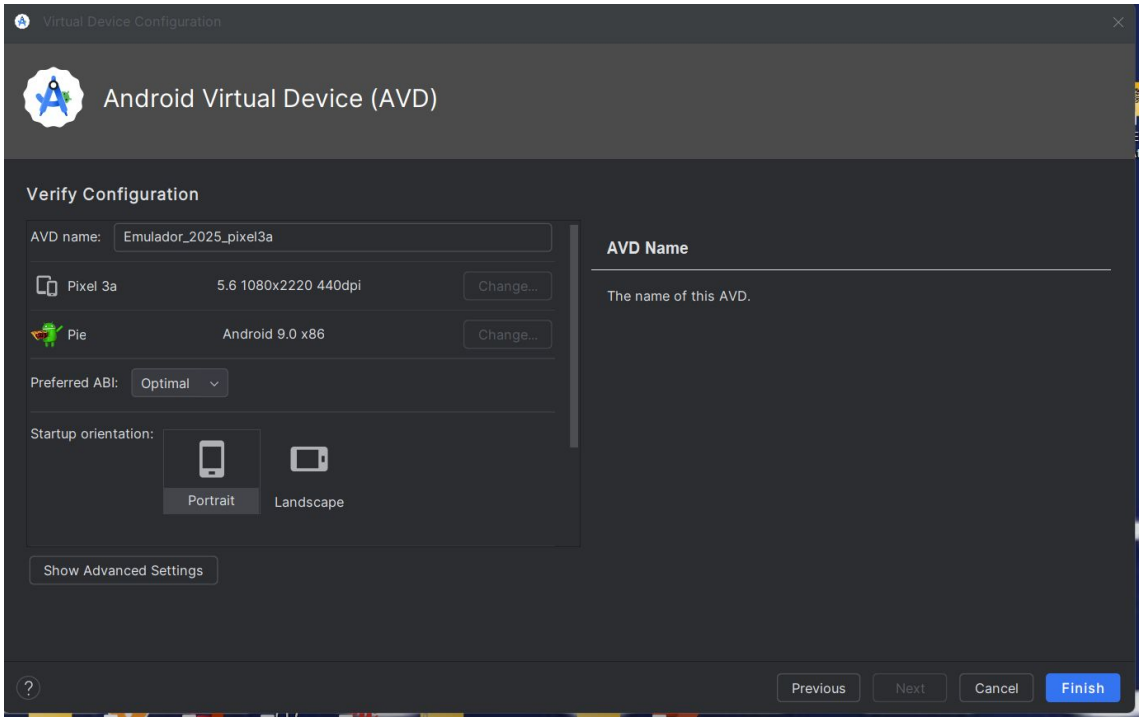
A versão escolhida será a versão Pie por ser mais leve para ser executada em nosso dispositivo.

Criando emulador Android



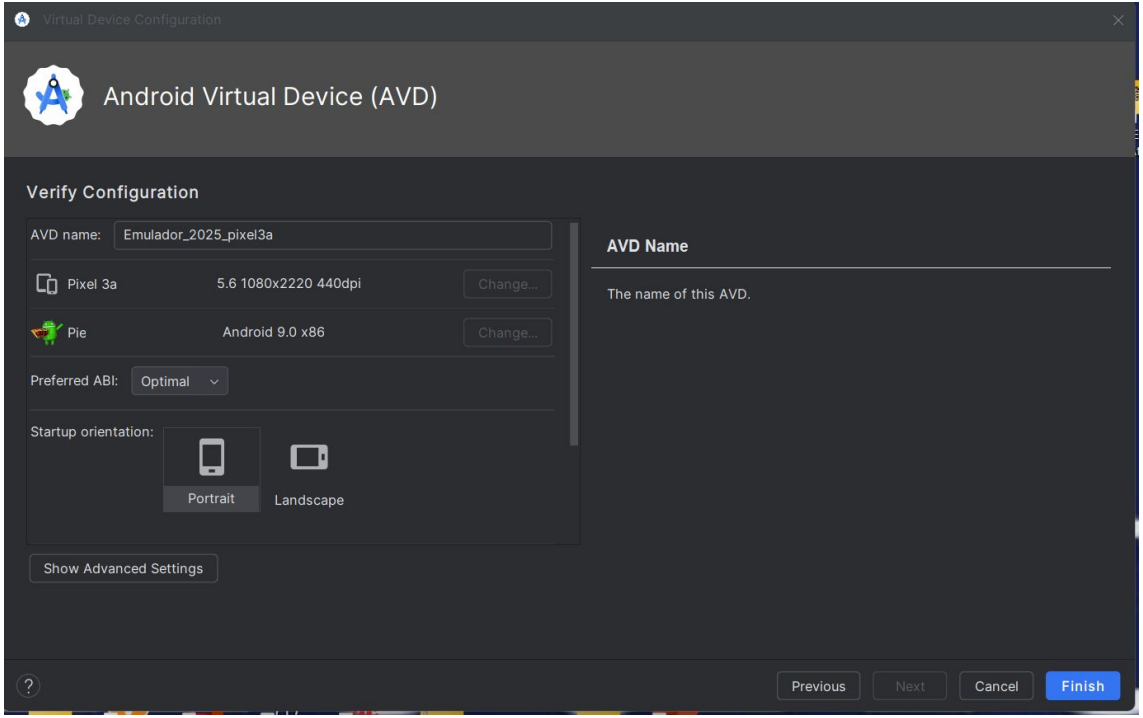
Criando emulador Android

Clicar em Next e nomear o emulador conforme a Figura



Criando emulador Android

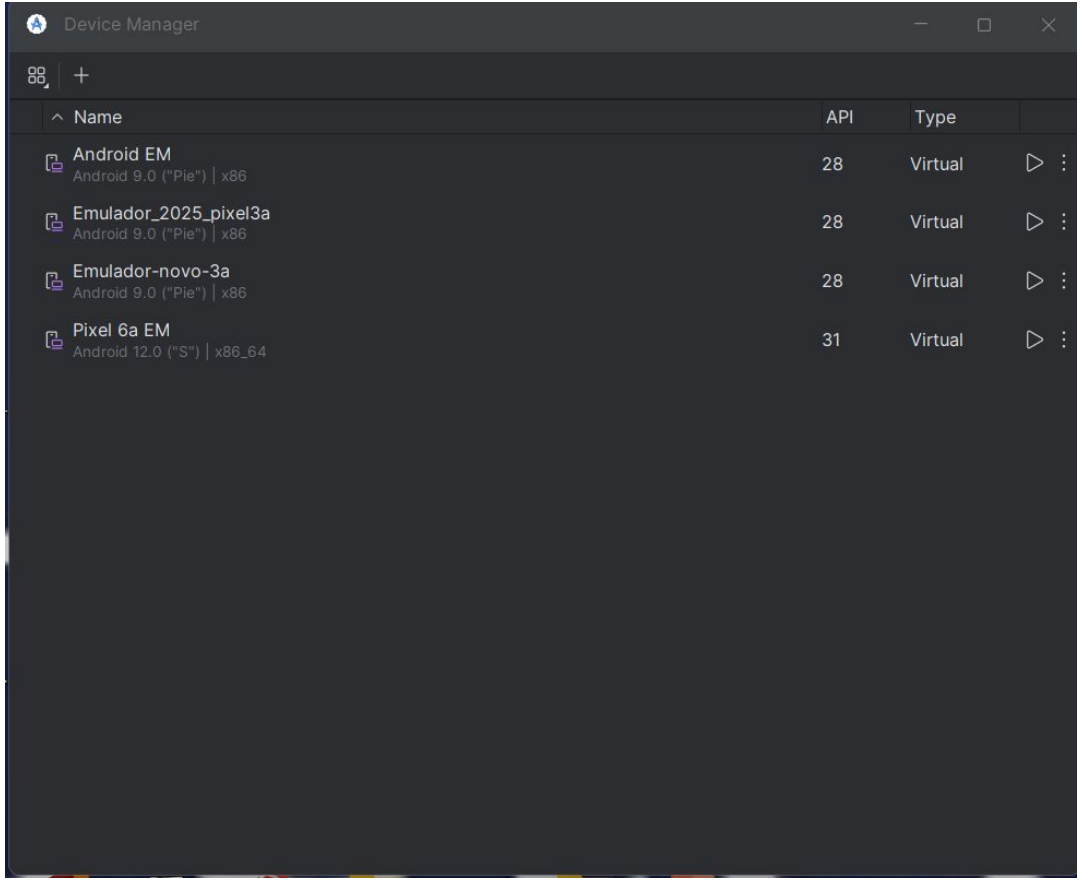
Após atribuir o nome Emulador_2025_pixel3a ao emulador clicar em Finish e o emulador será criado.



Criando emulador Android

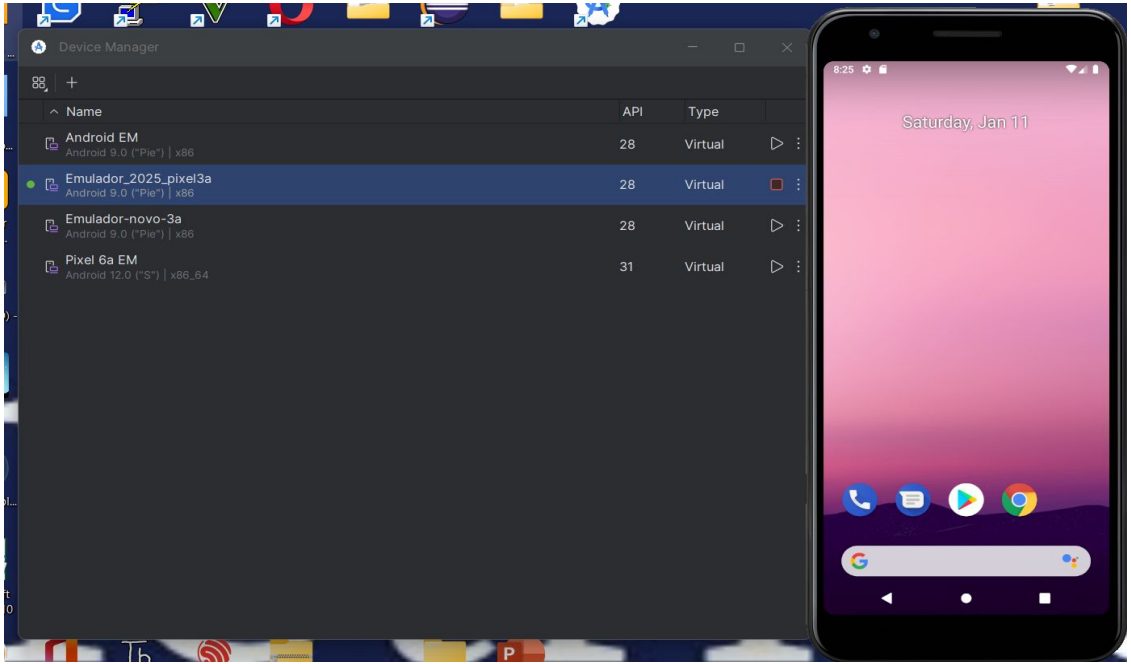
Na Figura é possível visualizar o emulador criado e é só clicar no play para o emulador ser inicializado abrindo o celular na Tela conforme a Figura

Criando emulador Android



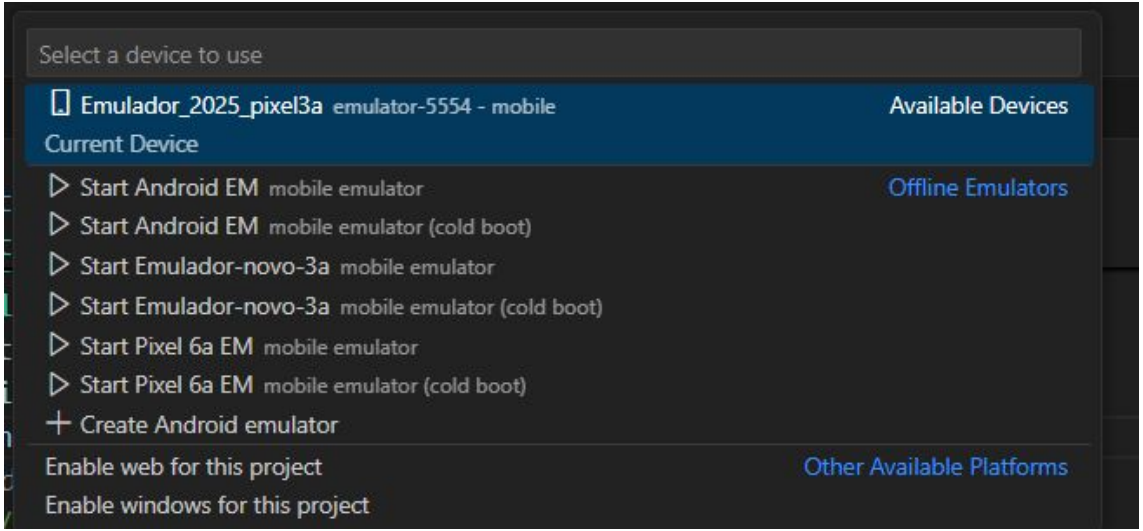
Criando emulador Android

Na Figura é possível ver o emulador sendo executado e agora é possível utilizá-lo para executar os aplicativos.



Criando emulador Android

Pronto, agora temos o emulador criado para executar os nossos aplicativos.



Obrigado!

Prof. Me Daniel Vieira

Email: danielvieira2006@gmail.com

Linkedin: Daniel Vieira

Instagram: Prof daniel.vieira95