UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
CAMPUS DE PONTA PORÃ

PEDRO HENRIQUE PELIÇON RAMOS

EXPLORANDO O FLUTTER: da instalação à experiência de desenvolvimento

Ponta Porã - MS
2024

PEDRO HENRIQUE PELIÇON RAMOS

EXPLORANDO O FLUTTER: da instalação à experiência de desenvolvimento

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Leonardo Souza Silva

Ponta Porã - MS
2024

AGRADECIMENTOS

Gostaria de expressar minha gratidão a todas as pessoas e instituições que contribuíram para a realização deste trabalho. Esta jornada acadêmica foi desafiadora e recompensadora, e não teria sido possível sem o apoio e incentivo de muitas pessoas especiais.

Em primeiro lugar, desejo expressar minha gratidão à minha família, em especial aos meus pais, Angela e Claudiomar, ao meu padrasto Jeová e ao meu irmão, João Carlos, pelo constante encorajamento e apoio emocional. Eles foram fonte de inspiração para mim em todos os momentos.

Não posso deixar de agradecer aos amigos que estiveram ao meu lado durante essa jornada acadêmica. Suas palavras de incentivo e momentos de descontração foram essenciais para manter o equilíbrio entre o estudo e a vida social.

Por último, mas não menos importante, quero agradecer a todas as fontes de inspiração que encontrei ao longo dessa jornada acadêmica. Cada autor, pesquisador e indivíduo que contribuiu com seu conhecimento e experiência merece meu reconhecimento.

Este trabalho reflete o esforço coletivo de muitas pessoas e estou profundamente grato a todos que de alguma forma participaram dessa trajetória. O apoio de vocês foi fundamental para que esta monografia se tornasse uma realidade

Lista de Ilustrações

SUMÁRIO

1. INTRODUÇÃO

Com a popularização da Internet, o desenvolvimento de aplicações voltadas à web se tornou um importante recurso para as organizações, que encontram um meio acessível para oferta de informações e serviços à população. Em paralelo, os avanços tecnológicos contribuíram para que os telefones celulares e, posteriormente, os smartphones, se tornassem a principal ferramenta de conexão da população com a Internet e com os serviços oferecidos por organizações públicas e privadas.

Neste cenário, considerando a pluralidade de dispositivos utilizados no acesso à Internet e, em especial, a Web, surge a questão de como favorecer a usabilidade destas aplicações em dispositivos que possuem telas menores (smartphones) ou mesmo intermediárias (tablets), quando comparadas ao tamanho de telas dos computadores e laptops normalmente utilizados.

Dentre as soluções podem ser citadas o desenvolvimento de aplicações específicas voltadas aos dispositivos móveis e o uso das chamadas tecnologias responsivas.

As tecnologias responsivas permitem que as abordagens de design e desenvolvimento de um site ou aplicativo se adapte e forneça uma experiência de usuário otimizada em uma variedade de dispositivos e tamanhos de telas. Como exemplo, o CSS Media Queries que são uma parte essencial do desenvolvimento responsivo no qual o CSS pode ser adaptado com base nas características do dispositivo como largura da tela, altura, resolução, entre outros.

O Flutter é um framework para desenvolvimento de aplicativos mobile, web e desktop que utiliza a linguagem de programação Dart, que foi anunciada pela primeira vez em maio de 2017, mas só foi lançado uma versão estável em dezembro de 2018, criada pela Google e que permite a criação de aplicativos (apps) com facilidade e produtividade(FRAMEWORK Flutter. 2017).

Uma das principais vantagens de programar usando o Flutter para seu desenvolvimento mobile é a possibilidade de utilizar uma única base de código e criar aplicativos que podem ser executados tanto nos dispositivos com sistemas Android como iOS.

Bastante conhecida entre os desenvolvedores, o Flutter ainda é desconhecido de grande parte dos acadêmicos de cursos de Computação, e por essa razão, este trabalho se propõe a apresentar o desenvolvimento de um aplicativo por meio do framework Flutter, para favorecer a disseminação do conhecimento entre os acadêmicos do Câmpus de Ponta Porã.

2. O AMBIENTE DE DESENVOLVIMENTO FLUTTER

O Flutter é um conjunto de bibliotecas e ferramentas de código aberto desenvolvido pela Google, pensado e projetado para criar aplicativos nativos de alta qualidade, que utilizem uma única base de código e possibilitem a geração de aplicativos para diversas plataformas e/ou sistemas operacionais.

Utiliza a linguagem de programação Dart, também desenvolvida pela Google, e oferece uma abordagem inovadora para o desenvolvimento de aplicativos, permitindo a construção de interfaces de usuários bonitas e responsivas. A arquitetura do ambiente permite a sua rápida atualização e a implementação de novos recursos, além de proporcionar uma experiência consistente em diferentes plataformas.

Essa versatilidade e eficiência tornaram o Flutter uma escolha popular entre as empresas de desenvolvimento de aplicações para Web e dispositivos móveis, permitindo a criação de aplicativos visualmente atraentes e funcionais com grande produtividade.

Outra vantagem de usar Flutter são os widgets personalizáveis, que podem ser descritos como blocos de construção que apoiam o desenvolvimento da interface com o usuário. Esses widgets são altamente personalizáveis e oferecem flexibilidade para criar designs exclusivos e interativos, ao oferecer ao desenvolvedor uma base para que, caso desejem, possam realizar customizações pertinentes ao projeto.

Uma das características mais bem elogiadas pela comunidade de desenvolvedores é o chamado “Hot Reload”, que permite aos desenvolvedores visualizar imediatamente as alterações feitas no código, agilizando o processo de desenvolvimento e testes dos aplicativos(ZAMMETTI, Frank. 2020).

No contexto das ferramentas de código aberto não se pode ignorar o suporte oferecido pela chamada comunidade de desenvolvedores, que são profissionais e entusiastas da tecnologia que dedicam seu tempo a apoiar outros profissionais e ao

desenvolvimento do ambiente. No caso do Flutter, sua comunidade é muito ativa e engajada, em franco crescimento, significando uma ampla gama de recursos, bibliotecas e suporte disponível para os desenvolvedores e usuários do ambiente (ZAMMETTI, Frank. 2020).

Falar também das desvantagens que o Flutter tem como o tamanho dos aplicativos que por conseguir integrar mais de uma plataforma o tamanho de arquivo é maior se comparado com aplicativos nativos, isso pode impactar no tempo de download assim como no espaço de armazenamento do dispositivo. Também ressalta a falta de algumas integrações com recursos nativos. O Flutter oferece uma maneira de se comunicar com o código nativo e vice-versa, no entanto, o uso desses canais podem ser mais complexos, especialmente para desenvolvedores que não estão familiarizados com as APIs nativas da plataforma.

No geral, o Flutter é uma ferramenta poderosa para construção de aplicativos multiplataformas, ou responsivos, com uma aparência nativa e desempenho de alta qualidade, que simplifica o processo de desenvolvimento e que oferece boa experiência de uso aos seus usuários.

2. 1 DART

Dart é uma linguagem de programação livre e de código aberto para o desenvolvimento de aplicativos com alta qualidade em qualquer plataforma presente no framework Flutter.

Desenvolvida pela Google, ela é projetada para ser eficiente, robusta e adequada para uma ampla gama de casos de uso, desde aplicativos móveis até aplicações web. Pois tem uma sintaxe que se assemelha a de muitas outras linguagens, como JavaScript, Java e C#, com isso a curva de aprendizado se torna mais suave para os desenvolvedores já familiarizados com outras linguagens.

A linguagem Dart oferece uma tipagem opcional e forte, o que significa que você pode optar por fornecer tipos explícitos ou deixar o sistema de tipos inferir automaticamente. Isso oferece uma flexibilidade aos desenvolvedores e permite a utilização de uma tipagem forte quando desejado, além de contribuir para um código mais robusto e menos propenso a erros (SISTEMA Dart. 2022).

Com isso o código em linguagem Dart ganha eficiência e desempenho, ao ser projetado para ter um desempenho rápido utilizando técnicas de otimização que o

tornam adequado para o desenvolvimento de aplicativos que requerem alto desempenho em sua execução (ZAMMETTI. Frank. 2020).

Não menos importante, a linguagem Dart oferece suporte nativo para programação assíncrona, facilitando o tratamento de operações que ocorrem de forma não sequencial, como as requisições de rede. Além disso, possui recursos de programação funcional, como funções de alta ordem e operações em listas, que podem tornar o código mais conciso e legível.

Junto a isso, o Dart possui um conjunto de ferramentas robusto, incluindo um compilador just-in-time(JIT) para desenvolvimento rápido e um compilador ahead-of-time(AOT) para produzir código nativo otimizado para a produção. Essas funcionalidades conferem ao Dart uma abordagem flexível ao ciclo de vida do desenvolvimento de software. Durante as fases iniciais de desenvolvimento o compilador JIT do Dart proporciona uma vantagem significativa, compilando o código-fonte conforme necessário e com essa abordagem facilita a rápida iteração e depuração, com alterações refletindo instantaneamente durante a execução. À medida que o projeto amadurece e se encaminha para a produção, a transição para o compilador AOT torna-se crucial, pois o código nativo gerado é altamente otimizado, adaptado à arquitetura específica da plataforma de destino. Isso não apenas impulsiona o desempenho, mas também simplifica a distribuição, pois os binários resultantes são independentes da plataforma, reduzindo o tempo de inicialização e eliminando a necessidade de interpretação ou compilação JIT em ambiente de produção (CHAVAN. Balram Morsing, 2020).

Além de ser uma linguagem versátil e poderosa que se integra perfeitamente com o Flutter, possui uma comunidade de desenvolvedores bastante ativa, que contribui com o desenvolvimento e manutenção de bibliotecas e ferramentas para facilitar o desenvolvimento. Dessa forma, proporciona aos desenvolvedores uma experiência de desenvolvimento consistente e eficiente para criar aplicativos para multiplataformas.

A Figura 1, mostra um exemplo de um pequeno trecho de código em linguagem Dart, responsável perguntar o nome e a idade sendo digitado pelo teclado e depois exibindo o que foi escrito.

Figura 1 — Código em Dart perguntando e armazenando brevemente o nome e a idade de alguém e depois mostrando o que for digitado para as perguntas

```dart
import 'dart:io';

void main() {
  // Perguntando o nome
  stdout.write('Qual é o seu nome? ');
  String nome = stdin.readLineSync()!;

  // Perguntando a idade
  stdout.write('Qual é a sua idade? ');
  int idade = int.parse(stdin.readLineSync()!);

  // Exibindo a mensagem com o nome e a idade
  print('Hello, $nome! Você tem $idade anos.');
}
```

Fonte: Elaborado pelo autor (2024).

2. 2 INSTALAÇÃO NECESSÁRIA PRO FLUTTER

Nesta seção, serão apresentadas as configurações necessárias para o desenvolvimento de aplicativos e aplicações com Flutter, como o Ambiente Integrado de Desenvolvimento (IDE).

O primeiro passo no desenvolvimento de um aplicativo ou aplicação é a escolha do ambiente integrado de desenvolvimento, dentre as várias possibilidades de IDE podemos citar: o Intellij ambiente leve e generalista, que oferece suporte a várias linguagens e tem um ambiente para colocarmos emuladores. O VSCode, uma IDE considerada leve, mas que exige o uso de um emulador externo a ele.

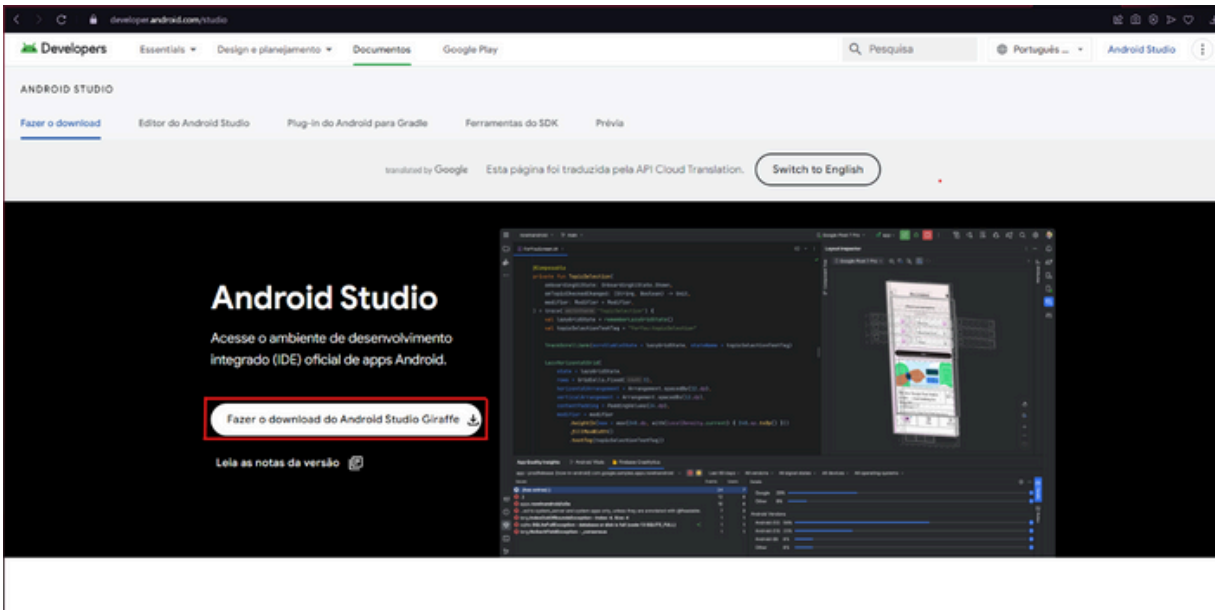
Outro um pouco mais robusto, mas específico para o sistema iOS, é o XCode, que possuía a vantagem de ser muito bem performático aos usuários do sistema operacional iOS. Da mesma forma, usuários do sistema operacional Android possuem o Android Studio.

Algumas outras opções on-line que podem ser mencionadas são: FlutLab, que é leve e pago, e o FlutterFlow, que oferece uma versão gratuita, contudo, limitada nos recursos oferecidos.

Neste trabalho, optamos por comentar a combinação entre os ambientes Android Studio e o VSCode, sendo VSCOde usado como editor e ambiente de desenvolvimento integrado, mas que por não possuir emulador, exige o uso do Android Studio para avaliação da aplicação.

Para baixar o Android Studio basta entrar no site do Android Studio “https://developer.android.com/studio?hl=pt-br” e clicar na opção “Fazer o download do Android Studio Giraffe”, como ilustrado na Figura 2.

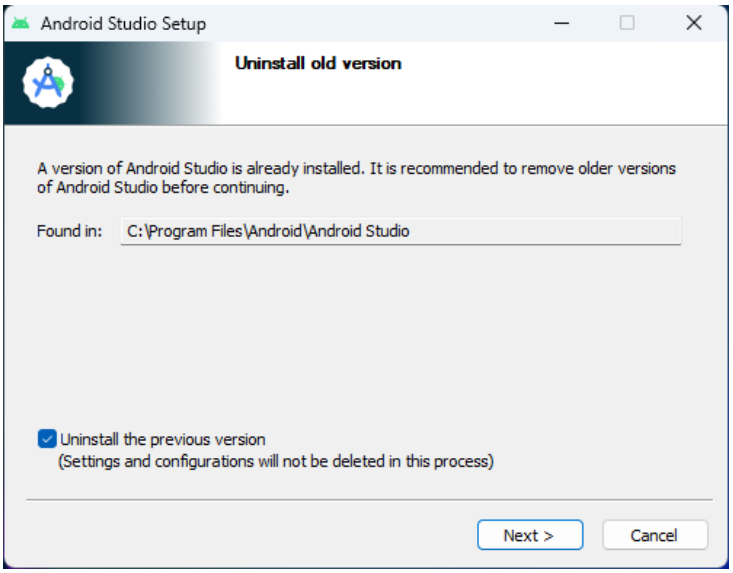
Figura 2 — Tela para o download do Android Studio



Fonte: Elaborado pelo autor (2024).

Na tela seguinte, escolha o diretório para onde gostaria de transferir, ou aceite a sugestão apresentada e faça a instalação (Figura 3).

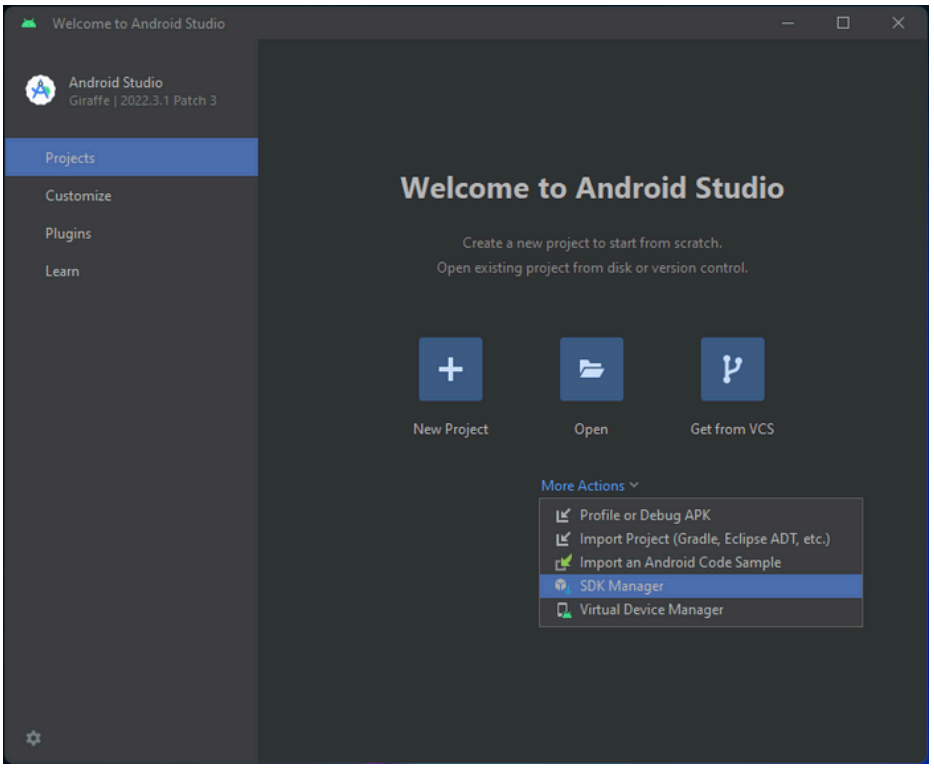
Figura 3 — Tela de instalação do Android Studio onde decide a pasta onde será instalado



Fonte: Elaborado pelo autor (2024).

Após a transferência, execute a aplicação e com a opção “Projects” selecionada, escolha a opção “SDK Manager” nas opções “More Actions”, conforme a Figura 4.

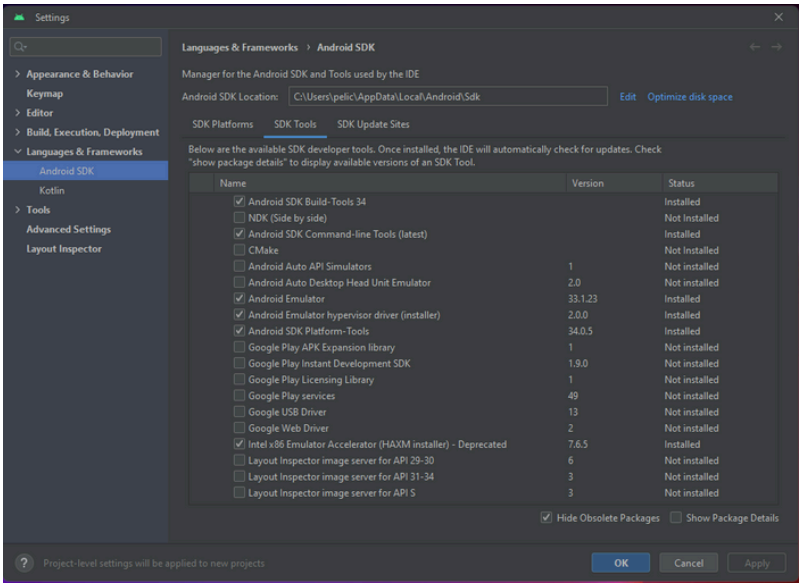
Figura 4 — Tela do Android Studio onde gerencia as SDKs



Fonte: Elaborado pelo autor (2024).

Na tela seguinte, figura 5, na opção “SDK Tools”, certifique se que as opções abaixo estão assinaladas.

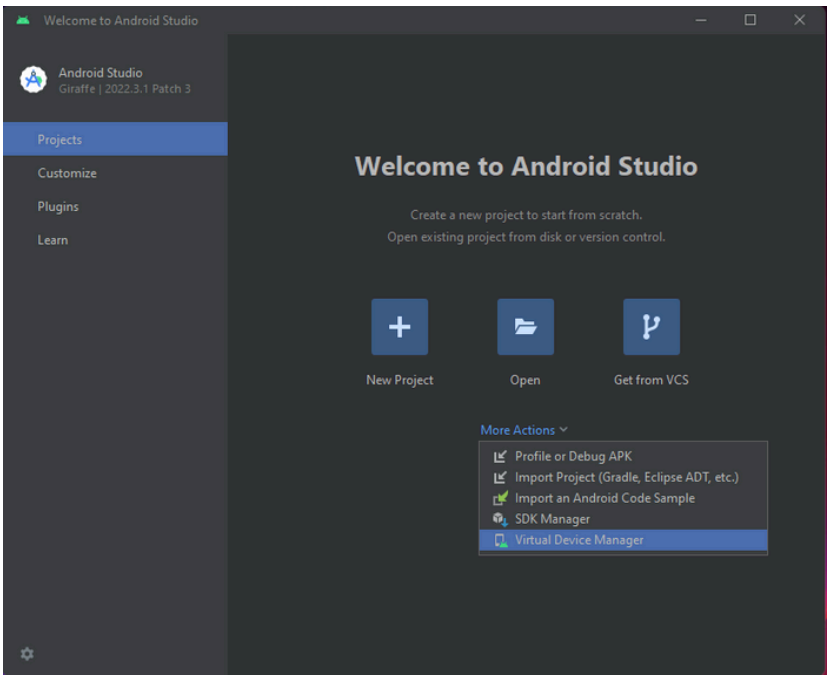
Figura 5 — Tela do Android Studio que lista todos os SDK Tools



Fonte: Elaborado pelo autor (2024).

De volta à tela principal, é necessário associar um emulador ao ambiente integrado de desenvolvimento, para isso escolha a opção “Virtual Device Manager”, conforme a tela da figura 6.

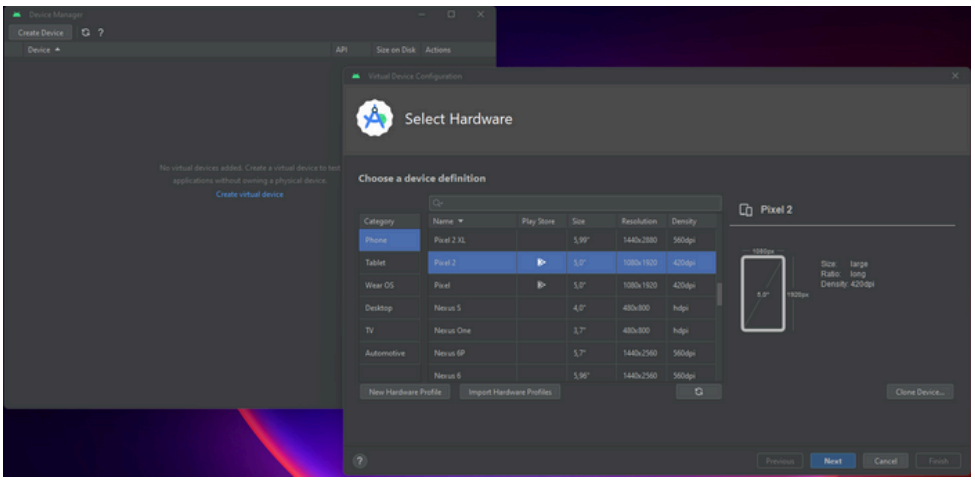
Figura 6 — Tela do Android Studio onde seleciona para abrir as configurações dos emuladores



Fonte: Elaborado pelo autor (2024).

Na tela que será aberta (Figura 7), será possível encontrar uma lista de opções de emuladores que o Android Studio oferece, neste trabalho fizemos a opção por emular o Phone Pixel 2 com uma tela de 5 polegadas (aprox. 13 cm).

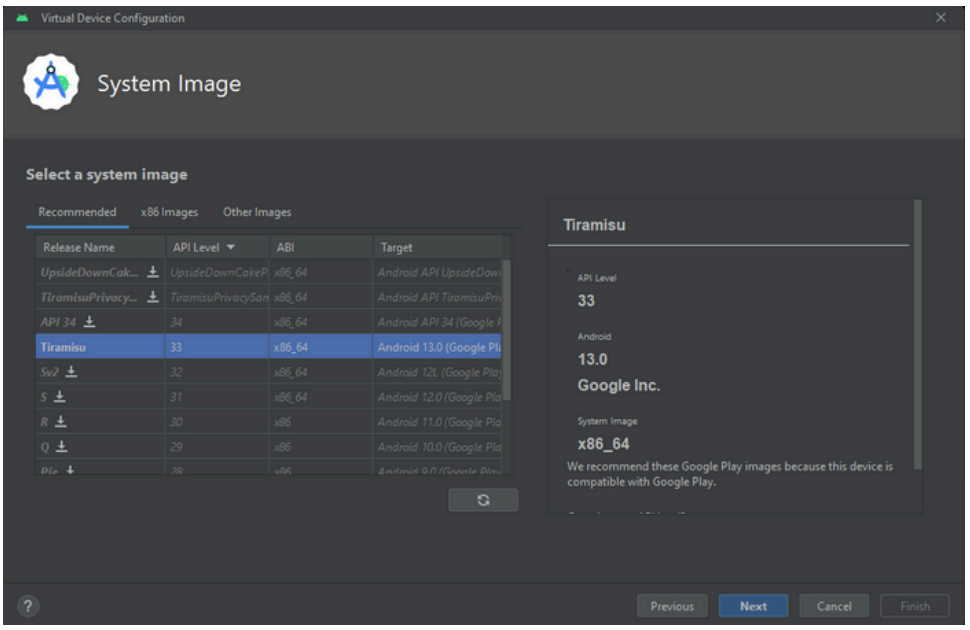
Figura 7 — Tela do Android Studio que lista os devices que podem ser usados



Fonte: Elaborado pelo autor (2024).

Definido o tamanho da tela, faz-se necessário escolher a versão do Android a ser utilizada como base para o aplicativo. Empresas de desenvolvimento costumam definir uma versão mínima para o desenvolvimento de suas aplicações para evitar problemas de compatibilidade e suporte com versões antigas. Conforme a Figura 8, selecionamos a versão **Tiramisu** que é a versão do Android 13.

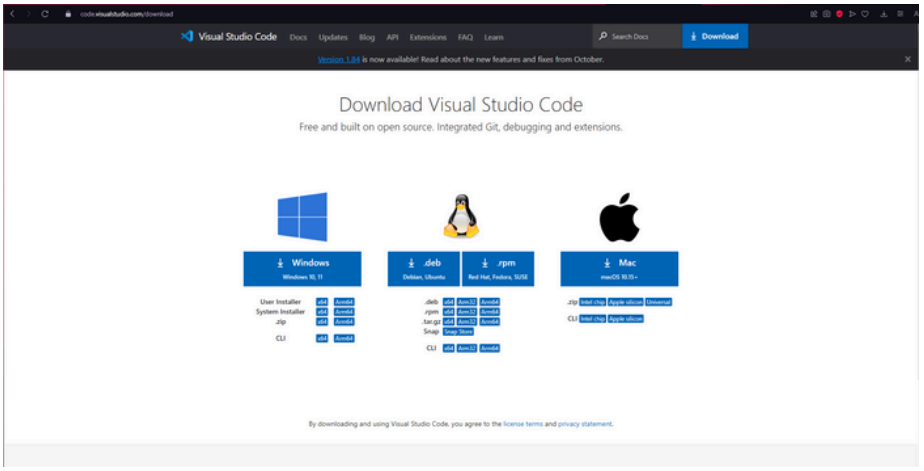
Figura 8 — Tela do Android Studio onde está selecionada a versão do android



Fonte: Elaborado pelo autor (2024).

Após a instalação do emulador, o próximo passo é a instalação do VSCode que pode ser feita pelo próprio site deles “https://code.visualstudio.com/download”, conforma a Figura 9.

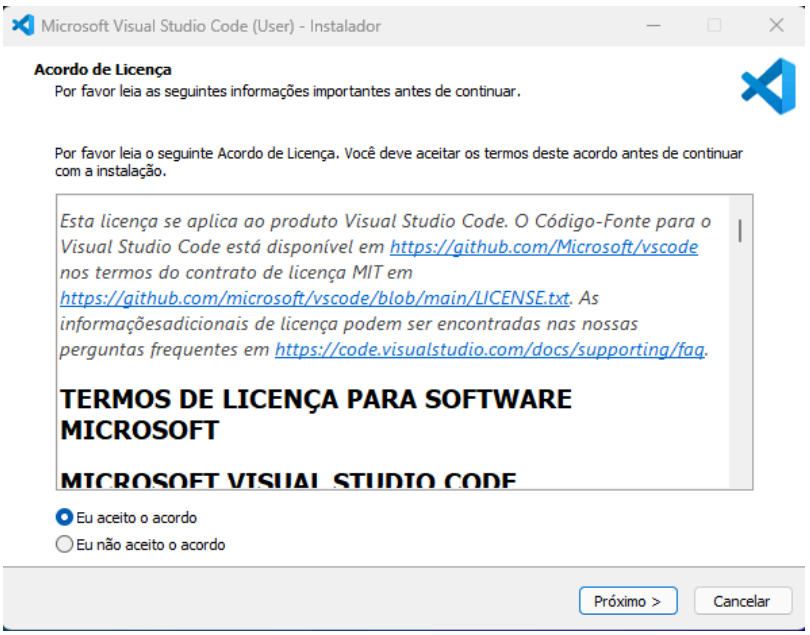
Figura 9 — Tela do navegador que mostra as diversas possibilidades de instalação do Visual Studio Code



Fonte: Elaborado pelo autor (2024).

Novamente, após o download inicie o processo de instalação clicando em Próximo, após aceitar os termos da licença de uso (Figura 10).

Figura 10 — Acordo da Microsoft para usar o VSCode



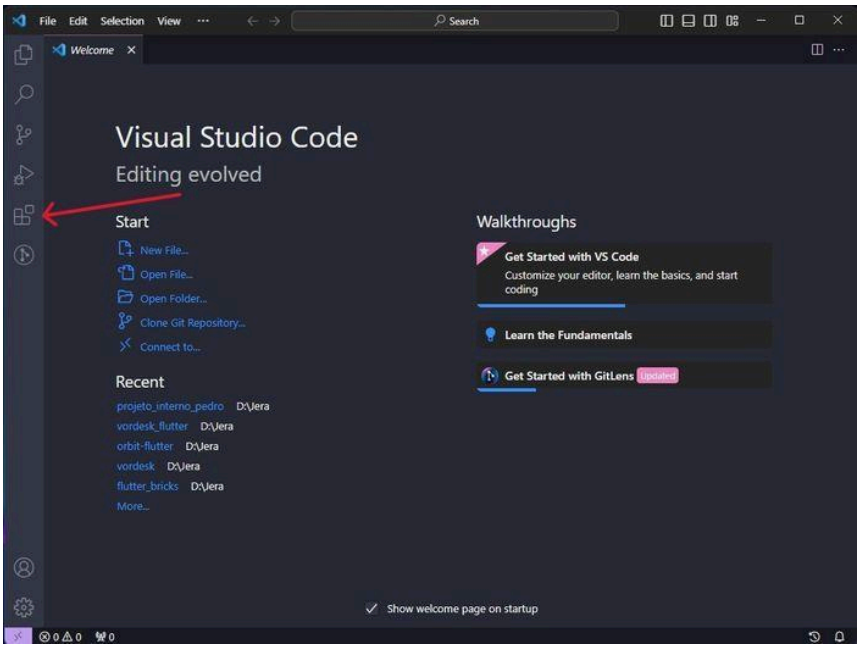
Fonte: Elaborado pelo autor (2024).

Após finalizar a instalação do VSCode é necessário ativar algumas extensões que serão importantes para o desenvolvimento em Flutter, sendo elas:

- Dart
- Flutter
- Flutter Intl
- DotENV
- Code Spell Checker

Para instalar essas extensões abra o VSCode e vá em Extensions na barra lateral esquerda como na figura 11.

Figura 11 — Tela do VSCode mostrando onde ficam as extensões



Fonte: Elaborado pelo autor (2024).

Até o momento, fizemos a instalação dos recursos mínimos necessários para o desenvolvimento de aplicações baseadas em Flutter, contudo, algumas outras extensões podem ser ativadas com o intuito de alterar o visual do VSCode, oferecendo personalizações aos seus usuários. São elas:

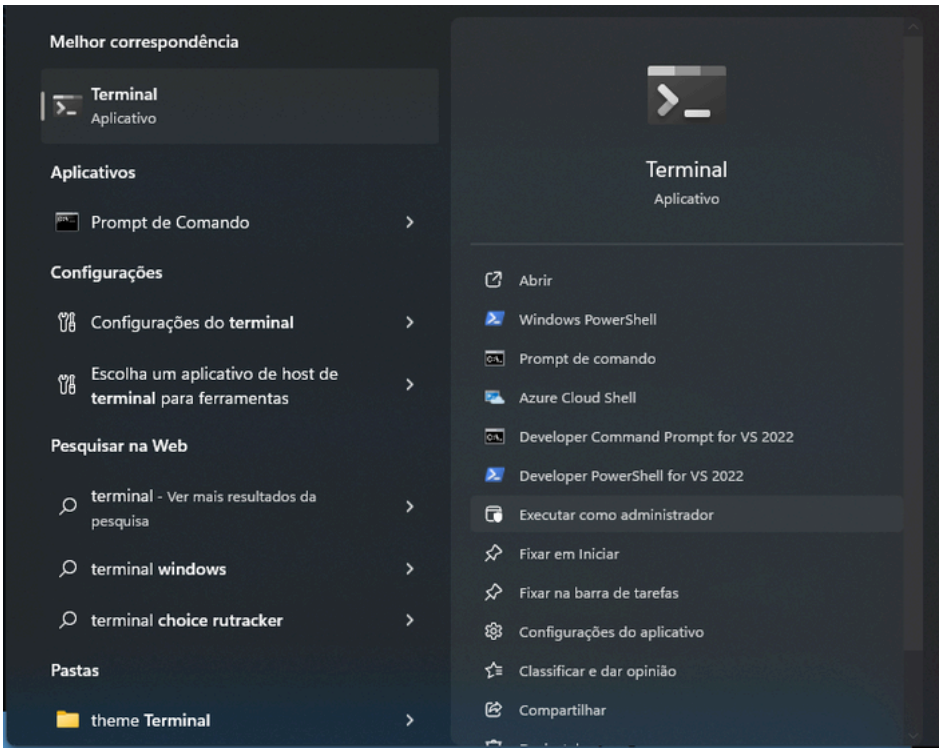
- Dracula Oficial
- Material Icon Theme
- Better Comments

Dracula Oficial é uma extensão de cor do VSCode e da formatação do código conforme a linguagem programada. O Material Icon Theme muda os ícones na lista lateral que pode auxiliar na procura de arquivos() e o Better Comments destaca linhas comentadas ou alguns erros ajudando a enxergar possíveis erros e deixando o código mais legível.

Para finalizar, após a instalação do Android Studio e do VSCode, instalaremos o Flutter na sua máquina. Para isso, utilizaremos o aplicativo do Terminal, no modo Administrador, do sistema operacional Windows, contudo, vale ressaltar que existem outras formas de realizar a instalação do ambiente.

Para abrir o Terminal basta clicar no botão “Windows” do seu teclado e escrever “Terminal” e clicar em “Executar como administrador” como ilustrado na Figura 12.

Figura 12 — Menu do Windows com a pesquisa Terminal



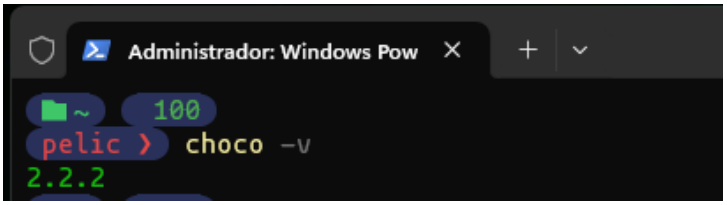
Fonte: Elaborado pelo autor (2024).

Com o terminal aberto como administrador, agora começaremos a colocar os comandos para a instalação do Chocolatey que será responsável por instalar o Flutter. Digite o comando descrito a seguir seguindo pela tecla Enter para dar início ao processo de instalação.

Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-ObjectSystem.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

Para ter certeza que o Chocolatey foi instalado corretamente, na tela do próprio Terminal, após a finalização da instalação digite o comando $choco -v . Se tudo tiver correto, a versão instalada será apresentada como resposta ao comando, conforme a Figura 13.

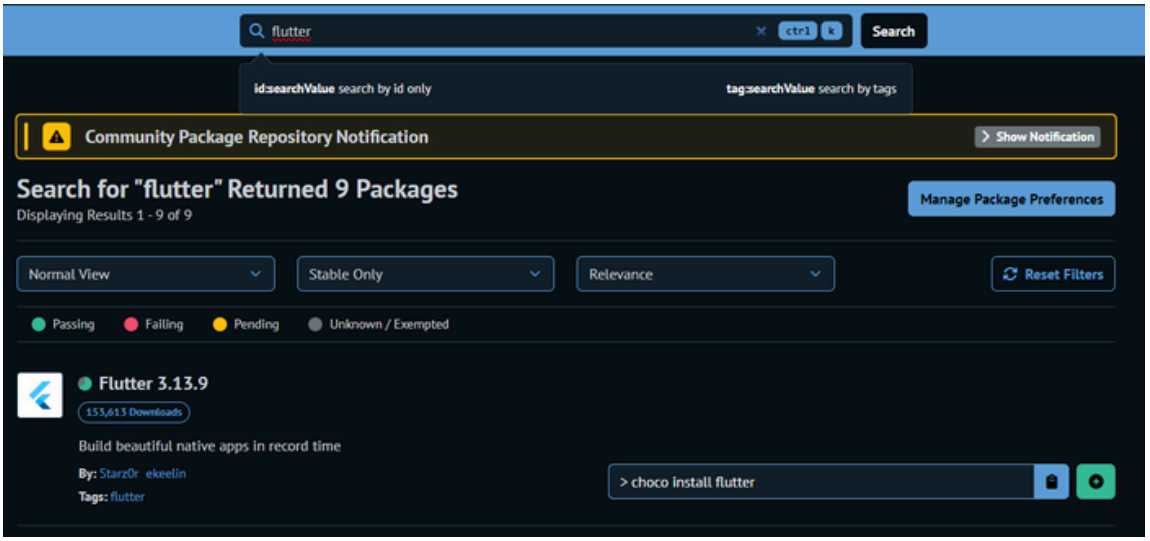
Figura 13 — Comando de verificação de versão do chocolatey



Fonte: Elaborado pelo autor (2024).

Com o Chocolatey instalado podemos colocar os próximos comandos para a instalação do Flutter, para isso, no próprio site do Chocolatey a comunidade libera os pacotes de instalação e conseguimos pesquisar os pacotes no site: “https://community.chocolatey.org/packages”. Vamos então pesquisar o pacote do Flutter (Figura 14).

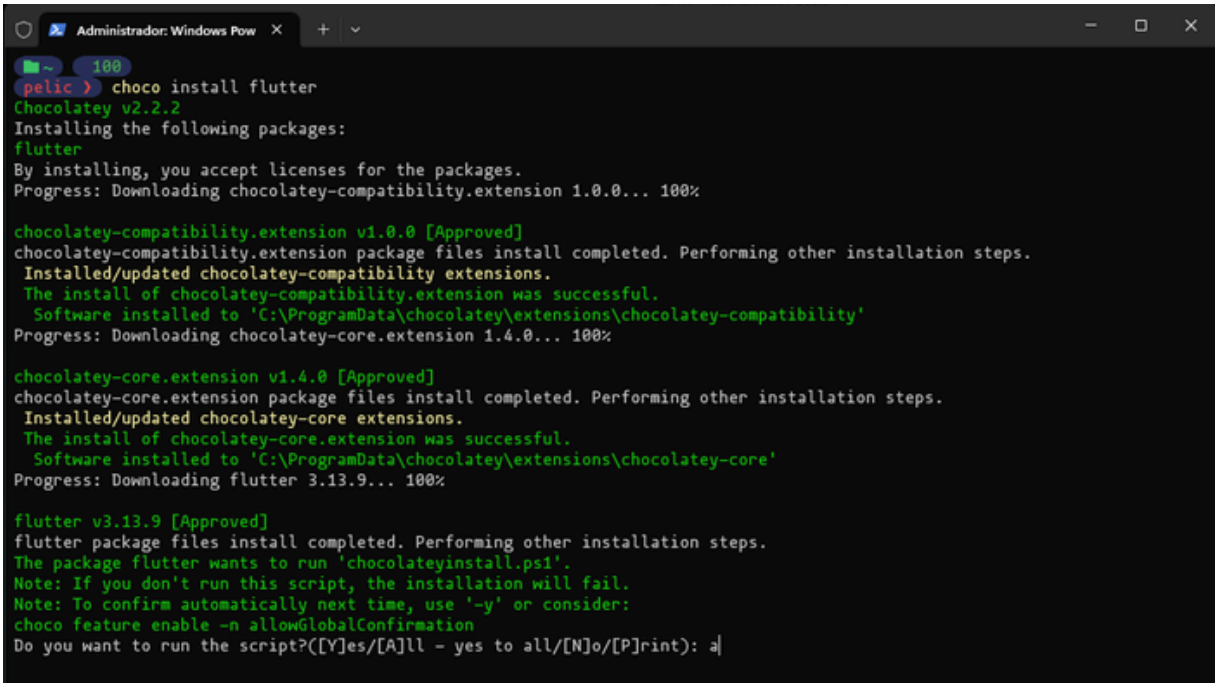
Figura 14 — Pesquisa de como instalar o pacote do Flutter com o Chocolatey



Fonte: Elaborado pelo autor (2024).

Como podemos ver na figura 14 acima basta colocar o comando “choco install flutter” e aceitar todos os scripts como ilustrado na figura 15.

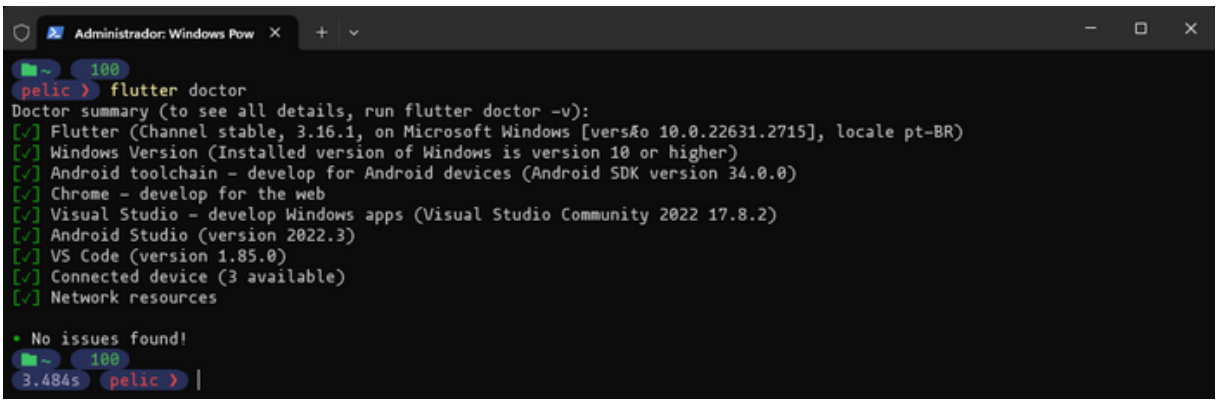
Figura 15 — Terminal perguntando se aceita a instalação



Fonte: Elaborado pelo autor (2024).

Quando acabar a instalação do Flutter coloque o comando “flutter doctor” para ter certeza que tudo está correto para usar o Flutter em sua máquina (figura 16).

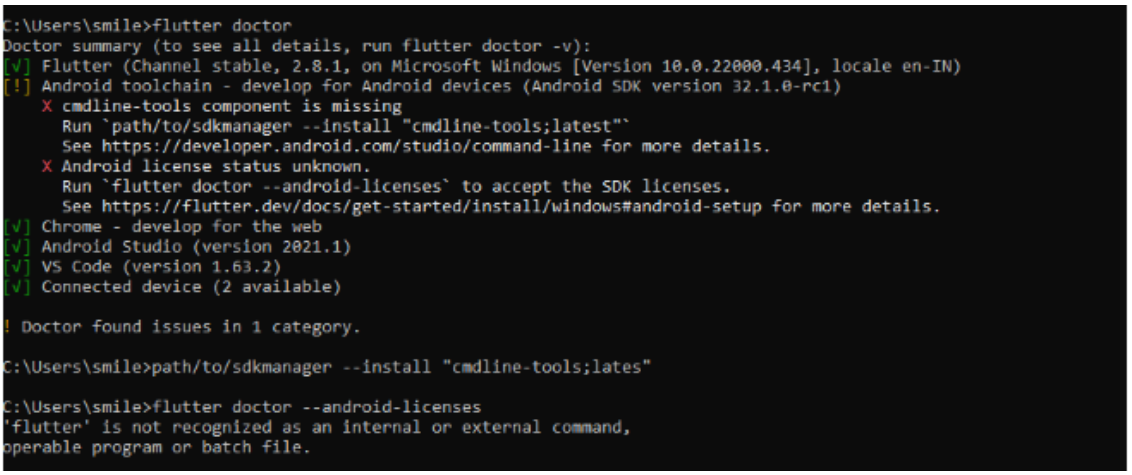
Figura 16 — Terminal com verificação se está tudo certo com o flutter permitindo iniciar os projetos



Fonte: Elaborado pelo autor (2024).

Caso algum dos tópicos acima não esteja correto(figura 17), no próprio Terminal aparecerá as devidas soluções que precisará ser feita para o Flutter funcionar corretamente na sua máquina.

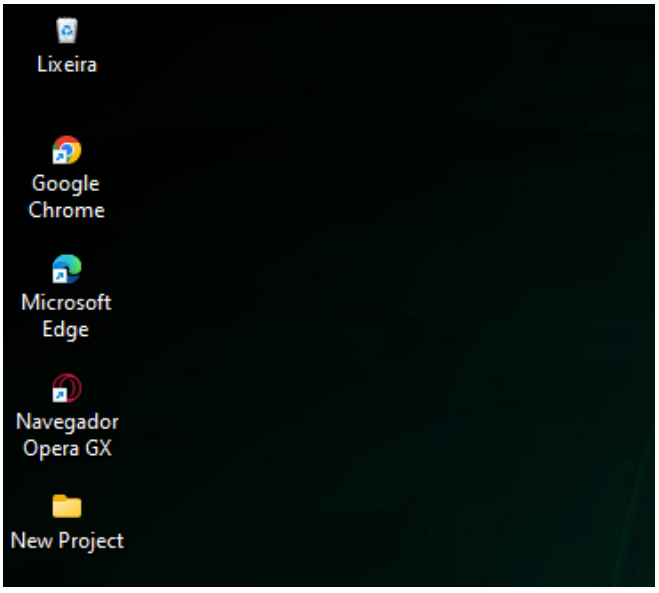
Figura 17 — Terminal com caso de erro no flutter e mostrando a possível solução



Fonte: Elaborado pelo autor (2024).

Com tudo devidamente instalado poderemos iniciar a criação de um projeto, para isso crie uma pasta onde ficará todos os seus projetos figura 18.

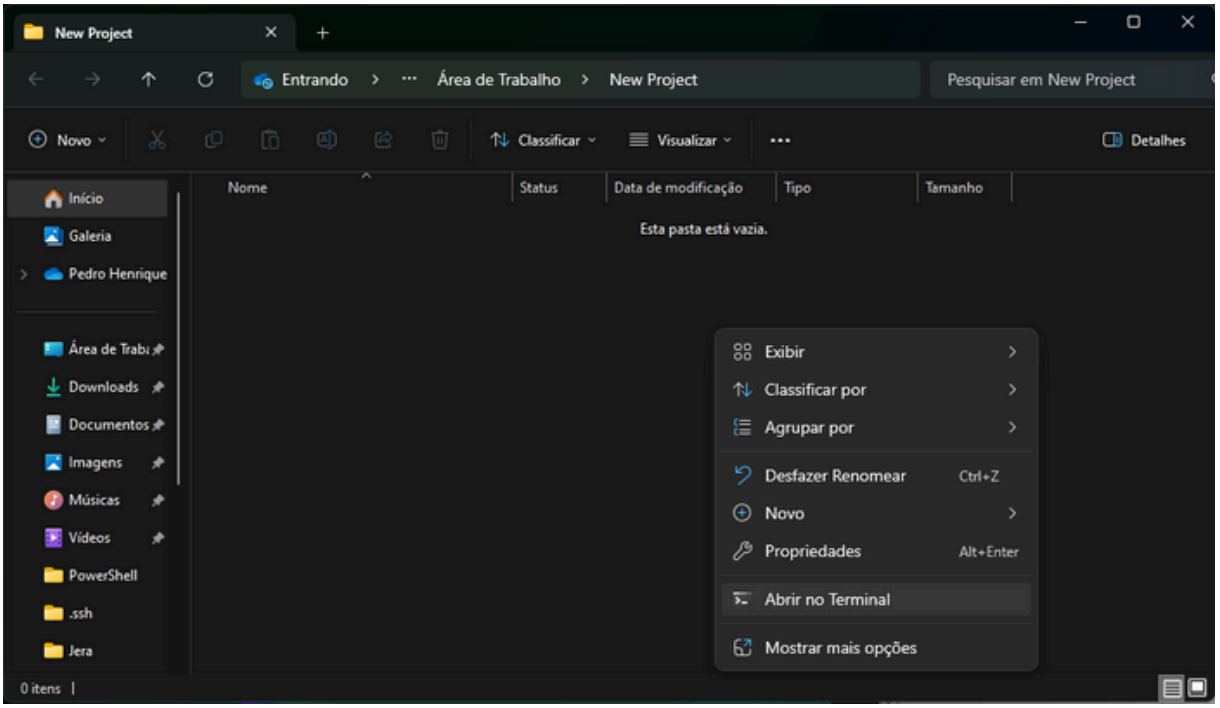
Figura 18 — Pasta onde ficará guardado todos os possíveis projetos



Fonte: Elaborado pelo autor (2024).

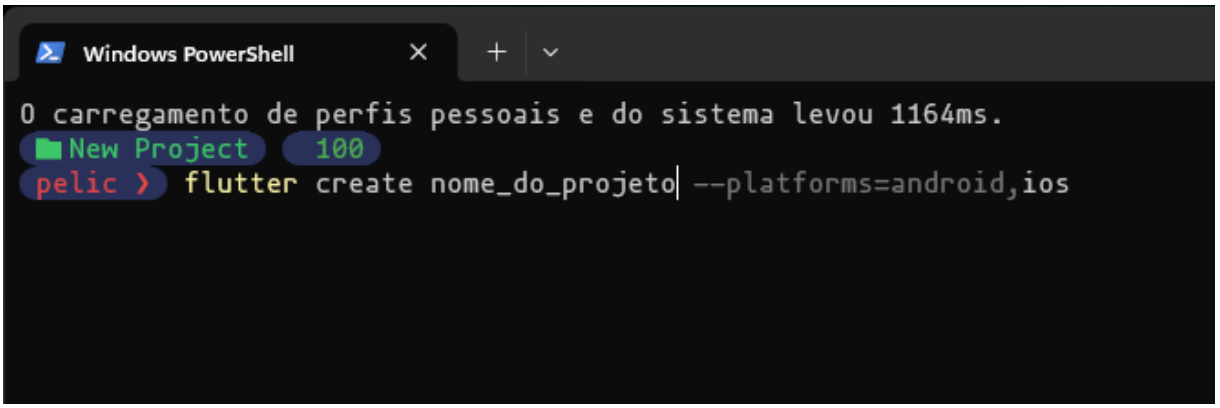
Com a pasta criada existem duas formas para criar o projeto, tanto pelo terminal quanto abrindo o Android Studio e criando por lá. Mostremos ambas as formas. Com a pasta onde ficaram os projetos aberta clica com o botão direito do mouse e selecione a opção abrir no terminal (Figura 19) ou pelo terminal tente chegar até na pasta. Importante ter certeza na pasta certa para rodar o comando de criação do projeto (Figura 20).

Figura 19 — Figura mostrando a opção de “abrir no terminal” para abrir o terminal com o diretório já na pasta



Fonte: Elaborado pelo autor(2024).

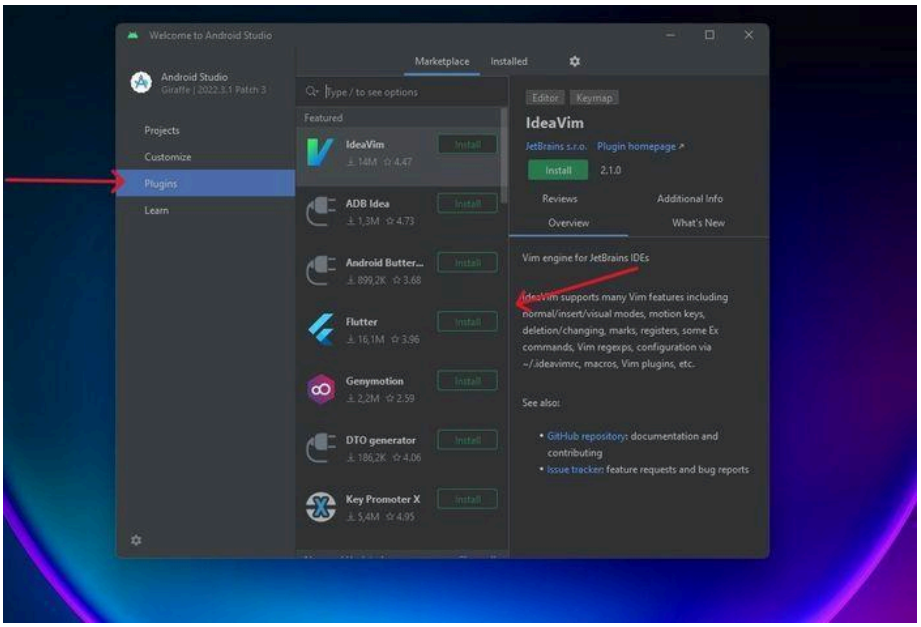
Figura 20 — Terminal com o comando para criar o projeto em Flutter



Fonte: Elaborado pelo autor (2024).

Pelo Android Studio antes de clicar em “New Project” vamos ter que instalar o plugin do Flutter como ilustrado na figura 21 e depois terá que reiniciar a IDE.

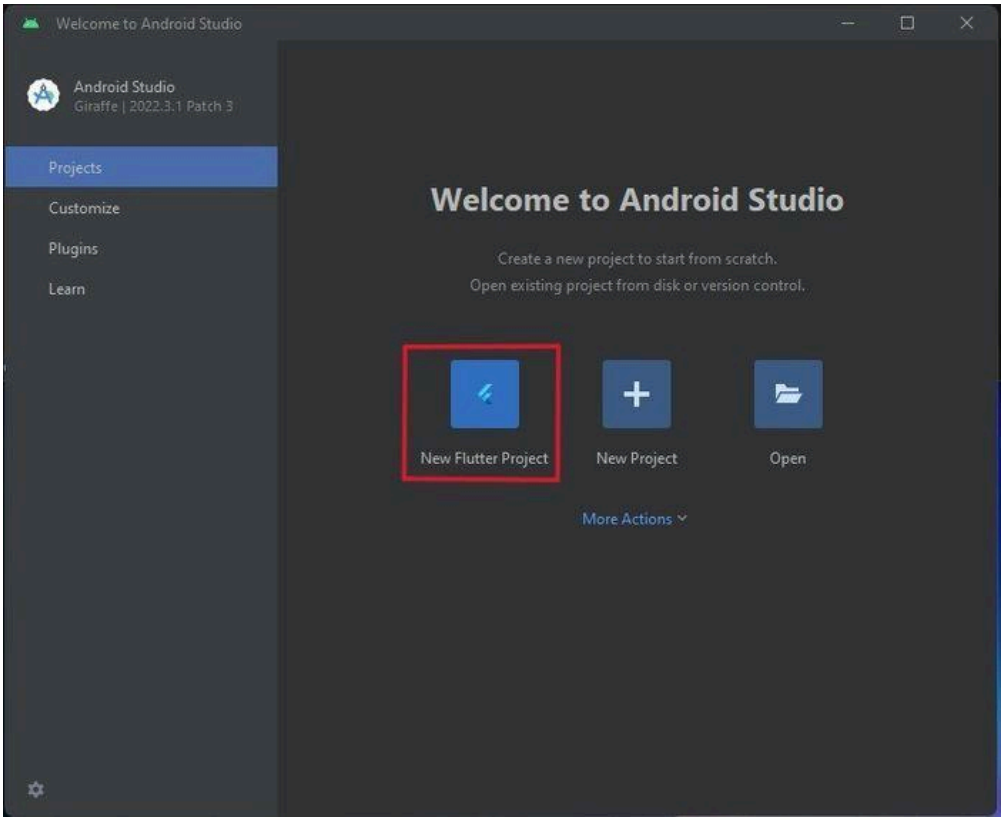
Figura 21 — Mostrando o plugin do flutter que tem que ser instalado



Fonte: Elaborado pelo autor (2024).

Com o Plugin instalado e com a IDE reiniciada, já aparecerá a opção de criar um projeto novo em Flutter (figura 22).

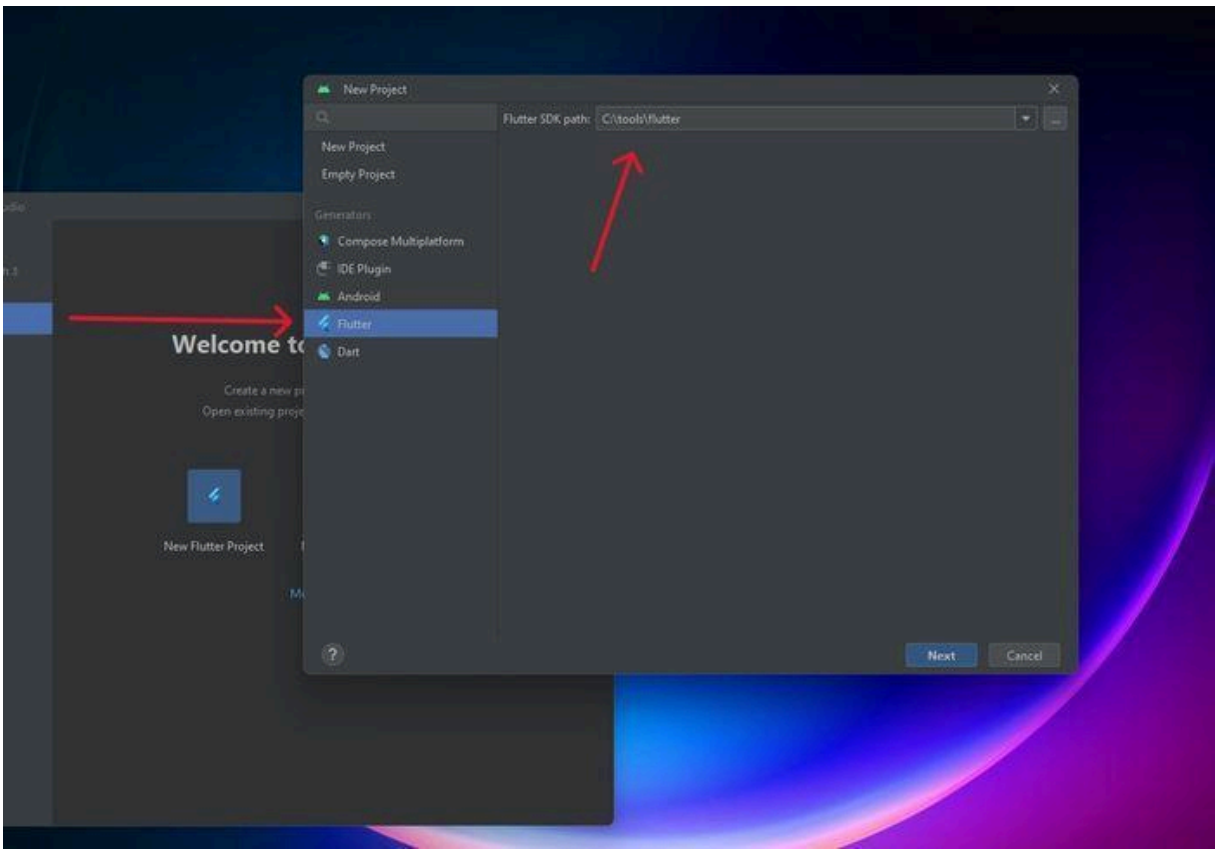
Figura 22 — Android Studio com o plugin do Flutter instalado



Fonte: Elaborado pelo autor (2024).

Agora escolha a opção para criar com o Flutter e dai terá que escolher a pasta onde o foi instalado o flutter na sua máquina. Caso tenha seguido até o momento estará nesse destino “C:\tools\flutter” como ilustrado abaixo (Figura 23).

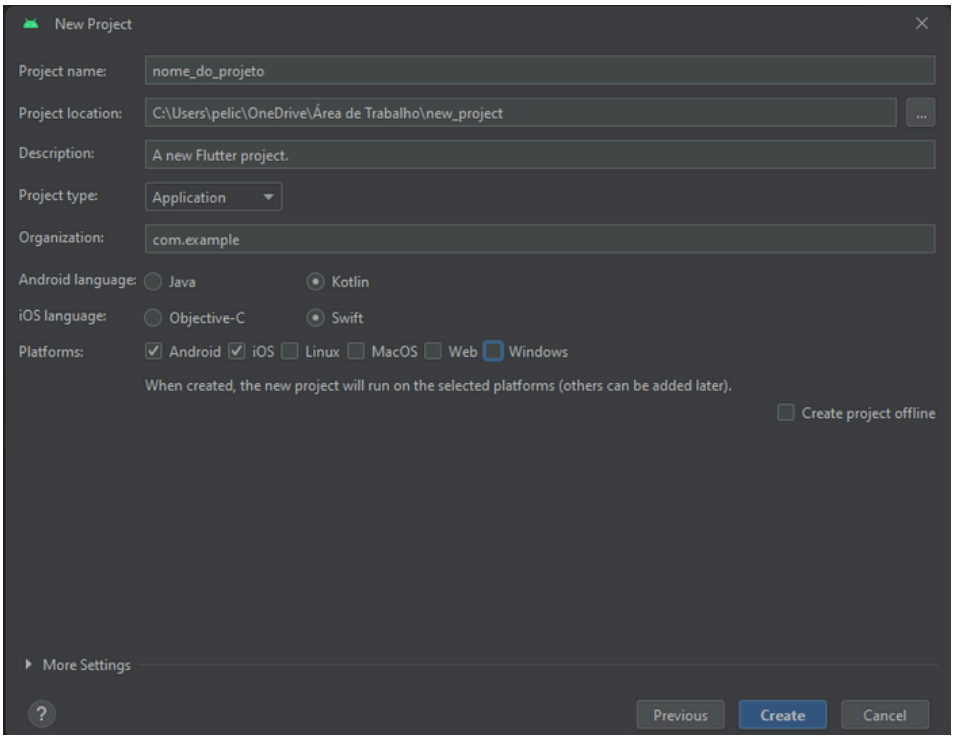
Figura 23 — Criação do projeto em flutter onde terá que colocar a pasta onde o flutter está instalado



Fonte: Elaborado pelo autor (2024).

Clicando em “Next” aparecerá uma parte importante onde colocaremos o nome do projeto. Veja na figura 24 o nome do projeto, é indicado sempre ser com letras minúsculas, sem nenhum caractere estranho e sem espaço usando somente o “Subtraço” para separar as palavras. Logo após o local onde ficará o projeto. A parte da descrição é opcional, caso queira adicionar algo sobre o projeto. O próximo tópico é o tipo do projeto, nesse caso deixe “Application”. O Organization é o domínio do projeto, normalmente em empresas se coloca o domínio da empresa para ter vínculo com a empresa. As linguagens tanto de Android como do iOS podem ser deixados como na figura 24, Kotlin e Swift que são as linguagens nativas que o projeto irá construir. Por último as plataformas a qual poderá gerar os aplicativos e como na figura 24 podemos deixar marcado somente Android e iOS. Feito tudo isso pode clicar em “Create”.

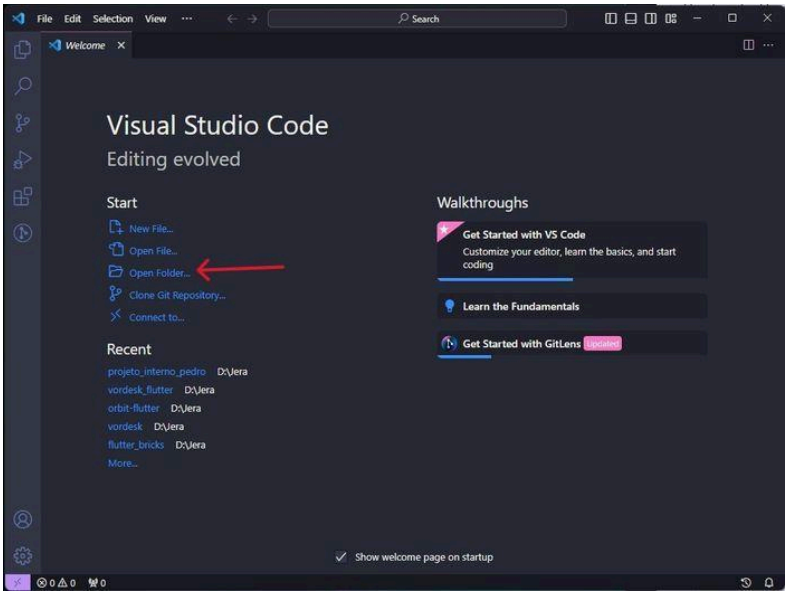
Figura 24 — Parte de colocar o nome do projeto



Fonte: Elaborado pelo autor (2024).

Com o projeto criado agora pode fechar o Android Studio e abriremos o VSCode onde será o nosso ambiente de programação. Para isso, ao abrir o VSCode clica em “Open Folder” para abrir a pasta onde está o projeto criado (Figura 25).

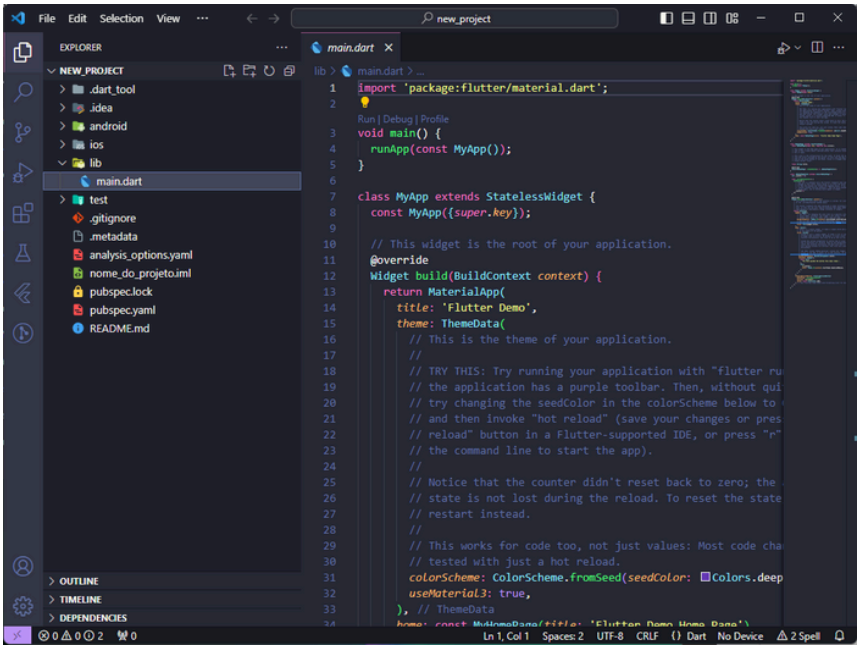
Figura 25 — Página inicial do VSCode



Fonte: Elaborado pelo autor (2024).

Quando abrir aparecerá vários arquivos no lado esquerdo figura 26. Na pasta “lib” será possível encontrar a “main” onde já poderá emular o código que já vem.

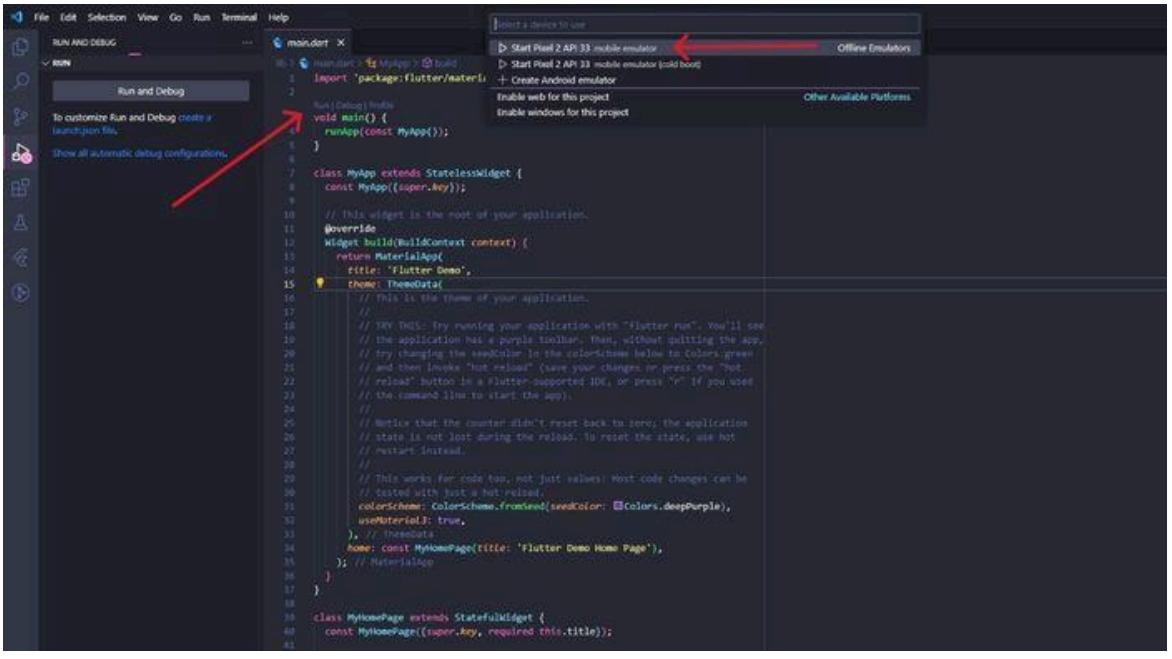
Figura 26 — Tela da main do projeto



Fonte: Elaborado pelo autor (2024).

Para rodar esse código basta clicar em “Run” em cima da função main como ilustrado na figura 27. E depois escolher qual emulador gostaria de usar.

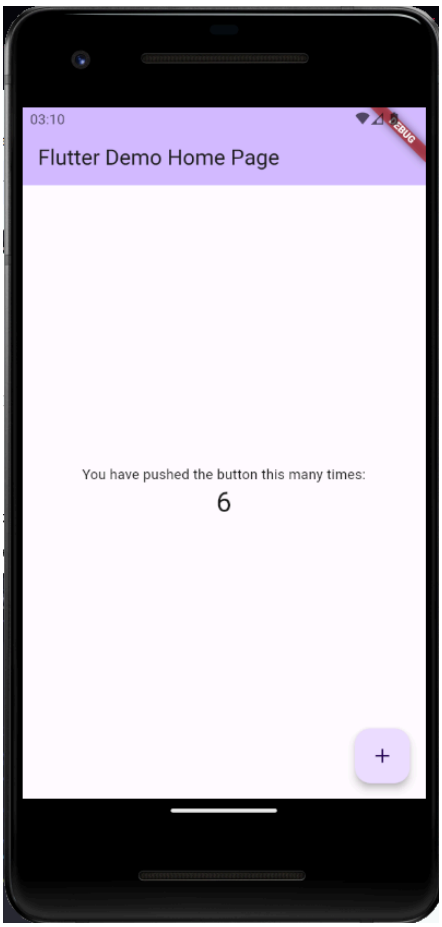
Figura 27 — Rodando o projeto pela primeira vez e escolhendo o emulador



Fonte: Elaborado pelo autor (2024).

Sempre vem um código já pronto com um botão com o símbolo + que toda vez que pressionado acrescenta um a uma variável e mostrando na tela quantas vezes esse botão foi pressionado (Figura 28).

Figura 28 — Emulador com o código que já vem quando se cria um projeto



Fonte: Elaborado pelo autor (2024).

2. 3 ENTENDENDO O CÓDIGO FLUTTER

Na programação orientada a objetos trabalhamos com classes e no Flutter existem duas classes principais que são usadas que são “StatelessWidget” e “StatefulWidget”, que representam diferentes abordagens para a construção de widgets, dependendo da necessidade de gerenciar ou não o estado interno do widget.

Por exemplo, um “StatelessWidget” é imutável, o que significa que, uma vez que seus dados são definidos durante a construção, eles não podem ser alterados. Já no “StatefulWidget” seu estado é dinâmico, por poder ter um estado interno que

pode ser modificado ao longo do tempo, permitindo que o widget reaja a eventos, interações do usuário ou mudanças em variáveis internas.

Então quando se deve usar um, ou quando se deve usar o outro. Tudo depende do que a sua tela terá, se tiver informações estática como, texto, ou imagem, que não vão mudar ao longo do tempo, a melhor alternativa é usar “StatelessWidget”, já se a tela vai ter alterações recorrentes como na figura 28 em que tem um botão que contabiliza quantas vezes o botão é clicado temos que usar “StatefulWidget”.

Explicaremos um pouco o que esse código está fazendo.

Figura 29 — Primeira parte do código Dart que vem quando se cria um projeto

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

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ), // ThemeData
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    ); // MaterialApp
  }
}

```

Fonte: Elaborado pelo autor (2024).

Olhando para a figura 29, na primeira linha encontramos um comando import e logo após entres aspas simples um package, esse package é uma biblioteca em que o código está usando. Dentro dela que explica para o código o que é um widget ElevatedButton ou um Scaffold.

Entre a linha 3 a 5 podemos ver uma função void. Funções do tipo void não tem nenhum retorno para quem chamar essa função, diferentemente se colocarmos algum tipo antes o que traria uma obrigatoriedade de retornar o tipo que pedir. Existem alguns tipos como os booleanos, inteiros, widget entre outros.

Ocorre então a chamada da função runApp que é a função que dá o início ao projeto e dentro dela está uma classe chamada MyApp que já é a classe que terá as informações sobre a aplicação.

Da linha 7 à linha 21 é a classe que dará início a construir a tela. Na linha 11 podemos observar que está sendo chamado uma função buil que tem um tipo Widget. Por sua vez, na linha 12 está retornando um widget chamado MaterialApp esse widget é responsável por ter o nome do projeto no caso, que está na linha 13 (o title seria o nome do projeto “Flutter Demo”, logo após na linha 14 o theme será o tema que seu projeto terá, geralmente cor padrão que o projeto terá.

A próxima função chamada é a home onde é passado para ela outra classe MyHomePage (Figura 30) que por sua vez essa classe é a classe responsável pela criação de toda a tela.

Figura 30 — Segunda parte do código Dart que vem quando se cria um projeto

```
23  class MyHomePage extends StatefulWidget {
24    const MyHomePage({super.key, required this.title});
25
26    final String title;
27
28    @override
29    State<MyHomePage> createState() => _MyHomePageState();
30  }
31
32  class _MyHomePageState extends State<MyHomePage> {
33    int _counter = 0;
34
35    void _incrementCounter() {
36      setState(() {
37        _counter++;
38      });
39    }
40
41    @override
42    Widget build(BuildContext context) {
43      return Scaffold(
44        appBar: AppBar(
45          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
46          title: Text(widget.title),
47        ), // AppBar
48        body: Center(
49          child: Column(
50            mainAxisAlignment: MainAxisAlignment.center,
51            children: <Widget>[
52              const Text(
53                'You have pushed the button this many times:',
54              ), // Text
55              Text(
56                '$_counter',
57                style: Theme.of(context).textTheme.headlineMedium,
58              ), // Text
59            ], // <Widget>[]
60          ), // Column
61        ), // Center
62        floatingActionButton: FloatingActionButton(
63          onPressed: _incrementCounter,
64          tooltip: 'Increment',
65          child: const Icon(Icons.add),
66        ), // This trailing comma makes auto-formatting nicer for build methods. /
67      ); // Scaffold
68    }
69  }
70
```

Fonte: Elaborado pelo autor (2024).

Nessa classe encontramos uma estrutura um pouco diferente, pois essa tela tem um botão que ao clicar vai somando quantas vezes foi clicado e mostrado ao usuário no meio da tela.

Na linha 24 vemos o construtor da classe. Nesse exemplo o construtor nos diz que para essa classe ser usada ela precisa de um title e é isso que acontece na linha 18 da figura 29. Quando se chama a função MyHomePage dentro dela tem um parâmetro nomeado como title.

Na linha 26 temos uma variável do tipo String chamada title e além do tipo dela está falando ser uma variável final, ou seja, o valor de dentro dela não será alterado em nenhum lugar no código. Desse jeito a classe tem uma dependência toda vez, que for chamada em qualquer outro lugar.

Na linha 28 temos um @override,eles permitem sobrescrever alguns métodos e na linha abaixo o que pode ser sobrescrito ou reconstruído caso necessite.

Logo após, na linha 32 temos outra classe que está estendendo o tipo necessário para a função createState. Nessa classe será onde estará todo o layout da tela.

Na linha 33 temos uma nova variável do tipo inteira chamada _counter que está sendo inicializada com o valor zero. Veja que ela está diferente em alguns aspectos, ela não tem o final antes, porque ela será a variável responsável para saber quantas vezes o botão foi clicado na tela e ela tem uma underline antes do nome dela, isso serve para deixar essa variável privada a classe, ou seja, somente essa classe conseguirá ver e acessar o conteúdo dessa variável.

Entre a linha 35 a 39 temos um função _incrementCounter que essa função é responsável por aumentar o contador de vezes que o botão foi apertado. Podemos observar que assim como falado sobre as variáveis nas funções temos um underline antes do nome da função e isso significa o mesmo, assim somente essa classe consegue ver e acessar essa função. Na função temos outra função sendo chamada que é a função setState essa função toda vez, que for chamada reconstruirá toda a classe _MyHomePageState que é toda nossa tela.

A partir da linha 41 é onde temos os Widgets responsáveis pela estrutura e design da tela. Mas o que seria esses Widgets. Podemos pensar que widgets são como peças de lego, ou seja, são peças fixas que podemos encaixar umas nas outras, e assim como as peças de legos que tem tamanho, cor e formas diferentes os widgets funcionam assim.

Um dos primeiros widgets usados para a construção da tela é o Scaffold, fundamental por trazer a estrutura básica de uma tela de aplicativo. Dentro dela temos alguns parâmetros que estão sendo usados para a construção da tela como a AppBar, Body e FloatingActionButton.

A AppBar seria como o cabeçalho da tela, onde geralmente ficam alguns botões de ações como voltar por exemplo, na figura 31 está indicado com um contorno laranja. Vemos nesse exemplo de código que a appBar tem dois parâmetros, um title que tem um widget chamado Text que recebe a variável dentro dele para mostrar o conteudo da variavel na tela e um backgroundColor que é a cor de fundo que o retângulo terá, neste exemplo ele está pegando a cor primária do tema definido anteriormente.

O Body é o corpo do Widget, é a área central onde o conteúdo principal da tela é exibido e dentro dela pode conter uma variedade de widgets como listas, imagens, formulários, etc. Retratado com um contorno azul na figura 31. Nesse programa vemos mais widgets novos como o Center e Column. O Center vai sempre centralizar tudo na tela tanto em largura como em altura. Agora podemos ver um relacionamento de herança, pois o Center tem como parâmetro um child que nesse exemplo é a nossa Column. Já na nossa Column temos coisas novas como o mainAxisAlignment e children que ambos são parâmetro ele tem.

O mainAxisAlignment permite indicar onde na coluna estará seu conteúdo, por exemplo se mudarmos a linha 50 para “mainAxisAlignment: MainAxisAlignment.start” veremos que todos os textos que estavam no centro agora estão na parte superior(figura 32).

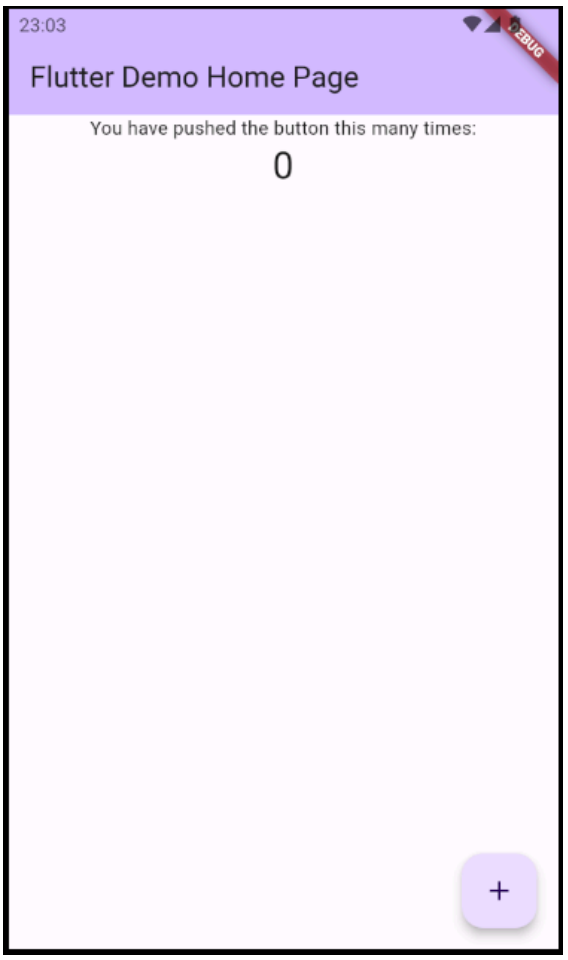
Outra diferença é que a Column tem children ao invés de child, possibilitando colocar uma quantidade finita de widgets. Para essa quantidade de children for infinito a tela precisa de novos widgets de scroll, um exemplo seria colocar a Column como filha de um SingleChildScrollView (figura 33), que esse widget iria colocar scroll na tela somente quando atingir o tamanha máximo da tela.

Figura 31 — Exemplo de uso do SingleChildScrollView

```
48      body: Center(
49        child: SingleChildScrollView(
50          child: Column(
51            mainAxisAlignment: MainAxisAlignment.start,
52            children: <Widget>[
53              const Text(
54                'You have pushed the button this many times:',
55              ), // Text
56              Text(
57                '$_counter',
58                style: Theme.of(context).textTheme.headlineMedium,
59              ), // Text
60            ], // <Widget>[]
61          ), // Column
62        ), // SingleChildScrollView
63      ), // Center
```

Fonte: Elaborado pelo autor (2024).

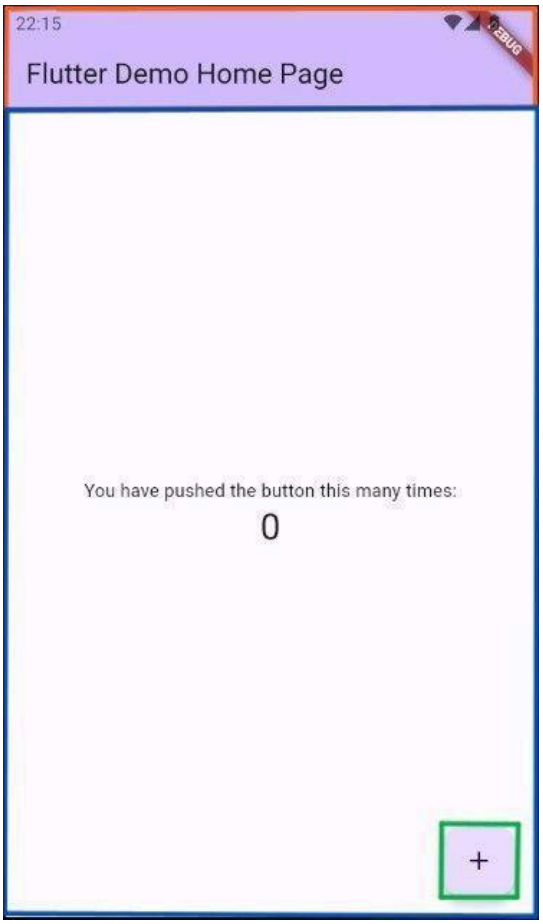
Figura 32 — Tela mudando o mainAxisAlignment



Fonte: Elaborado pelo autor (2024).

E também o FloatingActionButton, sendo um tipo de botão que representa uma ação para a tela principal que pode ser colocado tanto do lado direito como vemos quanto do lado esquerdo. Representado com um contorno verde na figura 31. Podemos ver no código que o widget FloatingActionButton tem três parâmetros sendo usados, onPressed, tooltip e um child. O onPressed é passado a função de incrementar mais um ao contador, ou seja, ao pressionado chamará a função. O tooltip diz o que o botão faz e seu child tem outro widget que é o Icon, responsável em colocar o símbolo de + no botão.

Figura 33 — Tela do celular dividindo AppBar, Body e FloatingActionButton



Fonte: Elaborado pelo autor (2024).

3. CONSIDERAÇÕES FINAIS

Ao longo deste trabalho, exploramos os passos iniciais para instalação e configuração do Flutter, um framework de desenvolvimento de aplicações multiplataforma, e destacamos as vantagens associadas ao seu uso. O Flutter, desenvolvido pelo Google, oferece uma abordagem inovadora e eficiente para criar interfaces de usuário ricas e responsivas, compartilhando uma base de código entre as plataformas Android, iOS e, cada vez mais, web e desktop.

Ao considerar a experiência proporcionada durante a instalação e os primeiros passos com o Flutter, fica evidente o quão acessível e amigável é o ambiente de desenvolvimento que a ferramenta proporciona. A simplicidade da configuração, juntamente com o poderoso sistema de widgets e o recurso de Hot Reload, acelera significativamente o ciclo de desenvolvimento, permitindo aos desenvolvedores visualizar rapidamente as alterações e aprimorar a qualidade do código.

Além disso, a capacidade do Flutter de criar interfaces de usuário consistentes e visualmente atraentes em diferentes plataformas contribui para a eficiência do desenvolvimento. A reutilização de código é maximizada, reduzindo a complexidade e os esforços necessários para manter aplicações nativas separadas para cada plataforma.

A comunidade ativa em torno do Flutter, a extensa documentação e a crescente adoção por grandes empresas reforçam sua posição como uma escolha valiosa para o desenvolvimento de aplicativos multiplataforma. A flexibilidade e versatilidade do Flutter não apenas economizam tempo e recursos, mas também proporcionam aos desenvolvedores a oportunidade de oferecer experiências de usuário excepcionais em uma variedade de dispositivos.

Diante disso, diante disso, que o Flutter se destaca como uma ferramenta eficaz e promissora para desenvolvedores que buscam simplificar e otimizar o processo de desenvolvimento de aplicativos, proporcionando benefícios tangíveis, desde a instalação até os estágios iniciais do desenvolvimento. À medida que o Flutter continuar a evoluir, seu impacto positivo no cenário de desenvolvimento de software multiplataforma é inegável, tornando-o uma escolha notável para o presente e o futuro.

REFERÊNCIAS

ALURA. Disponível em: https://cursos.alura.com.br/course/flutter-widgets-stateless-stateful-imagens-animacoes/. Acesso em: 26 jan. 2024.

CHAVAN. Balram Morsing. Object Oriented Programming with Angular. BPB Publications, 2020

FLUTTER Community. Disponível em: https://github.com/fluttercommunity/community. Acesso em: 26 jan. 2024.

FLUTTER. Disponível em: https://github.com/flutter/flutter. Acesso em: 26 jan. 2024.

FRAMEWORK Chocolatey. Disponível em: https://chocolatey.org/. Acesso em: 26 jan. 2024.

FRAMEWORK Flutter. Disponível em: https://flutter.dev. Acesso em: 26 jan. 2024.

SISTEMA Dart. Disponível em: https://dart.dev/. Acesso em: 26 jan. 2024.

ZAMMETTI, Frank. Flutter na prática. Editora Novatec, 2020. Acessado em 26 jan. 2024