

# Desenvolvimento para Dispositivos Móveis

## Criando o primeiro projeto em Flutter e executando no Emulador

Profº: Joseph Donald

Contatos:

☎ (83) 98228-8607

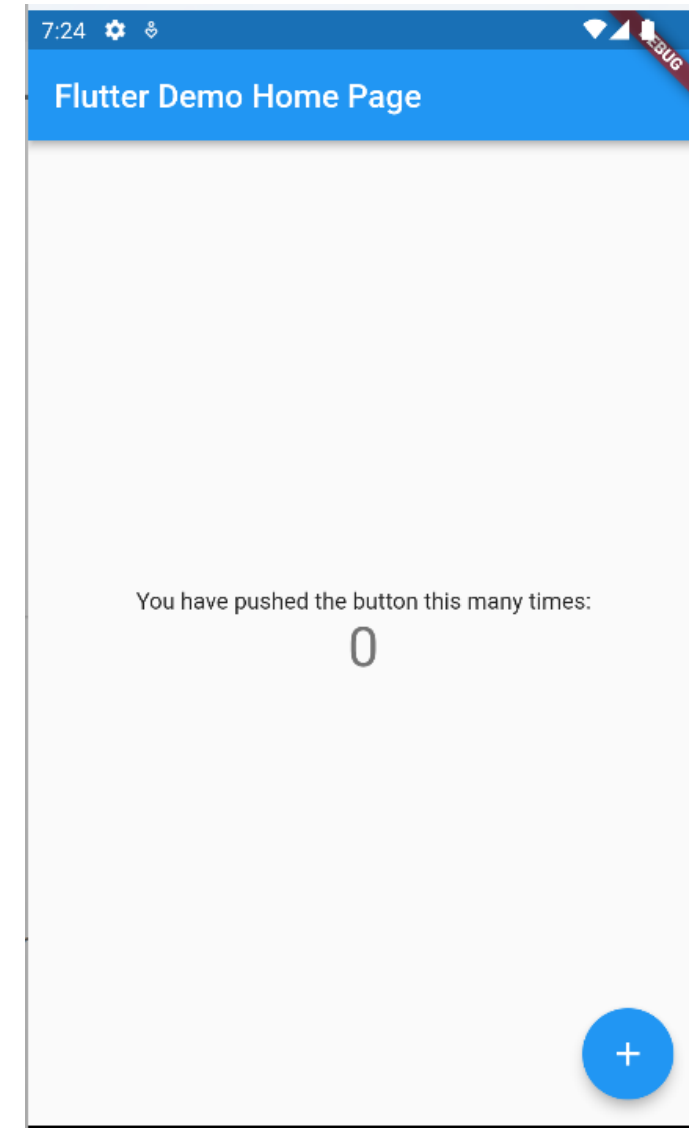
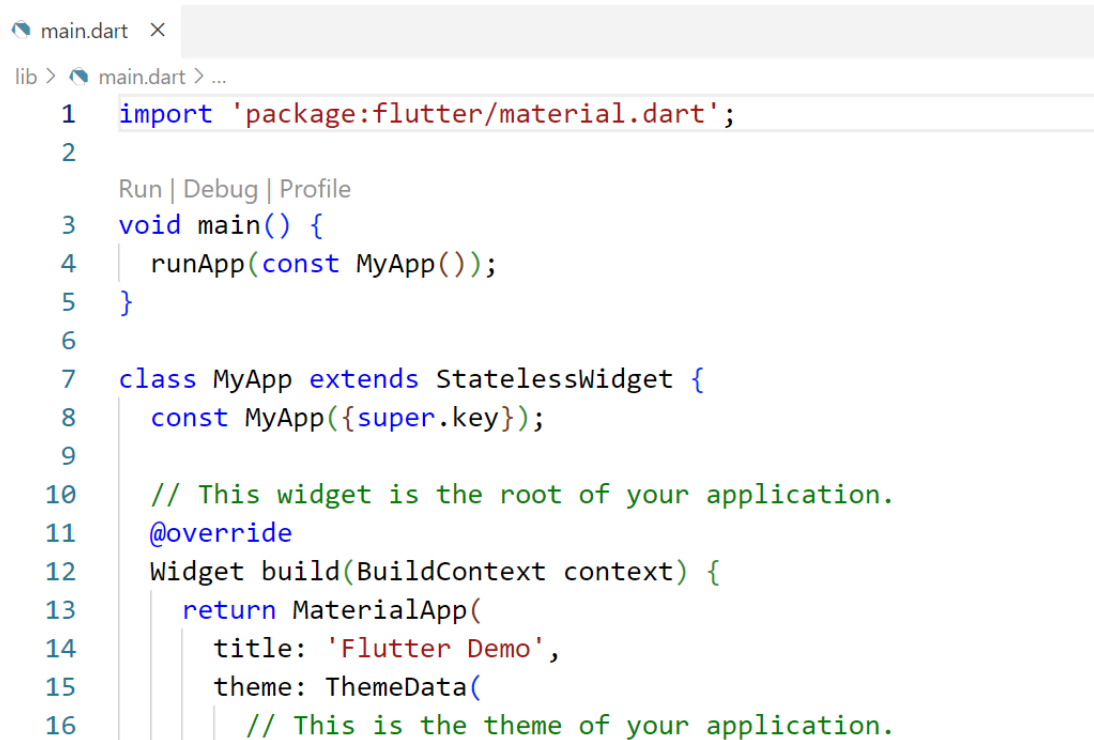
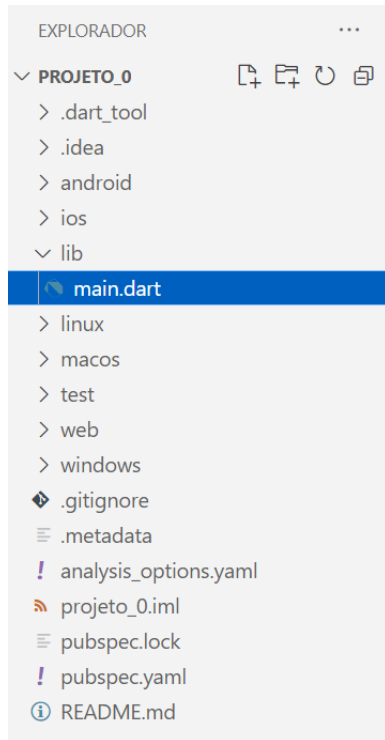
📷 @josephdonald

✉ 030106382@prof.uninassau.edu.br

*“Se você tem uma maçã e eu tenho outra; e nós trocamos as maçãs, então cada um terá sua maçã. Mas se você tem uma ideia e eu tenho outra, e nós as trocamos; então cada um terá duas ideias.”*

George Bernard Shaw

# Projeto 0



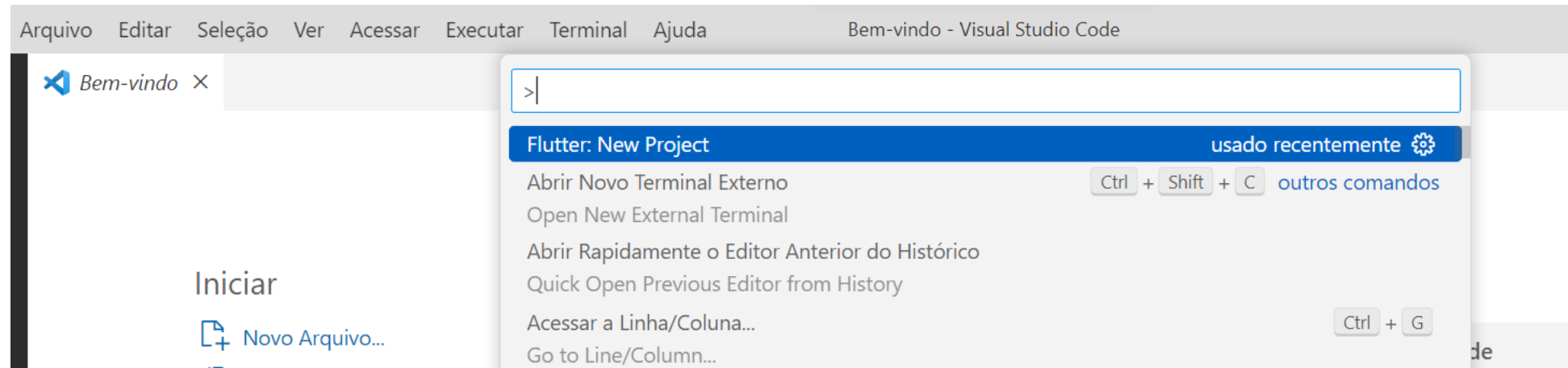
# Que formas posso criar um projeto?

- Você pode criar um projeto em flutter através do prompt do Windows ou através de uma IDE como, por exemplo, o VSCode.

Prompt →

```
cd ..\Desenvolvimento Mobile\projeto_teste; flutter create projeto_0
```

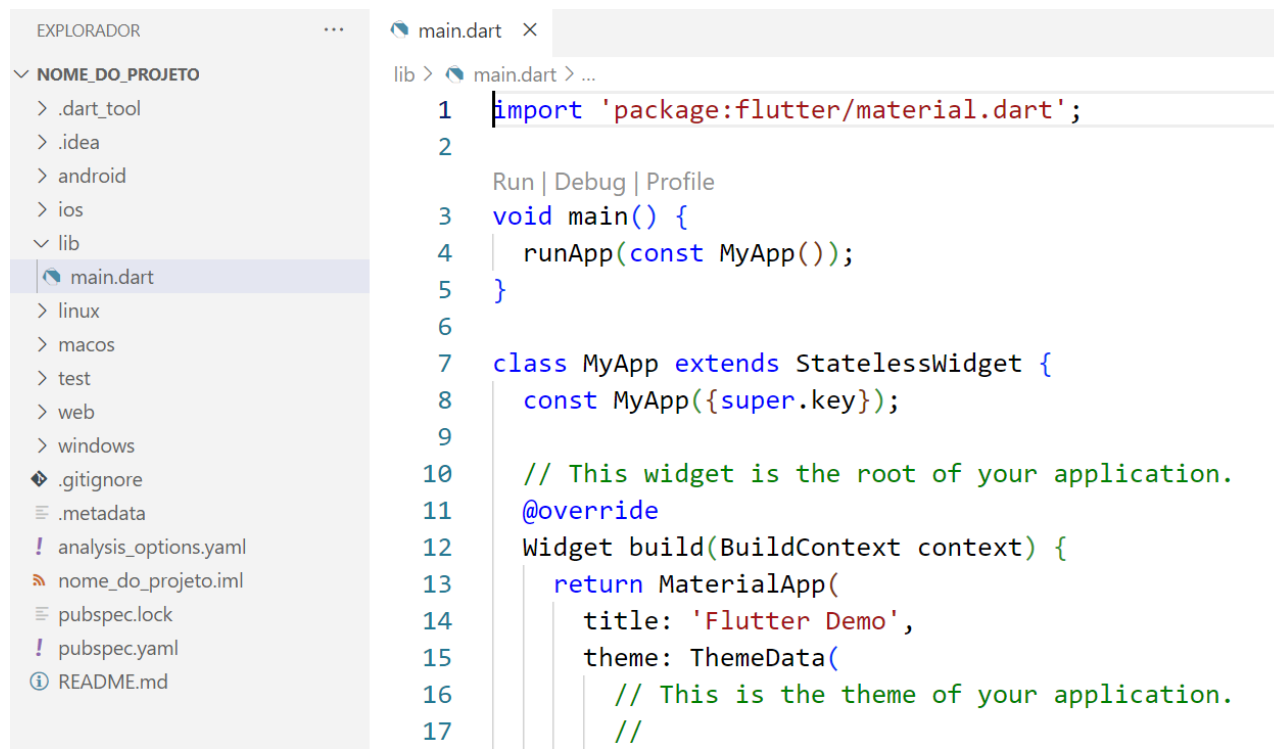
IDE →



- Ao criar um projeto pelo prompt, é possível utilizar-se de algumas customizações do projeto, onde você pode escolher, por exemplo, por: linguagem do iOS, linguagem do Android, templates ou até inserir a URL da empresa responsável o pelo projeto.
- O comando base para a criação dos projetos é: ***flutter create <nome-do-projeto>***
- Abaixo temos as opções para o comando flutter create:

```
Usage: flutter create <output directory>
-h, --help                Print this usage information.
--[no-]pub                Whether to run "flutter pub get" after the project has been created.
                           (defaults to on)
--[no-]offline            When "flutter pub get" is run by the create command, this indicates whether to run it in offline mode or
                           not. In offline mode, it will need to have all dependencies already available in the pub cache to succeed.
--[no-]overwrite          When performing operations, overwrite existing files.
--description             The description to use for your new Flutter project. This string ends up in the pubspec.yaml file.
                           (defaults to "A new Flutter project.")
--org                     The organization responsible for your new Flutter project, in reverse domain name notation. This string is
                           used in Java package names and as prefix in the iOS bundle identifier.
                           (defaults to "com.example")
--project-name            The project name for this new Flutter project. This must be a valid dart package name.
-i, --ios-language        The language to use for iOS-specific code, either Objective-C (legacy) or Swift (recommended).
                           [objc, swift (default)]
-a, --android-language    The language to use for Android-specific code, either Java (legacy) or Kotlin (recommended).
                           [java, kotlin (default)]
--platforms              The platforms supported by this project. Platform folders (e.g. android/) will be generated in the target
                           project. This argument only works when "--template" is set to app or plugin. When adding platforms to a
                           plugin project, the pubspec.yaml will be updated with the requested platform. Adding desktop platforms
                           requires the corresponding desktop config setting to be enabled.
                           [ios (default), android (default), windows (default), linux (default), macos (default), web (default)]
-t, --template=<type>    Specify the type of project to create.
```

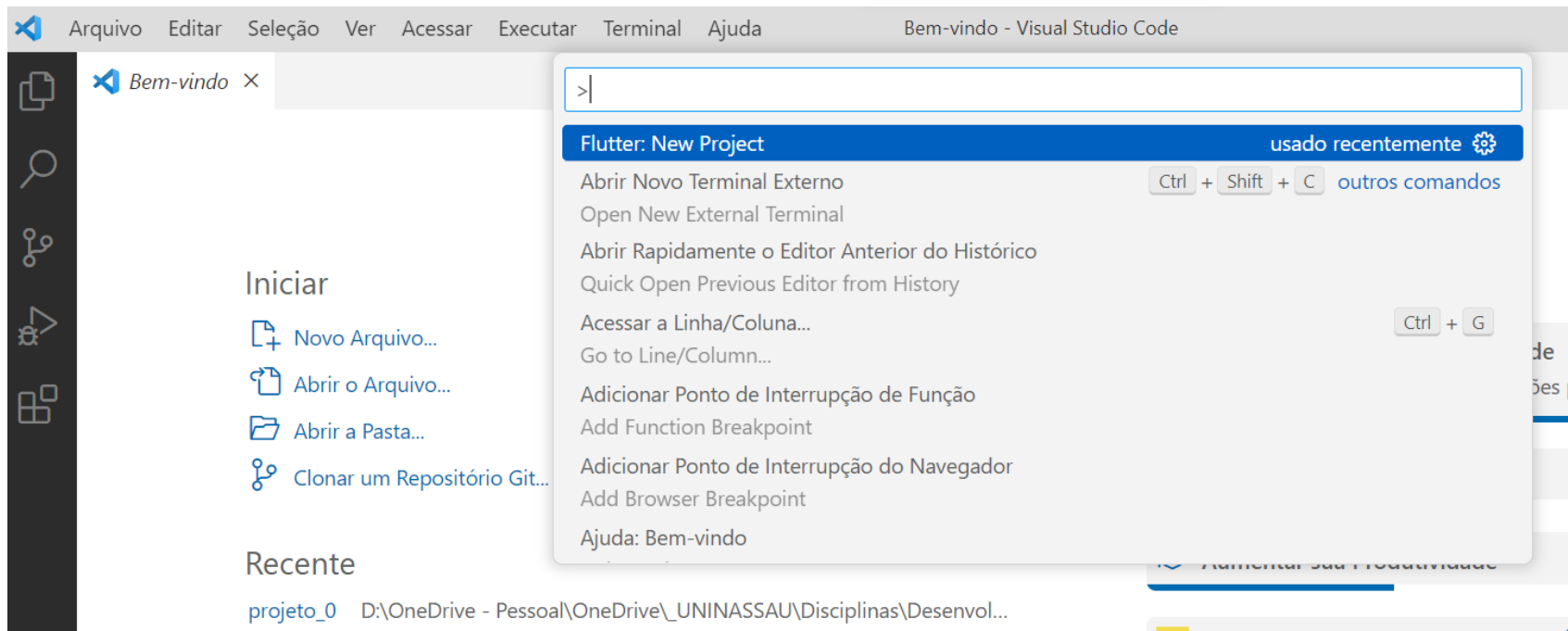
- Após a execução do comando, alguns vão ser criadas pastas e arquivos pertinentes ao projeto.
- Agora entre na pasta do projeto digitando, por exemplo: ***cd nome\_do\_projeto***
- E depois abra o projeto com o comando: **code .**



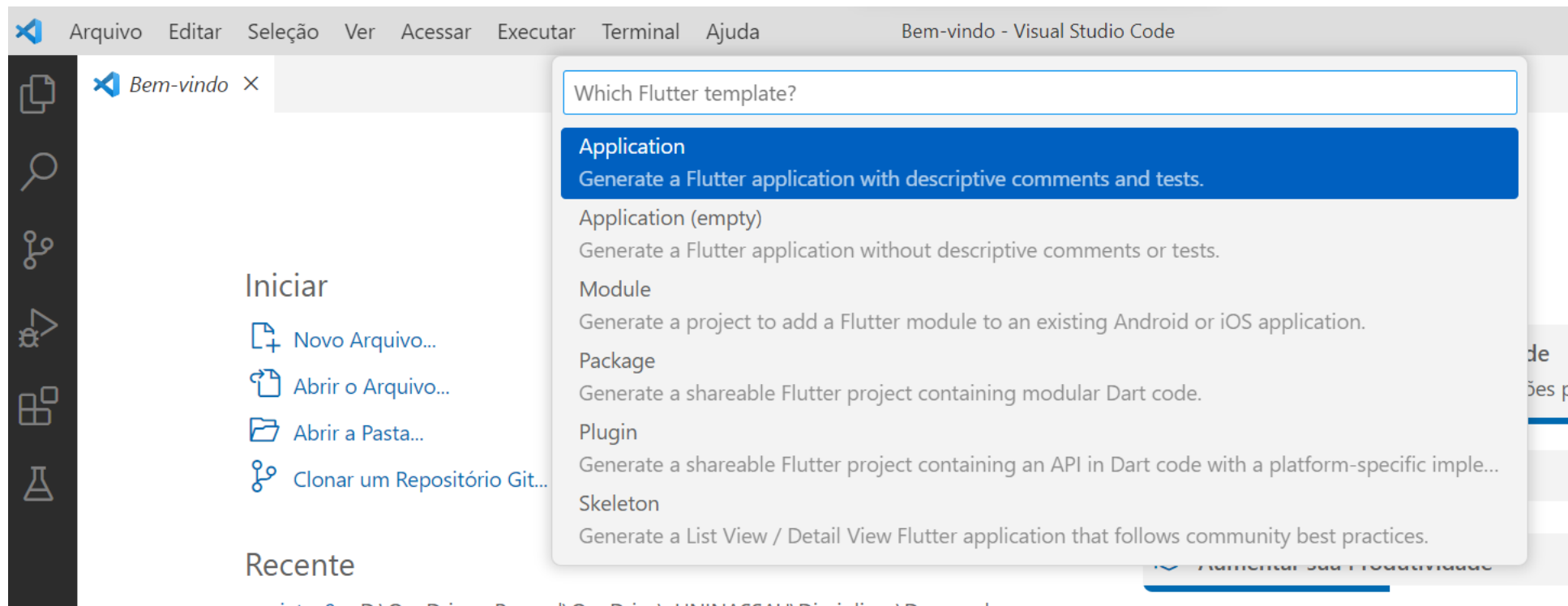
The screenshot shows an IDE interface. On the left, the 'EXPLORADOR' (Explorer) panel displays a project structure for 'NOME\_DO\_PROJETO'. The 'lib' directory is expanded, showing 'main.dart' selected. Other files and folders visible include '.dart\_tool', '.idea', 'android', 'ios', 'linux', 'macos', 'test', 'web', 'windows', '.gitignore', '.metadata', 'analysis\_options.yaml', 'nome\_do\_projeto.iml', 'pubspec.lock', 'pubspec.yaml', and 'README.md'. The main editor area shows the 'main.dart' file with the following code:

```
lib > main.dart > ...
1  import 'package:flutter/material.dart';
2
   Run | Debug | Profile
3  void main() {
4      runApp(const MyApp());
5  }
6
7  class MyApp extends StatelessWidget {
8      const MyApp({super.key});
9
10     // This widget is the root of your application.
11     @override
12     Widget build(BuildContext context) {
13         return MaterialApp(
14             title: 'Flutter Demo',
15             theme: ThemeData(
16                 // This is the theme of your application.
17                 //
```

- Para criar um projeto em Flutter pelo VSCode, siga os seguintes passos:
- ✓ Abra o VScode aperte o atalho: **CTRL+SHIFT+P**
- ✓ Irá aparecer uma tela semelhante a esta:

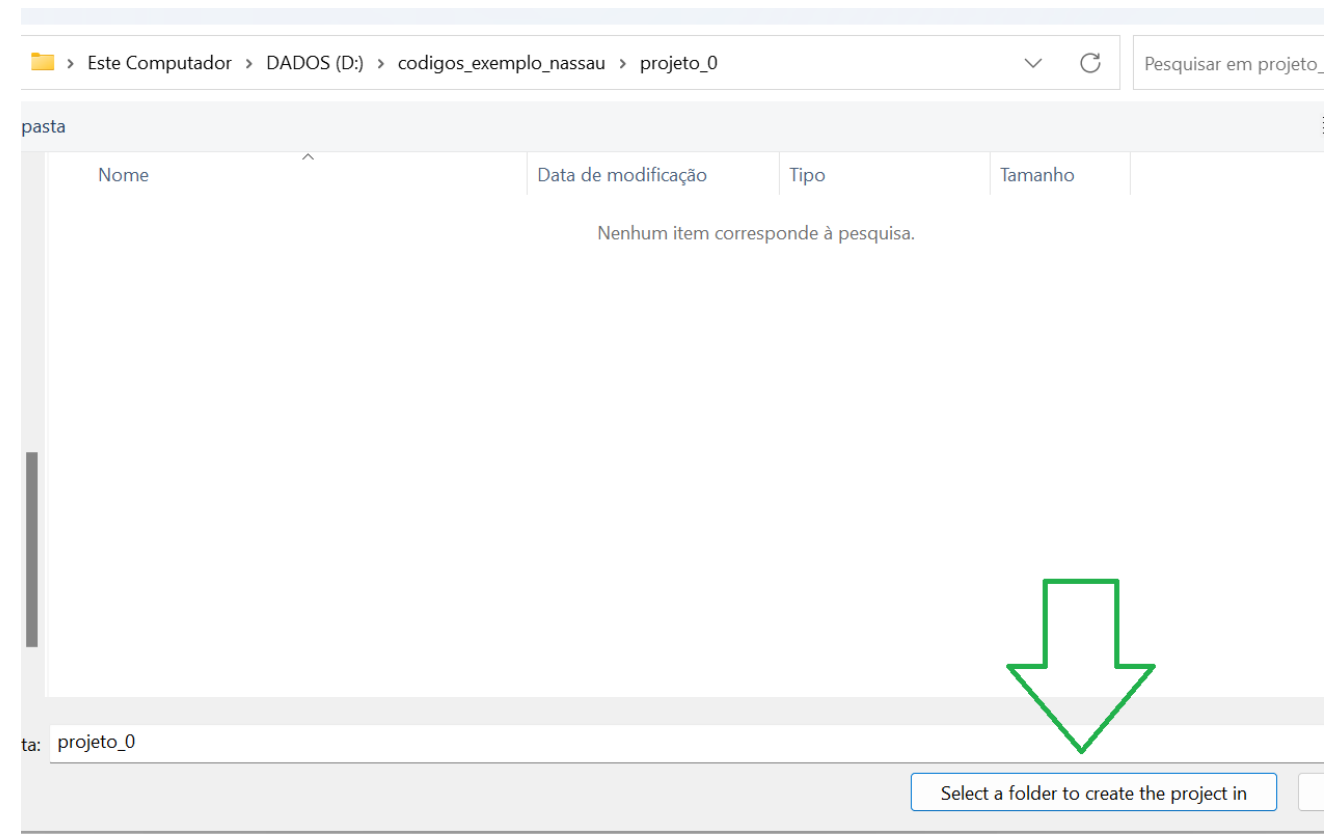
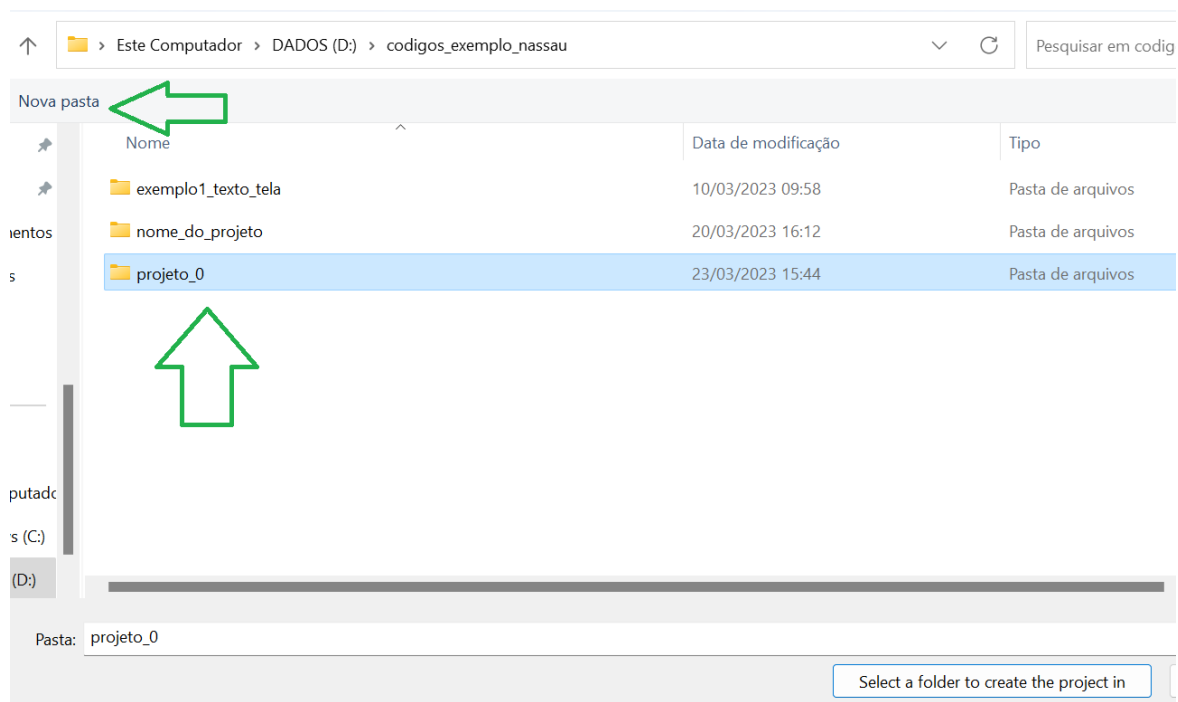


- Então clique em ***Flutter: New Project***
- Escolha a opção: ***Application***



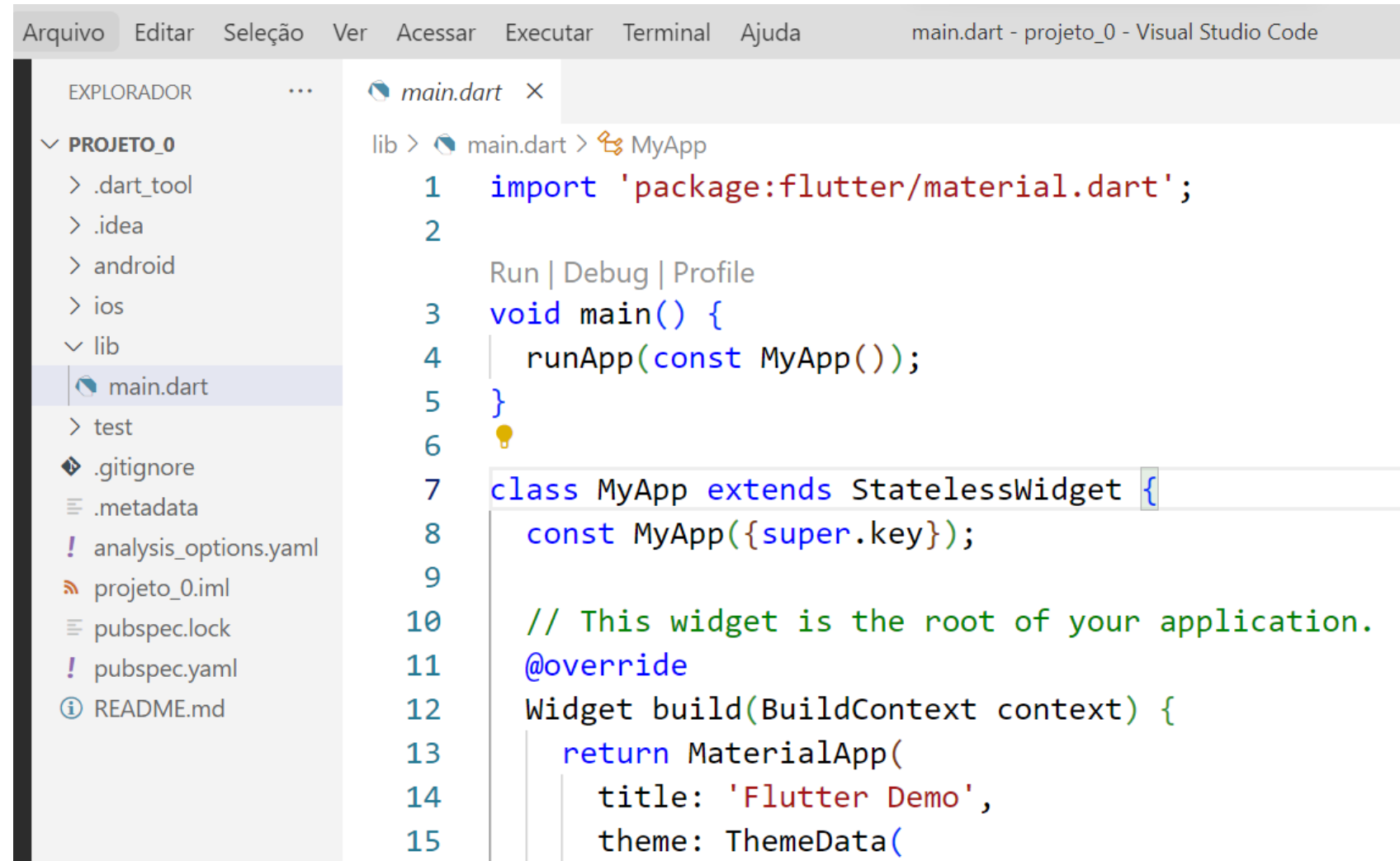
# Criando projeto pelo VSCode

- Crie uma pasta com o nome que desejar, por exemplo: “***projeto\_0***”
- ***Abra a pasta*** e depois clique em: “***Select a folder to create the project in***”





- Digite o nome do projeto, exemplo: “**projeto\_0**” e tecla **ENTER**
- Essa deverá ser a imagem do projeto criado.



The screenshot shows the Visual Studio Code interface with a project named 'projeto\_0'. The Explorer sidebar on the left shows the project structure, including folders like '.dart\_tool', '.idea', 'android', 'ios', 'lib', and 'test', and files like '.gitignore', '.metadata', 'analysis\_options.yaml', 'projeto\_0.iml', 'pubspec.lock', 'pubspec.yaml', and 'README.md'. The 'lib' folder is expanded, showing 'main.dart' selected. The main editor displays the code for 'main.dart', which includes an import statement for 'package:flutter/material.dart', a 'void main()' function that calls 'runApp(const MyApp())', and a 'class MyApp extends StatelessWidget' with an '@override' method 'build(BuildContext context)' that returns a 'MaterialApp' widget. The title bar of the editor shows 'main.dart - projeto\_0 - Visual Studio Code'.

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  main.dart - projeto_0 - Visual Studio Code

EXPLORADOR  ...
  PROJETO_0
    > .dart_tool
    > .idea
    > android
    > ios
    > lib
      main.dart
    > test
    .gitignore
    .metadata
    ! analysis_options.yaml
    projeto_0.iml
    pubspec.lock
    ! pubspec.yaml
    (i) README.md

lib > main.dart > MyApp
1  import 'package:flutter/material.dart';
2
   Run | Debug | Profile
3  void main() {
4    runApp(const MyApp());
5  }
6  ⚡
7  class MyApp extends StatelessWidget {
8    const MyApp({super.key});
9
10 // This widget is the root of your application.
11 @override
12 Widget build(BuildContext context) {
13   return MaterialApp(
14     title: 'Flutter Demo',
15     theme: ThemeData(
```

# Entendendo o projeto

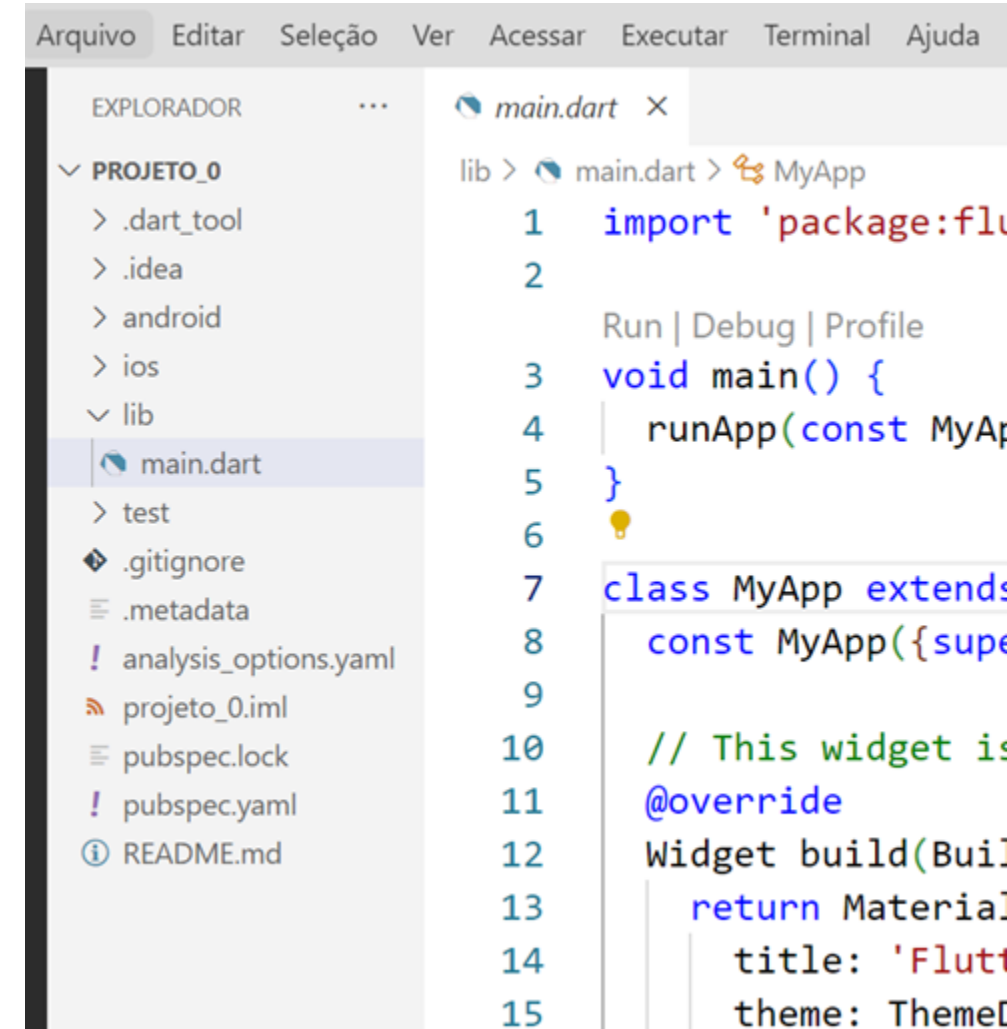
- Em um projeto básico do Flutter, existem vários diretórios com funções específicas. Aqui está uma breve explicação sobre a função de cada um deles:

1. **lib:** Esse diretório contém todo o código-fonte do aplicativo Flutter, incluindo arquivos Dart que contêm a lógica de negócios e a interface do usuário. Neste diretório o usuário pode criar pastas e subpastas para melhor organizar seu código conforme achar necessário.

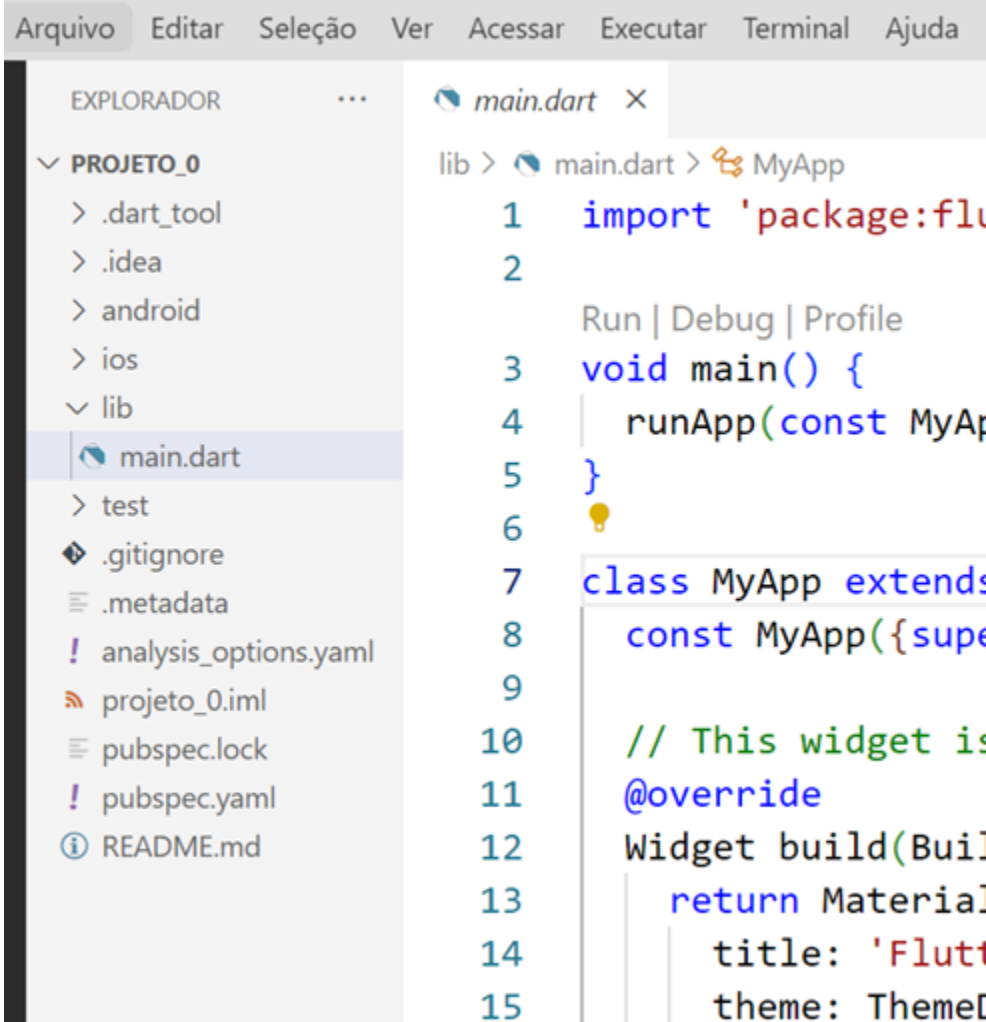
2. **test:** Esse diretório contém os arquivos de teste do aplicativo, incluindo testes unitários e de integração.

3. **android:** Este diretório contém o código-fonte específico do Android do aplicativo. Isso inclui o arquivo "build.gradle" que define as dependências e configurações do projeto Android.

4. **ios:** Este diretório contém o código-fonte específico do iOS do aplicativo. Isso inclui o arquivo "Podfile" que define as dependências e configurações do projeto iOS.



5. **web:** Este diretório contém o código-fonte específico do aplicativo Flutter para a web. Isso inclui arquivos HTML, CSS e JavaScript que são usados para exibir o aplicativo no navegador.
6. **assets:** Este diretório contém arquivos estáticos que o aplicativo usará, como imagens, fontes e arquivos de dados.
7. **build:** Este diretório é criado automaticamente pelo Flutter e contém arquivos de compilação do aplicativo.
8. **ios/Runner.xcworkspace:** Este é o arquivo principal do projeto iOS do aplicativo. Ele é aberto no Xcode e usado para compilar e implantar o aplicativo iOS.
9. **android/app/build.gradle:** Este arquivo define as configurações de compilação específicas do Android para o aplicativo. Ele inclui as dependências do aplicativo e outras configurações importantes.



The screenshot shows an IDE interface. On the left, the 'EXPLORADOR' (Explorer) panel displays the project structure for 'PROJETO\_0'. The 'lib' directory is expanded, showing 'main.dart' selected. On the right, the 'main.dart' file is open in the editor. The code shows the import of 'package:flutter' and the definition of the 'MyApp' class, which extends 'MaterialApp'. The 'main' function calls 'runApp' with 'MyApp' as an argument. The code is as follows:

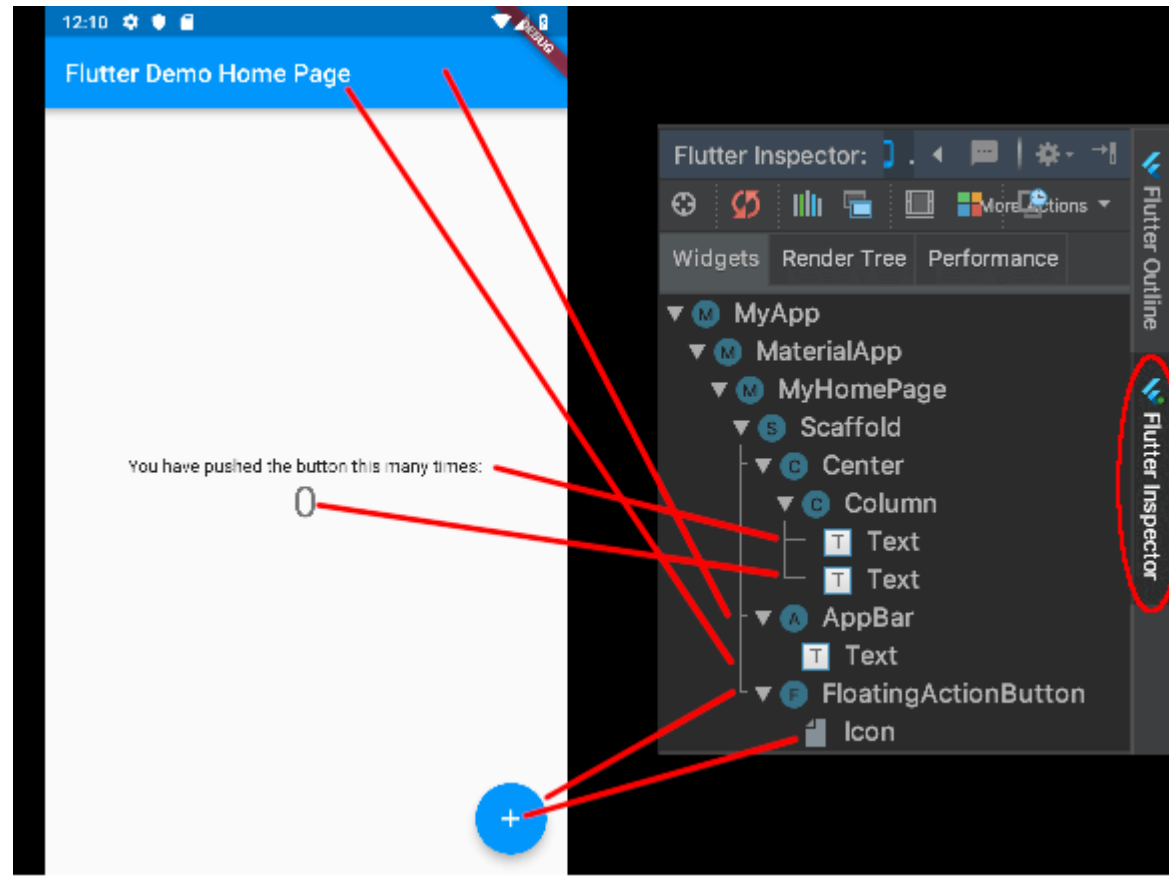
```
1 import 'package:flutter' as flutter;
2
3 void main() {
4   runApp(const MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   const MyApp({super.key});
9
10  // This widget is the root of the application.
11  @override
12  Widget build(BuildContext context) {
13    return MaterialApp(
14      title: 'Flutter Demo',
15      theme: ThemeData(
```



# Desafio 1

- Através do prompt crie um projeto em Flutter com o nome “**projeto\_desafio\_1**” habilitando para a Plataforma ANDROID. Depois abra o projeto através do prompt utilizando o VSCode.
- Crie um outro projeto Flutter utilizando o VSCode com o nome “**projeto\_desafio\_2**” habilitando para a plataforma WEB.

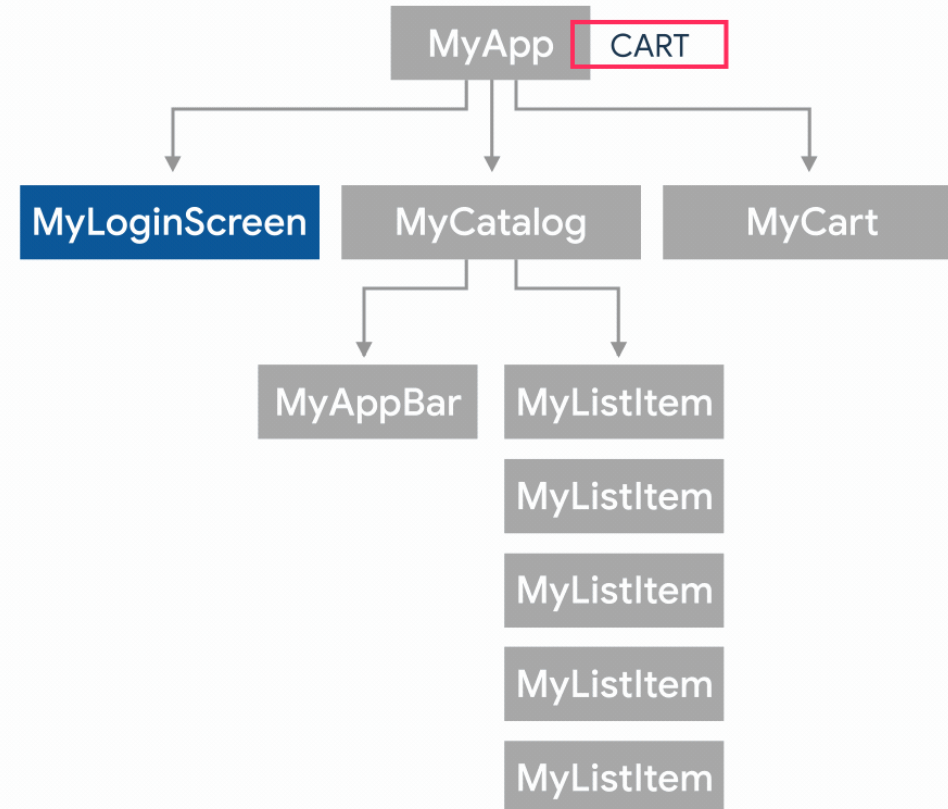
- **Definição:** Widgets em Flutter são os blocos básicos de construção da interface do usuário (UI) em aplicativos Flutter. Eles representam tudo o que é visível na tela, desde botões, caixas de texto e imagens, até layouts complexos e animações.



- A árvore de widgets em Flutter é uma estrutura hierárquica que define a composição da interface de usuário. Cada widget é um objeto com propriedades e comportamentos específicos.
- A árvore de widgets começa com um widget raiz, geralmente um **MaterialApp** ou **Scaffold**, que contém um ou mais widgets filhos.
- Cada widget pode ter filhos e pais, formando uma hierarquia de widgets, onde o pai define o layout e as propriedades gerais dos seus filhos.

# Exemplo de Árvore de Widgets

A mobile application interface mockup. It features a blue header with the word "Welcome" in white. Below the header is a white login form with two input fields labeled "Login" and "Password". At the bottom of the form is a blue button with the text "Enter".





# Exemplo de Árvore de Widgets

```

    main() → void
      MyApp
    MyApp
      MyApp({super.key})
      build(BuildContext context) → Widget
        MaterialApp
          MyHomePage
    MyHomePage
      MyHomePage({super.key, required this.title})
      title → String
      createState() → State<MyHomePage>
    _MyHomePageState
      _counter → int
      _incrementCounter() → void
      build(BuildContext context) → Widget
        Scaffold
          AppBar
            Text(widget.title)
          Center
            Column
              Text("You have pushe...s many times:")
              Text('$_counter')
          FloatingActionButton
            Icon(Icons.add)
  
```

```

    main () → void
      MyApp
    MyApp
      MyApp ({super.key})
      build (BuildContext context) → Widget
        MaterialApp
          MyHomePage
    MyHomePage
      MyHomePage ({super.key, required thi...
      title → String
      createState () → State<MyHomePage>
    _MyHomePageState
      _counter → int
      _incrementCounter () → void
      build (BuildContext context) → Widget
        Scaffold
          AppBar
            Text widget.title
          Center
            Column
              Text 'You have pushed the button...
              Text '$_counter'
          FloatingActionButton
            Icon Icons.add
  
```





# STATELESS e STATEFULL

## Stateless Widget

WIDGET

VS.

## Stateful Widget

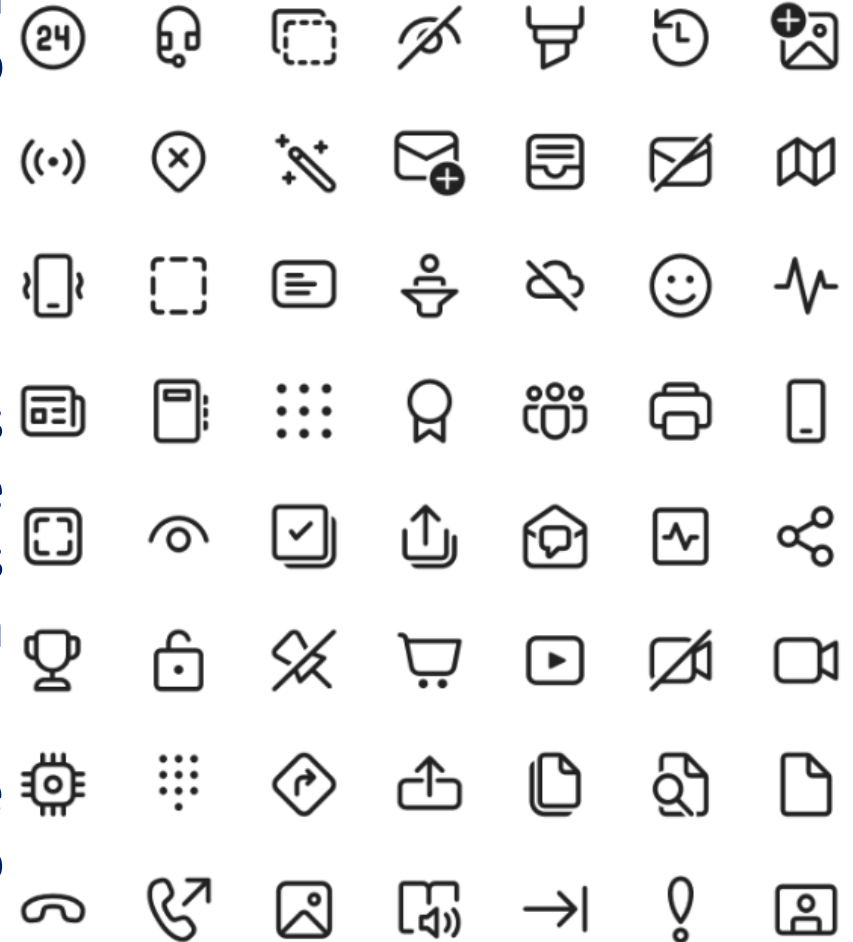
WIDGET

STATE

**Stateless** e **Stateful** são duas categorias de Widgets em Flutter, que diferem na sua capacidade de alterar o estado interno durante a execução do aplicativo.

## Stateless Widgets:

- São widgets que **não têm estado interno mutável**. Eles são construídos apenas uma vez, quando são criados, e exibem as informações recebidas por meio das propriedades que são passadas ao widget na construção.
- Uma vez que o Stateless Widget é criado, ele não pode ser alterado ou atualizado durante a execução do aplicativo.
- Exemplos comuns de Stateless Widgets são: Text, Icon, Image, etc.

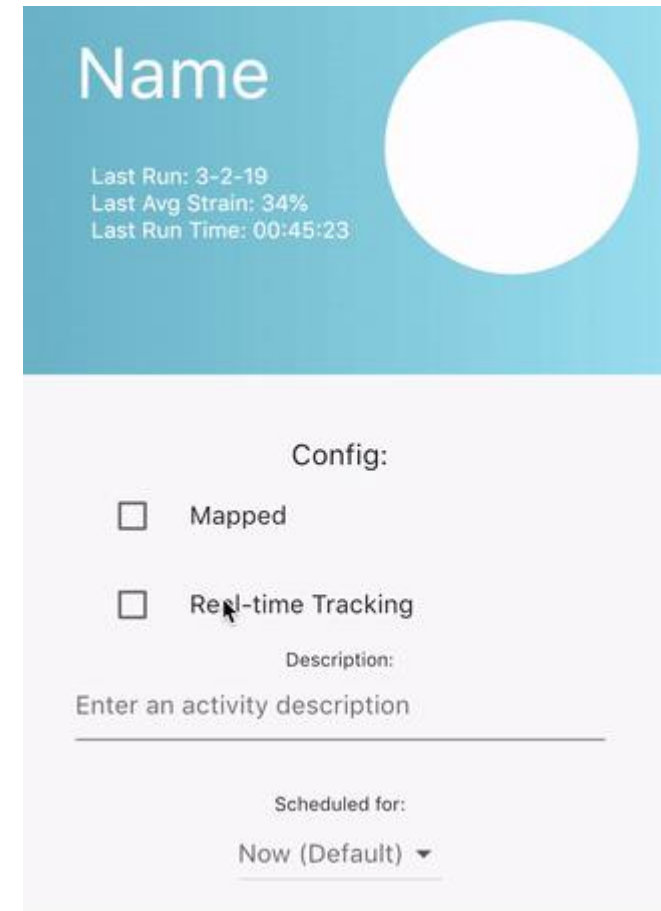


# STATELESS ou STATEFULL

## Stateful Widgets:

- São widgets que têm estado interno mutável. Eles podem ser construídos várias vezes durante a execução do aplicativo, com base em alterações de estado.
- Uma vez criado, um Stateful Widget pode ser atualizado em tempo real quando ocorrem mudanças de estado, e a interface do usuário é atualizada de acordo.
- Exemplos comuns de Stateful Widgets são: TextField, Checkbox, Slider, etc.

Em resumo, o Stateless Widget é estático e não pode ser alterado depois de ser construído, enquanto o Stateful Widget é dinâmico e pode ser atualizado em tempo real. A escolha entre um ou outro tipo de widget depende das necessidades e requisitos do aplicativo.



The screenshot shows a mobile application interface. At the top, there's a blue header with the word "Name" in white. Below it, on the left, are three lines of text: "Last Run: 3-2-19", "Last Avg Strain: 34%", and "Last Run Time: 00:45:23". To the right of this text is a large white circle. Below the header is a light gray section titled "Config:". It contains two checkboxes: "Mapped" and "Real-time Tracking", both of which are unchecked. Below these is a "Description:" label followed by a text input field with the placeholder text "Enter an activity description". At the bottom, there's a "Scheduled for:" label followed by a dropdown menu showing "Now (Default)".

- Em Flutter, existem dois tipos de widgets: **Stateless** e **Stateful**.
- Um widget **Stateless** é um widget que não mantém estado interno, ou seja, não é capaz de mudar a si mesmo ao longo do tempo. Esse tipo de widget é usado quando o conteúdo do widget é estático e não muda, como um ícone, um texto ou uma imagem.
- Um exemplo de um widget Stateless é o Icon. Ele é usado para exibir um ícone específico e não precisa de nenhum estado interno para funcionar. O Icon é criado uma única vez e exibido na tela sem precisar de atualizações.
- Já um widget Stateful é um widget que pode mudar seu estado interno ao longo do tempo, geralmente em resposta às interações do usuário ou a eventos específicos. Esse tipo de widget é usado quando o conteúdo do widget é dinâmico e pode mudar, como uma lista de tarefas ou um contador de cliques.
- Um exemplo de um widget Stateful é o TextField. Ele é usado para permitir que o usuário digite texto em um campo de entrada e seu estado interno é atualizado toda vez que o usuário digita ou apaga um caractere.

- Nos “projetos testes” que você criou. Analise o código e altere os seguintes dados:
1. Altere o título do APP para: “Contador de Cliques”
  2. Altere o texto central para: “Quantidade de vezes que você clicou:”
  3. Altere a cor da App Bar para verde
  4. Altere a cor do botão de cliques para amarelo