

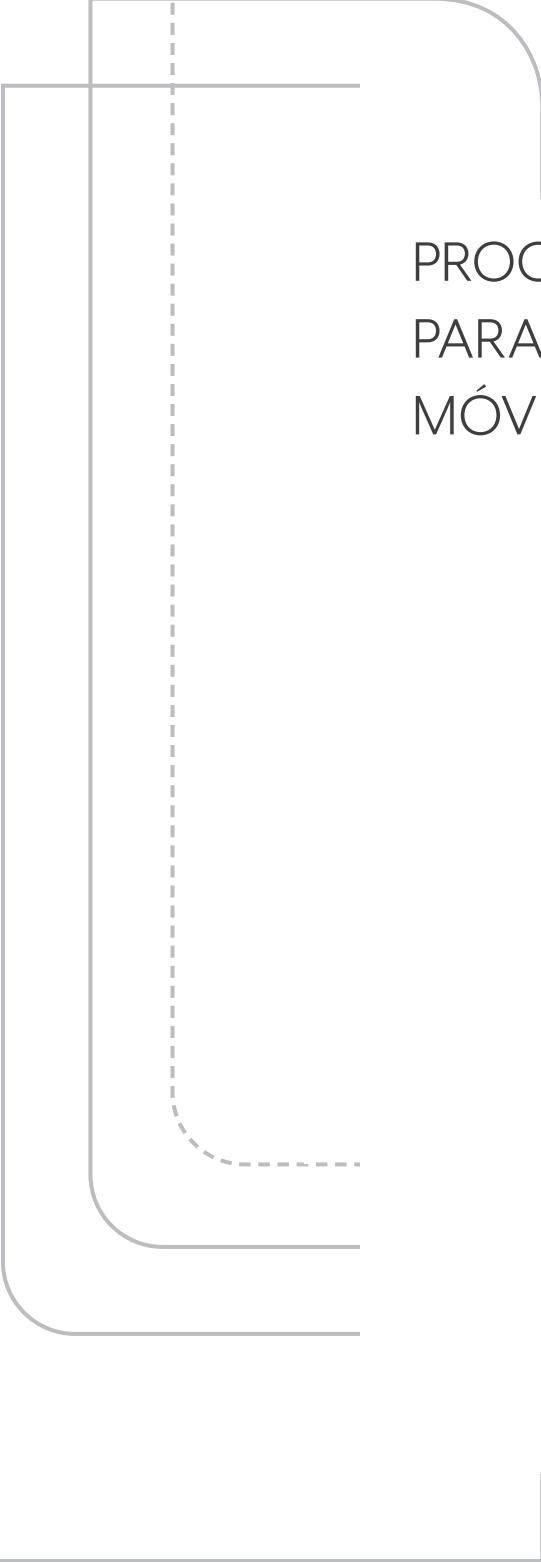


**Informação e  
Comunicação**

# PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

---

**SENAI-SP editora**



# PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

#### **DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)**

---

Vieira, Daniel Filipe

Programação para dispositivos móveis / Daniel Filipe Vieira. – 1. ed. – São Paulo : SENAI-SP editora, 2025.

488 p. : 18x25 cm

Inclui bibliografia.

ISBN 978-65-5912-080-2

1. Programação mobile
2. Desenvolvimento de aplicativos móveis
3. Flutter
4. Linguagem Dart
5. Android Studio
1. Título.

CDD: 004.165

---

Índice para o catálogo sistemático:

1. Programação para dispositivos específicos (aplicativos móveis) 004.165

Bibliotecário responsável: Luiz Valter Vasconcelos Júnior CRB-8 8446O

---

#### **SENAI-SP Editora**

Avenida Paulista, 1.313, 6º andar, 01311-923, São Paulo – SP

[editora@sesisenaisp.org.br](mailto:editora@sesisenaisp.org.br) | [www.senaispeditora.com.br](http://www.senaispeditora.com.br)

Informação e  
Comunicação

# PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS



## **SENAI-SP editora**

### **DEPARTAMENTO REGIONAL DE SÃO PAULO**

Josué Christiano Gomes da Silva

#### **Presidente**

Ricardo Figueiredo Terra

#### **Diretor Regional**

Marta Alves Petti

#### **Diretoria Corporativa SESI-SP e SENAI-SP**

Marcello Luiz de Souza Junior

#### **Gerência de Relações com o Mercado**

Daniel da Silva Motta

#### **Gerência de Inovação e Tecnologia**

Cássia Regina Souza da Cruz

#### **Gerência de Educação**

Daniel Filipe Vieira

#### **Elaboração de conteúdo**

Lucas Tadeu Monteiro Guedes Fernandes Salomão

#### **Revisão técnica**

Material didático utilizado nos cursos do SENAI-SP.

Adilson Castro de Souza Rocha

#### **Gerência editorial**

SENAI-SP Editora

#### **Edição**

Mariane Cristina de Oliveira

#### **Análise editorial**

Globaltec

#### **Preparação de texto**

Palimpsestos

#### **Revisão**

Caio Marques Rodrigues

Camila Ciarini Dias

Marisa Daniela Pereira Araújo

#### **Projeto gráfico e capa**

Rafael Zemantauskas

#### **Coordenação de produção gráfica**

Aline Navarro

José Augusto Moura do Amaral

Valquíria Chagas

#### **Produção gráfica**

Palimpsestos

#### **Diagramação**

Edilza Alves Leite

Viviane Medeiros de Souza Guedes

#### **Direitos autorais**

### **© SENAI-SP Editora, 2025**

A SENAI-SP Editora empenhou-se em identificar e contatar todos os responsáveis pelos direitos autorais deste livro. Se porventura for constatada omissão na identificação de algum material, dispomos-nos a efetuar, futuramente, os possíveis acertos.

# APRESENTAÇÃO

A evolução dos dispositivos móveis transformou profundamente a forma como interagimos com a tecnologia, exigindo dos profissionais da área de desenvolvimento um domínio cada vez mais amplo de ferramentas, linguagens e boas práticas. Este livro foi concebido para oferecer uma introdução sólida e progressiva ao universo da programação mobile, abordando tanto os fundamentos dos dispositivos móveis e seus ambientes de desenvolvimento como a criação de interfaces interativas, consumo de APIs RESTful, persistência de dados e integração com recursos de hardware como GPS, câmera e bluetooth. Com foco prático e didático, o conteúdo é ideal para estudantes, iniciantes e profissionais que desejam se aprofundar na criação de aplicativos modernos e funcionais.

Ao longo dos capítulos, o leitor será guiado por uma jornada estruturada que inclui a configuração de ferramentas como Git, Flutter e Android Studio, o aprendizado da linguagem Dart e seus principais conceitos, além da aplicação de paradigmas de programação orientada a objetos. A obra também explora aspectos essenciais da experiência do usuário, como layouts de tela, navegação entre telas e tratamento gestual, culminando na publicação do aplicativo. Com exercícios práticos, este livro proporciona uma base completa para quem deseja ingressar ou se especializar no desenvolvimento mobile, combinando teoria, prática e inovação.

# SUMÁRIO

## CAPÍTULO 1

<b>DISPOSITIVOS MÓVEIS</b>	<b>13</b>
Introdução	13
Definições	13
Tipos de dados	15
Principais tipos de dados	15
Histórico	17
Tipos de desenvolvimento - nativo e híbrido	21
Características e arquitetura	23
O que é Ionic e quais são suas características	27
Características do Ionic	27
Ambiente de desenvolvimento	31
Instalação e configuração	32

Git e o controle de versão no desenvolvimento	33
Instalação do código Git	36
Android Studio SDK + Emulador	42
Instalação do Android Studio	42
Linguagem Dart	47
Variáveis	49
Operadores relacionais e lógicos	52
Estruturas condicionais	55
Estrutura de repetição	58
Estrutura de dados	62
Listas (List)	62
Funções	74
Funções em aplicações móveis	76
Paradigmas de linguagem de programação	78
Paradigma imperativo	78
Programação orientada a objeto	79
Por que utilizar programação orientada a objeto?	80
Principais elementos da POO	81
Get e Set	90
Conclusão	95

## CAPÍTULO 2

<b>CRIAÇÃO DE INTERFACE</b>	<b>97</b>
Introdução	97
Widget tree (árvore de widgets)	99
Container	99
Relações de parentesco	99
Widgets básicos	100
Widgets de layout	101
Widgets interativos	102
Widgets de apresentação (Display Widgets)	111

Widgets de navegação (Navigation Widgets)	116
Layouts de tela e estrutura	128
Criando emulador para executar aplicativos	149
Gerenciadores de estado	154
StatefulWidget	155
Provider	157
Riverpod	160
BLoC (Business Logic Component)	161
GetX	162
Recursos e interfaces	170
Componentes de tela: menus, diálogos e barra de ação	174
Controle dos elementos na tela	184
Tratamento de eventos e exceções	189
Tratamento de exceções	190
Manipulação de listas na interface	192
Entrada, processamento e saída de dados	200
Navegação entre telas e passagem de parâmetros	204
Gerenciamento de dependências	214
Conclusão	217

### CAPÍTULO 3

<b>APIs E CONSUMO DE RESTFUL WEB SERVICE</b>	<b>219</b>
Introdução	219
Conceitos	220
O que é uma API?	221
O que é um Web Service?	221
Qual é a diferença entre API e Web Service?	221
Como funciona essa comunicação?	222
Por que o HTTPS é importante para APIs que processam dados sensíveis?	224

Requisições HTTP	226
GET	226
POST	226
PUT	228
PATCH	229
DELETE	229
Requisição do tipo GET	230
Requisição do tipo POST	253
Requisição do tipo PUT	257
Diferenças entre PUT e PATCH	260
Requisição do tipo DELETE	263
Manipulação de dados	266
JSON	266
XML	271
Requisições assíncronas	274
Criando aplicativo para realizar GET, POST, PUT, DELETE	279
Node.js	285
JSON Server	285
Exemplo de aplicativo completo com requisições	
GET, POST, PUT e DELETE	290
Conclusão	307

## CAPÍTULO 4

<b>PERSISTÊNCIA DE DADOS EM APLICATIVOS MÓVEIS</b>	<b>309</b>
Introdução	309
Armazenamento de dados	310
SharedPreferences	313
Sqflite	318
Hive	328
Drift	339
Firebase	347



Persistência local x Persistência nuvem	383
Persistência local	383
Persistência na nuvem	384
Abordagem híbrida	385
Conclusão	389

## CAPÍTULO 5

<b>RECURSOS DE HARDWARE</b>	<b>391</b>
Introdução	391
Câmera	392
Áudio	396
Localização e GPS	407
Desenvolvimento do aplicativo ONG Maps	408
Acelerômetro	437
Funcionamento	437
Aplicações do acelerômetro em smartphones	438
Limitações do acelerômetro	448
Diferenças entre acelerômetro e giroscópio	448
O futuro do acelerômetro	448
Wi-Fi	449
Bluetooth	453
Aplicações do bluetooth em Flutter	453
Principais bibliotecas para bluetooth no Flutter	454
Conclusão	467

## CAPÍTULO 6

<b>PUBLICAÇÃO DO APLICATIVO</b>	<b>469</b>
Introdução	469
Como alterar o ícone do aplicativo?	470
Compilação	474

Configuração prévia	474
Modos de compilação	475
Release	477
<b>Distribuição</b>	<b>478</b>
Distribuição no Android	478
Distribuição no iOS	479
Conclusão	480
<b>REFERÊNCIAS</b>	<b>481</b>
<b>SOBRE O ELABORADOR</b>	<b>485</b>

## CAPÍTULO 1

# DISPOSITIVOS MÓVEIS

## Introdução

Neste capítulo, abordaremos o histórico dos dispositivos móveis, suas características, quais recursos são necessários para criar aplicativos a serem usados nos dispositivos móveis, qual deve ser a configuração do ambiente de desenvolvimento e qual framework deve ser utilizado nesse processo. Também exploraremos a sintaxe da linguagem Dart, os tipos de dados, as estruturas condicionais e de repetição, os paradigmas de programação, além de conceitos como classe, herança, polimorfismo e classes abstratas.

## Definições

Framework é um conjunto estruturado de ferramentas, bibliotecas, padrões e componentes que oferece uma base para o desenvolvimento de aplicações.

Ele serve como um “esqueleto” ou infraestrutura para os desenvolvedores usarem e estender para construir software de forma mais rápida e consistente.

A tabela a seguir apresenta alguns tipos de framework.

**Tabela 1.1 – Exemplos de framework**

<b>Web</b>	Django (Python), Laravel (PHP), Spring (Java), Ruby on Rails (Ruby).
<b>Mobile</b>	Flutter (Dart), React Native (JavaScript), SwiftUI (Swift).
<b>Frontend</b>	Angular, Vue.js, React (considerado uma biblioteca, mas usado como framework em alguns casos).
<b>Backend</b>	Express.js (Node.js), ASP.NET (C#).

Para entendermos melhor o conceito de framework, vale ressaltar as diferenças entre eles e as bibliotecas. **Biblioteca** é o conjunto de funções acionadas conforme a necessidade, controlando o fluxo do programa; por exemplo, NumPy em Python. O **framework**, por sua vez, define o fluxo do programa, controlando a execução de certas partes e chamando o código do desenvolvedor; por exemplo, Django, que já oferece estrutura para o desenvolvimento web.

Agora vamos conhecer as vantagens e as desvantagens do framework.

**Tabela 1.2 – Vantagens e desvantagens do framework**

Vantagens	Desvantagens
<b>Produtividade:</b> acelera o desenvolvimento ao reutilizar soluções prontas.	<b>Curva de aprendizado:</b> pode ser difícil compreender frameworks mais complexos.
<b>Organização:</b> segue padrões predefinidos que facilitam a manutenção do código.	<b>Rigidez:</b> impõe certas regras, o que pode limitar a flexibilidade do desenvolvedor em casos específicos.
<b>Comunidade:</b> frameworks populares contam com grande suporte e documentação.	

Um framework é, portanto, uma ferramenta essencial para o desenvolvimento moderno, facilitando o trabalho do desenvolvedor ao mesmo tempo que garante eficiência e consistência.

# Tipos de dados

Os **tipos de dados** são classificações que definem quais valores uma variável pode armazenar e quais operações podem ser realizadas com esses valores. Eles são fundamentais em qualquer linguagem de programação, pois ajudam a estruturar e organizar os dados de forma eficiente.

## Principais tipos de dados

Primitivos (básicos)

Estes são os **tipos de dados fundamentais** disponíveis na maioria das linguagens:

- **Numéricos**
  - **Inteiros (int)**: números inteiros positivos ou negativos sem ponto decimal.  
Exemplo: -5, 0, 42.
  - **Ponto flutuante (float ou double)**: números com casas decimais.  
Exemplo: 3.14, -0.001.
  - **Decimal (ou precisão alta)**: números com alta precisão para cálculos financeiros.  
Exemplo: 1.0001.
- **Caractere (char)**

Representa um único caractere Unicode.  
Exemplo: 'a', '1', '?'.
- **String**

Conjunto de caracteres formando textos.  
Exemplo: "Olá, mundo!".

- **Booleano (bool)**

Representa valores lógicos: True ou False.

Exemplo: True.

## Estruturados

São usados para agrupar vários valores:

- **Listas (arrays ou vetores)**

Coleções ordenadas de elementos, geralmente do mesmo tipo.

Exemplo: [1, 2, 3] ou [“a”, “b”, “c”].

- **Tuplas**

Coleções ordenadas e imutáveis de elementos.

Exemplo: (1, “a”, True).

- **Conjuntos (sets)**

Coleções não ordenadas de elementos únicos.

Exemplo: {1, 2, 3}.

- **Dicionários (ou mapas)**

Coleções de pares chave-valor.

Exemplo: {“nome”: “Daniel”, “idade”: 30}.

## Abstratos

Definidos pelo programador e baseados em tipos primitivos ou estruturados:

- **classes** – criadas com a programação orientada a objetos;
- **interfaces** – definem o comportamento esperado de uma classe;
- **enums** – conjuntos fixos de constantes; por exemplo: enum Dia { Segunda, Terça, Quarta }.

Específicos de cada linguagem

Algumas linguagens possuem tipos de dados específicos:

- **Python**

dict, list, tuple, set.

- **JavaScript**

undefined, null, symbol.

- **SQL**

VARCHAR, INTEGER, DATE.

Conversões entre tipos (casting)

- **Implícito:** a linguagem converte automaticamente (exemplo: int para float).
- **Explícito:** o programador força a conversão (exemplo: float(5) em Python).

#### IMPORTANTE

É importante seguir algumas boas práticas com relação aos tipos de dados:

- escolher tipos que otimizem o uso de memória e desempenho;
- utilizar nomes descritivos para variáveis de tipos complexos;
- certificar-se de que os tipos sejam compatíveis ao realizar operações.

## Histórico

Embora os dispositivos móveis sejam itens populares atualmente, é importante conhecer quando surgiram e sua evolução até os dias atuais.

O smartphone certamente é o dispositivo móvel mais conhecido e utilizado; porém, laptops, tablets e leitores eletrônicos (e-readers) também são dispositivos móveis.

Mas, afinal, o que é um smartphone, como surgiu e como foi sua evolução? Um smartphone é um dispositivo móvel com um sistema operacional que combina funcionalidades de um telefone celular com as de um computador, permitindo não apenas chamadas e mensagens, mas também acesso à internet, uso de aplicativos, câmera e GPS, entre outros recursos.

### SAIBA MAIS

Para conhecer mais a respeito do surgimento dos smartphones, acesse o link ou o QR Code indicados neste boxe. Disponível em: <https://www.techtudo.com.br/guia/2023/03/o-que-e-smartphone-veja-o-que-significa-e-quem-inventou-edmobile.ghtml>. Acesso em: 28 jul. 2025.



A figura a seguir apresenta a evolução do telefone celular entre 1983 e 2024.



**Figura 1.1 – Evolução do telefone celular.**

**1983 – O primeiro telefone móvel chega ao mercado:** Em 1983, a Motorola revolucionou a comunicação ao lançar o DynaTAC 8000x, um

aparelho robusto, com 33 cm de altura e pesando 800 gramas. Sua autonomia de bateria era de apenas uma hora de conversa, e seu custo de quase 4 mil dólares o tornava inacessível para muitos. Mesmo assim, ele marcou o início de uma nova era.

**1989 – A revolução do flip:** O MicroTAC 9800X, lançado em 1989, trouxe um design inovador com sua tampa flip, tornando os celulares mais compactos e práticos. Essa inovação abriu caminho para os celulares de bolso.

**1994 – Telas sensíveis ao toque:** O IBM Simon foi pioneiro ao introduzir telas sensíveis ao toque e funcionalidades como calendário e bloco de notas. Apesar da visão futurista, seu alto custo e baixa aceitação limitaram sua popularidade.

**1996 – Primeiros passos para a internet móvel:** O Nokia 9000 Communicator foi o primeiro celular a oferecer acesso à internet, ainda que de forma limitada e cara. Seu design lembrava um pequeno notebook, antecipando funcionalidades dos smartphones.

**1998 – Telas coloridas chegam aos celulares:** Com o lançamento do Siemens S10, em 1998, as telas coloridas fizeram sua estreia. A Nokia rapidamente superou essa inovação com o modelo 9210, introduzindo telas de alta qualidade.

**2001 – A era das câmeras:** O Sharp J-SH04 trouxe a câmera integrada aos celulares, permitindo tirar fotos e compartilhá-las instantaneamente. Foi o primeiro passo para transformar os celulares em ferramentas multimídia.

**2007 – O iPhone e a revolução dos smartphones:** O lançamento do iPhone 2G redefiniu o mercado de celulares. Sua interface totalmente touchscreen, sem teclados físicos, e o sistema iOS trouxeram os aplicativos ao centro da experiência do usuário, criando o ecossistema que hoje domina o mercado.

**2008 – O Android surge como alternativa:** O HTC Dream introduziu o Android, um sistema operacional flexível e acessível a diversos fabricantes de celular. Essa abertura acelerou a adoção de smartphones em todo o mundo.

**2010 – Tablets e o avanço dos smartphones:** Com o iPad e melhorias significativas nos smartphones, como câmeras aprimoradas e processadores mais rápidos, o mercado de dispositivos móveis se consolidou como indispensável para o dia a dia.

**2013 – Assistentes virtuais redefinem a interação:** A Siri, seguida por Google Assistant e Alexa, transformou os smartphones em assistentes pessoais, simplificando o acesso a informações e automações.

**2016 – Pagamentos móveis se popularizam:** Serviços como Apple Pay e Google Pay tornaram os smartphones também carteiras digitais, trazendo mais conveniência e segurança às transações.

**2019 – 5G e smartphones dobráveis:** A chegada do 5G trouxe conectividade ultrarrápida, enquanto os smartphones dobráveis, como o Galaxy Fold, abriram novas possibilidades de design.

**2020 e além – IA e integração total:** A inteligência artificial consolidou os smartphones como ferramentas essenciais para produtividade, saúde e entretenimento. Recursos como câmeras avançadas com IA, reconhecimento facial e assistentes de voz redefiniram o que esperamos desses dispositivos.

**2025 – Um futuro promissor:** Hoje, os smartphones são assistentes pessoais completos, essenciais para nossa vida. A evolução não para, e o futuro promete ainda mais inovações que continuarão a transformar nossa relação com a tecnologia.

O Brasil está entre os cinco países com maior número de usuários de smartphones, e, com a crescente oferta de aplicativos para diversas necessidades do dia a dia, o mercado mobile está em constante expansão.

## CURIOSIDADE

### Brasil é o quinto país com maior número de celulares no mundo.

Com 118 milhões de usuários de smartphones, país fica atrás somente da Indonésia, Estados Unidos, Índia e China.

Para entender melhor essa classificação, veja o texto completo no link disponível em: <https://set.org.br/set-news/brasil-ocupa-o-5o-lugar-entre-os-que-tem-maior-numero-de-celulares-no-mundo/>. Acesso em: 28 jul. 2025. O texto pode ser acessado também pelo QR Code.



Entender quais recursos são necessários para o desenvolvimento de aplicativos para dispositivos móveis é de grande importância no atual mercado. Esse assunto é abordado na seção a seguir.

## Tipos de desenvolvimento - nativo e híbrido

Existem duas possibilidades de desenvolvimento para dispositivos móveis, o nativo e o híbrido. Os detalhes de cada tipo de desenvolvimento são apresentados a seguir.

O **desenvolvimento nativo** refere-se à criação de aplicativos exclusivos para um sistema operacional específico, seja iOS ou Android. Uma das principais vantagens do desenvolvimento nativo é a performance aprimorada e a experiência mais fluida para o usuário, já que o aplicativo é otimizado para uma plataforma específica. No entanto a abordagem nativa apresenta desafios em termos de custos e tempo de desenvolvimento, especialmente para aplicativos que precisam ser lançados em múltiplas plataformas.

No modelo de **desenvolvimento híbrido**, o aplicativo é desenvolvido para ser executado tanto no sistema operacional Android quanto no iOS, o que proporciona mais otimização no processo de desenvolvimento, já que o código é compartilhado entre as duas plataformas. No entanto, uma possível

desvantagem do desenvolvimento híbrido é a dificuldade de implementar interações mais complexas ou específicas de cada sistema operacional, o que pode impactar a experiência do usuário em alguns casos.

Quando os aplicativos mobile surgiram, o desenvolvimento era realizado utilizando recursos nativos do sistema operacional. Os aplicativos nativos forneciam acesso total aos recursos de hardware como GPS, câmera e sensores.

Em 2007 a Apple introduziu o iPhone com o sistema iOS que permitia criar aplicativos usando o iOS SDK. Linguagens como Objective C (posteriormente substituída por Swift) tornaram-se padrão para aplicativos iOS. Em 2008 o sistema Android foi lançado com o kit de desenvolvimento de software (Software Kit Development – SDK) que utilizava a linguagem Java, posteriormente substituída pela linguagem Kotlin.

Em 2009 surgiu o primeiro framework híbrido, o PhoneGap (que utilizava a tecnologia Adobe Cordova). Ele permitia criar aplicativos móveis utilizando tecnologias web como HTML, CSS e JavaScript, que eram empacotados em contêineres nativos para rodar em dispositivos móveis.

O **desenvolvimento híbrido** surgiu como uma opção para simplificar a criação de aplicativos que funcionassem em várias plataformas com um único código base e, em 2015, houve uma evolução nos frameworks para desenvolvimento híbrido, gerando melhorias significativas ao oferecer componentes de interface otimizados para dispositivos móveis. O Ionic, por exemplo, utilizava o Angular como base integrando o JavaScript com funcionalidades móveis.

Em 2016 o React Native, desenvolvido pelo Facebook, introduziu um novo paradigma ao permitir a criação de aplicativos híbridos com renderização nativa. Utilizava JavaScript e componentes React, e traduzia o código em interfaces nativas, oferecendo um desempenho próximo ao nativo.

Em 2018, o Google lançou o Flutter, um framework para desenvolvimento de aplicativos multiplataforma que utiliza a linguagem Dart. Ele oferece desempenho quase idêntico ao de aplicativos nativos e se destaca

por permitir uma personalização avançada da interface do usuário. Com o Flutter, é possível criar interfaces modernas, dinâmicas e altamente responsivas – conhecidas como “design rico” –, que incluem animações suaves, transições elegantes, tipografia personalizada, efeitos visuais detalhados e componentes gráficos sofisticados. Isso proporciona uma experiência visual de alto nível, semelhante aos melhores apps nativos.

Além disso, o Flutter conta com uma comunidade ativa e engajada, o que facilita o aprendizado e a resolução de problemas. Também possui suporte direto do Google, integração facilitada com APIs e serviços de back-end, é de código aberto e gratuito, garante compatibilidade com futuras atualizações de sistemas e permite a reutilização de grande parte do código entre diferentes plataformas.

## Características e arquitetura

Para decidir qual é a ferramenta mais adequada para o desenvolvimento de aplicativos móveis, é importante analisar as características indicadas na tabela a seguir.

**Tabela 1.3 – Desenvolvimento híbrido versus nativo**

Desenvolvimento híbrido	Desenvolvimento nativo
Economia de tempo e custo: um único código para múltiplas plataformas.	Ideal para aplicativos que exigem desempenho de alta qualidade e acesso amplo a recursos do hardware – por exemplo, jogos, aplicativos de realidade aumentada ou qualquer aplicativo com requisitos gráficos intensivos.
Tem suporte e atualizações simultâneas para todas as plataformas.	
Expansão da base de desenvolvedores: desenvolvedores web podem migrar facilmente para o desenvolvimento móvel.	

## GLOSSÁRIO

### INTEGRATED DEVELOPMENT ENVIRONMENT – IDE

(ambiente de desenvolvimento integrado) é um software que auxilia os programadores a desenvolverem código de forma organizada e eficiente.

Hoje, o desenvolvimento híbrido é uma tendência, especialmente pelo fato de frameworks como Flutter e Reactive Native reduzirem as diferenças de desempenho dos aplicativos híbridos diante dos aplicativos nativos.

Veja, a seguir, os recursos e a **IDE** utilizada para desenvolvimento nativo.

### Para o sistema operacional Android

- Linguagens – Java e Kotlin
  - » Java: uma das linguagens mais populares e robustas, utilizada desde o início do Android;
  - » Kotlin: linguagem moderna, oficializada pelo Google em 2017, projetada para ser mais expressiva, segura e produtiva.
- IDE: Android Studio.
  - » Ferramenta oficial do Google para desenvolvimento Android;
  - » Oferece emuladores, depuração, gerenciamento de recursos, suporte a Gradle (para automação de builds) e design visual para interfaces.

### Para o sistema operacional iOS

- Linguagens – Objective C e Swift
  - » Objective C: uma linguagem mais antiga e amplamente utilizada antes do lançamento do Swift;
  - » Swift: lançada pela Apple em 2014, é moderna, segura, rápida e com sintaxe mais limpa.
- IDE: Xcode
  - » Ferramenta oficial da Apple, disponível exclusivamente para macOS;
  - » Inclui simuladores, editor de interface visual (storyboard) gerenciamento de certificados e ferramentas de depuração.

Os recursos e frameworks utilizados para desenvolvimento híbrido são os indicados a seguir.

### **React Native**

- Criado pelo Facebook;
- Linguagem: JavaScript;
- Arquitetura: usa uma Bridge para traduzir o código JavaScript em elementos nativos, permitindo a interação com recursos do dispositivo.
- **Vantagens**
  - » Comunidade robusta e suporte a bibliotecas amplamente utilizadas no desenvolvimento web;
  - » Facilita o uso de componentes nativos, como câmera e GPS, por meio de plugins.
- **Desvantagens**
  - » dependência da Bridge pode gerar atrasos na interação com o hardware, dependendo da complexidade do aplicativo.

### **Flutter**

- Criado pelo Google;
- Linguagem Dart;
- Arquitetura: gera diretamente widgets nativos, eliminando a necessidade de uma Bridge;
- **Vantagens**
  - » Alto desempenho, comparável ao desenvolvimento nativo, devido à renderização direta;
  - » Design consistente entre plataformas com widgets personalizados e adaptáveis;
  - » Ferramenta de depuração integrada, como o hot reload que acelera o desenvolvimento.

- **Desvantagem:** comunidade menor em comparação ao React Native, porém está em crescimento constante.

## Xamarin

O Xamarin é uma plataforma de desenvolvimento multiplataforma que permite criar aplicativos para Android, iOS e Windows utilizando C# e o ecossistema .NET. Ele foi adquirido pela Microsoft em 2016 e está integrado ao Visual Studio.

- Criado por Nat Friedman e Miguel de Icaza em 2011, foi adquirido pela Microsoft em 2016, conforme já mencionado.
- Linguagens: C# e .NET.
- Arquitetura: usa o Xamarin.Android e Xamarin.iOS para interagir diretamente com as APIs nativas das plataformas.
- Oferece o Xamarin.Forms, que permite criar uma interface de usuário compartilhada para todas as plataformas.
- Integração: totalmente integrado ao ecossistema Microsoft, incluindo Azure e ferramentas de DevOps.
- Acesso a APIs nativas: permite usar APIs nativas das plataformas, garantindo desempenho próximo ao nativo.
- **Vantagens**
  - » Reutilização de código: possibilidade de compartilhar até 90% do código entre plataformas.
  - » Acesso a recursos nativos: interação direta com APIs nativas por meio do Xamarin.Android e Xamarin.iOS.
  - » Ecossistema robusto: integração com o Visual Studio, Azure e ferramentas Microsoft.
  - » Forte desempenho: aplicativos desenvolvidos com Xamarin.Android e Xamarin.iOS têm desempenho próximo ao nativo.

- » Comunidade madura: grande base de desenvolvedores, graças ao uso de C#.

- **Desvantagens**

- » Tamanho do aplicativo: os aplicativos criados podem ser maiores devido à inclusão do framework .NET.
- » Curva de aprendizado: pode ser complexo para quem não está familiarizado com C# ou .NET.
- » Xamarin.Forms limitado: apesar de facilitar a criação de interfaces compartilhadas, possui menos flexibilidade e personalização em comparação ao desenvolvimento nativo.
- » Atualizações lentas: algumas funcionalidades das plataformas podem demorar para serem disponibilizadas no Xamarin.

## O que é Ionic e quais são suas características

Ionic é um framework para desenvolvimento de aplicativos híbrido baseado em tecnologias web como HTML, CSS, JavaScript. Ele permite criar aplicativos para Android, iOS e web utilizando um único código base.

### Características do Ionic

- Linguagem: JavaScript (ou Type Script), integrado com frameworks como Angular, React ou Vue.js.
- Arquitetura: utiliza WebViews para renderizar a interface, encapsulando o aplicativo em contêineres nativos.
- Complementado pelo Capacitor, que permite acessar recursos nativos do dispositivo.

- Cross-platform: permite desenvolver aplicativos para Android, iOS e Progressive Web Apps (PWA).
- Componentes prontos: biblioteca extensa de UI pré-desenvolvida com design que segue as diretrizes do Material de Design Android e Cupertino (iOS).

- **Vantagens**

- » Familiaridade com tecnologias web: desenvolvedores com experiência em web (HTML, CSS, JavaScript) podem facilmente criar aplicativos.
- » Multiplataforma: criação simultânea de aplicativos móveis e PWAs.
- » Grande biblioteca de UI: componentes pré-desenvolvidos que otimizam o design e aceleram o desenvolvimento.
- » Flexibilidade com frameworks: suporte a Angular, React e Vue.js para diferentes preferências de desenvolvimento.
- » Ecossistema Capacitor: acesso a APIs nativas de forma mais eficiente.

- **Desvantagens**

- » Desempenho: aplicativos híbridos baseados em WebViews podem ser mais lentos em comparação aos nativos ou a frameworks híbridos, como Flutter.
- » Acesso limitado a recursos avançados: depende do Capacitor ou Cordova para interagir com o hardware, o que pode ser limitado ou mais complexo em dispositivos com requisitos específicos.
- » Experiência de usuário: a interface pode parecer menos integrada ao sistema operacional em comparação com aplicativos nativos.
- » Tamanho do aplicativo: aplicativos híbridos tendem a ser maiores devido à inclusão de bibliotecas web.

A tabela a seguir apresenta um resumo com as características de cada framework para desenvolvimento nativo.

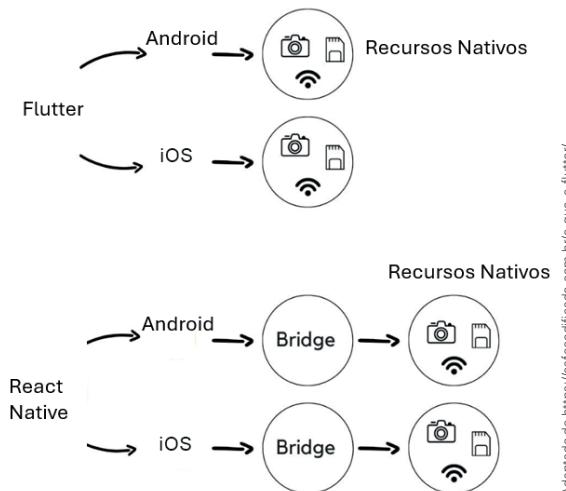
**Tabela 1.4 – Comparação entre os frameworks para desenvolvimento mobile**

Aspecto (modalidade de desenvolvimento)	Nativo	Híbrido (React Native)	Híbrido (Flutter)	Híbrido Xamarin	Híbrido (Ionic)
Linguagem	Swift, Objective C, Kotlin, Java	JavaScript	Dart	C#, .NET	JavaScript/ TypeScript
Acesso ao hardware	Direto	Via Bridge	Direto	Direto	Web-Views com Capacitor
Desempenho	Excelente	Bom, pode ser afetado pela Bridge	Excelente	Próximo ao nativo	Médio, pode ser afetado pelo Webview
Interface	Totalmente nativa	Parcialmente nativa	Renderização própria	Nativa ou compartilhada	Componentes pré-estilizados
Ferramenta de desenvolvimento	Xcode, Android Studio	VSCode, Android Studio	VSCode, Android Studio, IntelliJ	Visual Studio	Ionic Studio ou qualquer editor de código

Para desenvolver aplicativos em Swift, é necessário ter um computador Mac com o Xcode instalado, que é a IDE da Apple. Já o React Native é um framework criado pelo Facebook para o desenvolvimento de aplicativos móveis multiplataforma. Baseado no React, que é um framework JavaScript para desenvolvimento web, o React Native permite criar aplicativos para Android e iOS utilizando o mesmo código em JavaScript.

O desenvolvimento de aplicativos híbridos traz vantagens e desvantagens. Uma das principais vantagens é a economia de tempo e recursos, já que é possível criar um aplicativo para várias plataformas com um único código base, o que reduz os custos de desenvolvimento. Por outro lado, uma desvantagem é a possível perda de desempenho do aplicativo ao acessar

recursos de hardwares do smartphone, como Wi-Fi, câmera, GPS, entre outros. No entanto, no Flutter, essa perda de desempenho não ocorre, já que o código desenvolvido acessa esses recursos de hardware de forma direta, como mostrado na figura a seguir, sem a necessidade de passar por uma ponte (ou Bridge) para interagir com os componentes nativos do dispositivo.



**Figura 1.2 – Frameworks com acesso a recursos nativos do smartphone.**

No React Native, a perda de desempenho ocorre ao acessar os recursos nativos do smartphone devido à utilização da Bridge, que realiza a conexão com os recursos nativos do smartphone. Dessa forma, o React Native pode não ser a melhor escolha para aplicativos que dependem de um uso intenso dos recursos de hardware do smartphone, impactando a performance do aplicativo.

A Bridge em React Native é o mecanismo que permite a interação do JavaScript com os componentes nativos da plataforma (iOS ou Android). Ele funciona como uma ponte bidirecional, pela qual o JavaScript envia comandos para o código nativo e recebe respostas de volta. Essa arquitetura é essencial porque o React Native utiliza duas camadas distintas:

- **Camada JavaScript** – na qual o código da interface do usuário e a lógica do aplicativo são escritos.

- **Camada nativa** – na qual os componentes específicos da plataforma são implementados.

A comunicação entre essas camadas é assíncrona, o que significa que as operações não bloqueiam a execução principal da aplicação.

A Bridge funciona da seguinte maneira:

1. **JavaScript envia mensagens para o código nativo** – Quando precisa acessar funcionalidades nativas, como o GPS ou a câmera, o JavaScript faz essa solicitação por meio de módulos nativos.
2. **Código nativo processa as solicitações** – O código nativo executa as operações necessárias e retorna os resultados ao JavaScript.
3. **Mensagens de volta para o JavaScript** – Os resultados são enviados de volta para o lado JavaScript, que os processa para continuar a execução.

Essa arquitetura oferece flexibilidade e permite que desenvolvedores adicionem funcionalidades nativas personalizadas ao aplicativo.

A seguir, serão apresentados os passos para a configuração do ambiente de desenvolvimento dos aplicativos utilizando o framework Flutter.

## Ambiente de desenvolvimento

Agora serão abordadas as características do framework de desenvolvimento mobile Flutter, bem como os recursos necessários para desenvolvimento como SDK, IDE e Git.

O software Kit Development (SDK) – em português, kit de desenvolvimento de software – é um conjunto de ferramentas fornecidas pelo Google para criar aplicativos multiplataforma de forma eficiente. Essas ferramentas permitem que o desenvolvedor crie interfaces nativas para Android, iOS, Web e Desktop a partir de um único código base escrito em linguagem Dart.

Como já explicado neste capítulo, o Flutter é um kit de desenvolvimento de interface de usuário ou UI toolkit, um kit de ferramentas de interface do usuário de código aberto criado pelo Google em 2017. Como mencionado, o Flutter utiliza a linguagem de programação Dart.

Além disso, ele é uma tecnologia multiplataforma que permite o desenvolvimento de aplicações nativas para Android, iOS, Linux e MacOS a partir de uma única base de código. Com ele, é possível criar apps com desempenho nativo e acesso direto aos recursos do dispositivo, como câmera, Wi-Fi e armazenamento, entre outros. As aplicações desenvolvidas com Flutter tendem a apresentar um desempenho superior em relação ao React Native, pois todo seu código fonte é compilado diretamente para código nativo.

#### SAIBA MAIS

Para mais informações a respeito da documentação Flutter, veja o link disponível em: <https://docs.flutter.dev/reference/tutorials>. Acesso em: 29 jul. 2025. O arquivo também pode ser acessado pelo QR Code.



## Instalação e configuração

Para configurar o ambiente de desenvolvimento e começar a programar aplicativos móveis com o framework Flutter, são necessários os seguintes recursos:

- Git <https://git-scm.com/download/win>;
- SDK Flutter <https://github.com/usuario/repositorio.git>;
- Android Studio SDK + Emulador <https://developer.android.com/studio?hl=pt-br>;
- SDK do Dart <https://dart.dev/get-dart>.

## Git e o controle de versão no desenvolvimento

O Git é um sistema de controle de versão amplamente utilizado por desenvolvedores para gerenciar alterações no código-fonte de projetos. Ele permite que várias pessoas trabalhem no mesmo projeto simultaneamente, mantendo um histórico completo de alterações e facilitando o rastreamento e a reversão de modificações, se necessário.

No desenvolvimento com Flutter, o Git é essencial para gerenciar pacotes, versões e colaborar em projetos. É uma ferramenta indispensável para garantir organização e controle durante o desenvolvimento de software.

### ○ que é controle de versão?

Controle de versão é um sistema que registra e organiza as mudanças feitas em arquivos, desde um arquivo específico até vários arquivos de um projeto. A cada alteração, uma nova versão é criada. Isso possibilita:

- rastrear mudanças realizadas ao longo do tempo;
- trabalhar em equipe de forma eficiente;
- voltar a versões anteriores caso algo dê errado;
- gerenciar dependências e versões estáveis de software.

**Tabela 1.5 – Diferenças entre Git e GitHub**

Git	GitHub
Sistema de controle de versão distribuído.	Plataforma de hospedagem de repositórios Git.
Armazena o histórico de alterações localmente.	Facilita o compartilhamento de código e a colaboração on-line.
Permite criar e gerenciar branches (ramificações de código).	Oferece repositórios remotos (armazenamento na nuvem).
Funciona off-line.	Conecta-se ao Git por meio de comandos como push e pull.

A estrutura básica do Git é constituída pelos seguintes itens:

- Repositório local – armazenamento no computador do desenvolvedor;
- Repositório remoto – armazenamento na nuvem (por exemplo: GitHub, GitLab e Bitbucket);
- Conexão local-remota – feita por meio de comandos como push, que envia alterações para o repositório remoto e pull, que recebe alterações do repositório remoto.

## Comandos Git

Agora serão apresentados os principais comandos para configurar o sistema de versionamento Git com usuário e senha, orientações sobre como criar um repositório, adicionar arquivos e realizar commits, entre outras atividades.

### **Configuração inicial**

Para realizar a configuração de um repositório localmente, utiliza-se o comando **git config**.

```
git config
```

Para configurar opções globais, a fim de identificar o usuário, deve-se utilizar os seguintes comandos:

```
git config --global user.name "Seu Nome"
git config --global user.email "seuemail@exemplo.com"
```

O comando **git config -list** lista todas as configurações atuais do Git.

### **Trabalhando com repositórios**

Para criar ou inicializar um repositório localmente, utiliza-se o comando **git init**, que também cria um repositório vazio no diretório atual.

Para clonar um repositório, deve-se usar o comando **git clone**; ele faz uma cópia de um repositório remoto no computador. Acesse o link disponível em: <https://github.com/>. Acesso em: 29 jul. 2025.

## Gerenciando alterações

Para visualizar as alterações no repositório, utiliza-se o comando **git status**. Esse comando exibe o estado atual do repositório.

Outros comandos e suas funções são indicados a seguir:

**git add** – adiciona arquivos ou alterações para serem confirmados;

**git add arquivo.txt #** – adiciona um arquivo específico;

**git add . #** – adiciona todos os arquivos;

**git commit** – salva as alterações no histórico;

**git commit -m** – “descrição das alterações”.

## Histórico de alterações

Para visualizar o histórico das alterações realizadas nos arquivos do repositório, utiliza-se o comando **git log**. Esse comando exibe o histórico de commits.

## Trabalho em equipe

Para realizar a conexão com um repositório remoto, utiliza-se o **git remote**. Esse comando gerencia conexões com repositórios remotos. Comando: **git remote add origin**. Disponível em: <https://github.com/>. Acesso em: 29 jul. 2025.

Outros comandos e suas funções são indicados a seguir:

**git push** – envia commits para o repositório remoto;

**git push origin main** – envia commits para o repositório remoto na branch main;

**git pull** – atualiza o repositório local com alterações do remoto;

**git pull origin main** – faz o download do código com as alterações realizadas na branch main;

**git fetch** – baixa alterações do remoto sem integrá-las automaticamente;

**git fetch origin** – busca todas as atualizações no repositório remoto – chamado origin – sem alterar a branch atual.

### **Gerenciamento de branchs**

Para realizar o gerenciamento das branchs, utiliza-se o **git branch**. Esse comando lista, cria ou exclui branches, conforme indicado a seguir:

**git branch #** – lista as branches;

**git branch nova-branch #** – cria uma nova branch;

**git checkout** – troca de branch ou restaura arquivos;

**git checkout nova-branch** – sai da branch atual e entra na nova branch;

**git merge** – combina alterações de uma branch com outra;

**git merge nova-branch** – realiza a mesclagem de uma branch nova com a main.

### **Revertendo alterações**

Para reverter uma alteração, utiliza-se o **git reset**. Esse comando desfaz alterações ou commits.

**git reset --hard HEAD~1** # – desfaz o último commit;

**git reset arquivo.txt** # – remove arquivo da área de staging;

**git revert** – reverte um commit criando um novo;

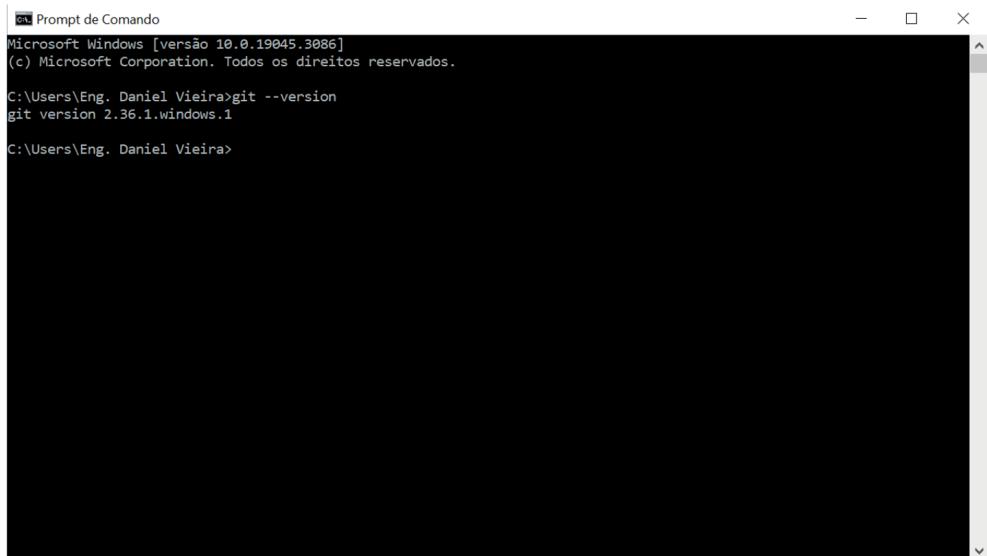
**git revert [ID do commit]** – reverte um commit criado anteriormente através do seu ID.

## **Instalação do código Git**

Para realizar a instalação do Git, observe o passo a passo de instalação do primeiro recurso necessário, que é o sistema de versionamento de código Git.

**1º passo:** verificar se o Git está instalado no computador.

No menu **Iniciar**, digite **cmd**. O prompt de comando será aberto. Então, digite o comando **git --version**.



```
Prompt de Comando
Microsoft Windows [versão 10.0.19045.3086]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Eng. Daniel Vieira>git --version
git version 2.36.1.windows.1

C:\Users\Eng. Daniel Vieira>
```

**Figura 1.3 –** Prompt de comando para verificar versão Git.

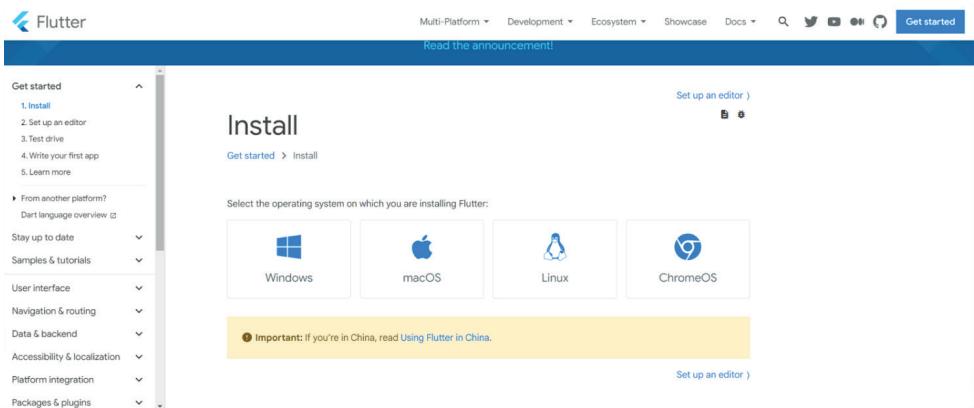
**2º passo:** entrar no site Git e fazer o download para executar a instalação. Link disponível em: <https://git-scm.com/download/win>. Acesso em: 12 jul. 2025.

## SDK do Flutter

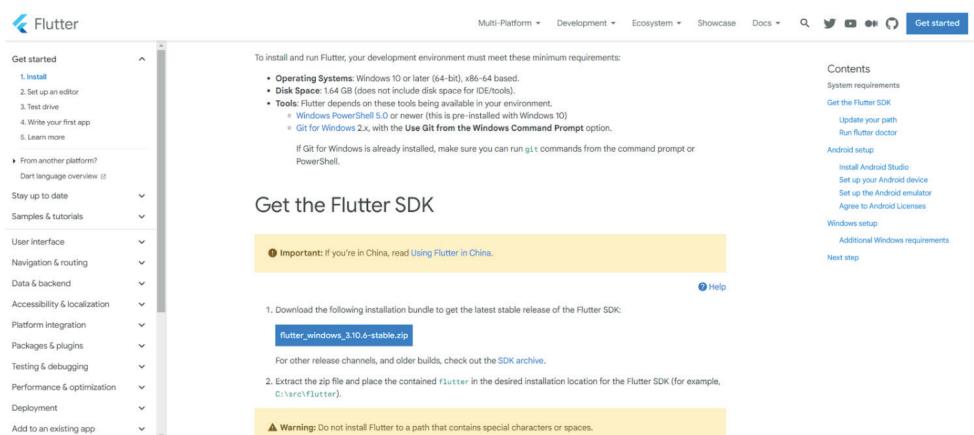
Prosseguindo com o passo a passo, veremos agora como funciona o **Flutter SDK**. Ele é um conjunto de ferramentas desenvolvido pelo Google para criar aplicativos multiplataforma com uma única base de código. Com o Flutter, é possível desenvolver aplicativos para **Android**, **iOS**, **web**, **desktop** e dispositivos embarcados.

**1º passo:** entrar no site Flutter e selecionar o sdk do Flutter de acordo com o sistema operacional: Windows, Linux, Mac.

## 2º passo: baixar a versão do Flutter.



**Figura 1.4 – Site do Flutter.**



**Figura 1.5 – Site do Flutter após a seleção do sistema operacional.**

**3º passo:** após fazer o download, descompactá-lo.

**4º passo:** criar uma pasta no disco C chamada **src**.

Observação: não colocar a pasta **Flutter** no diretório raiz **C**, pois será solicitada permissão de administrador. Para evitar isso, colocar na pasta **src** criada anteriormente.

**5º passo:** copiar a pasta **Flutter** para a pasta **src**.

**6º passo:** entrar na pasta **src > Flutter > bin** e copiar esse caminho (**C:\src\flutter\bin**) para configurar as variáveis de ambiente, conforme indicado a seguir.

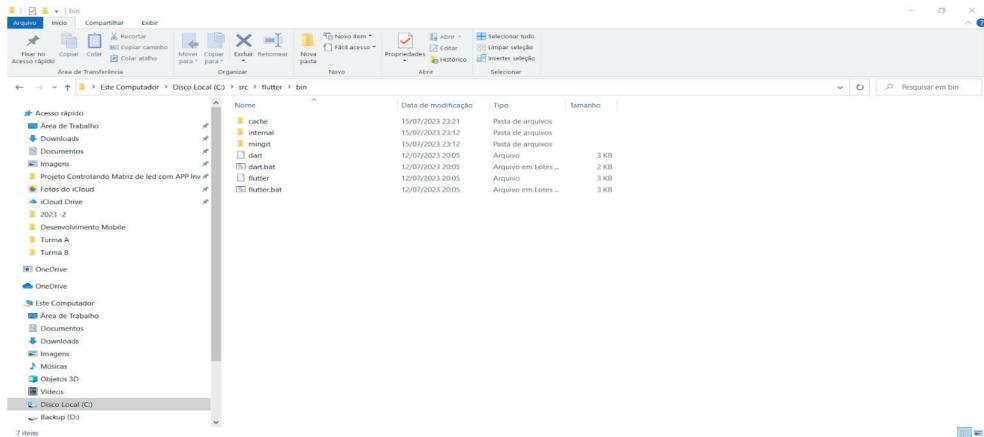


Figura 1.6 – Pasta **src/bin**.

**7º passo:** configurar variáveis de ambiente pelo seguinte caminho:  
Meu Computador > Propriedades > Configurações avançadas do sistema.

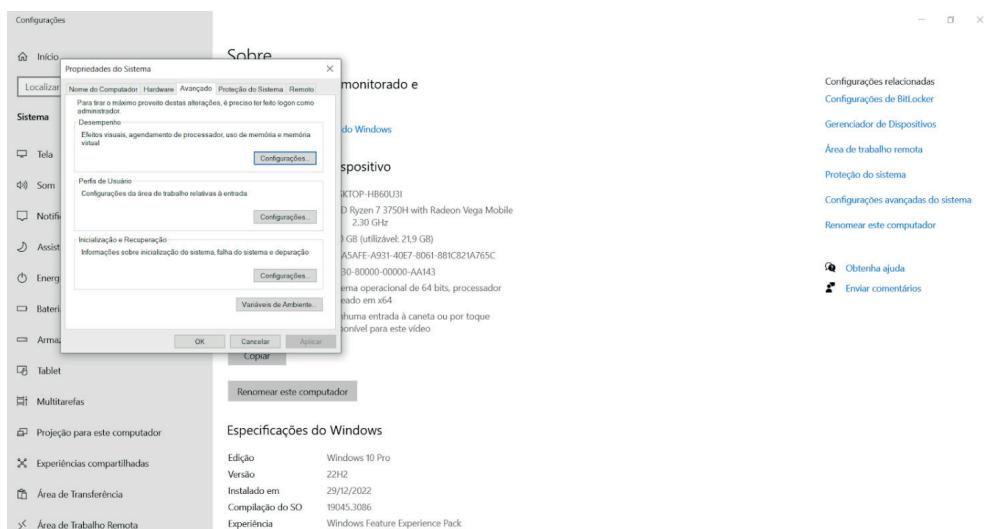


Figura 1.7 – Configurações avançadas do sistema.

**8º passo:** para configurar variáveis de ambiente, clicar em “variáveis de ambiente”, procurar pelo campo **Path** e clicar em **editar**.

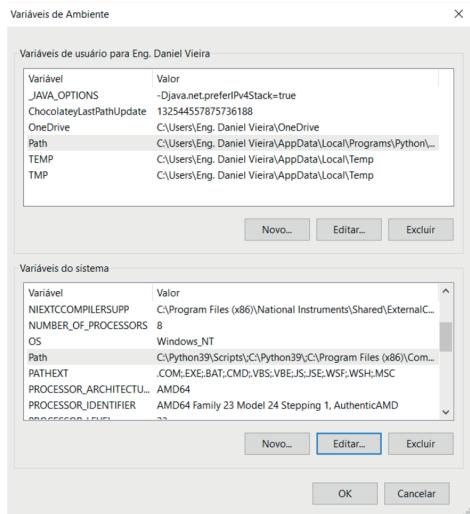


Figura 1.8 – Variáveis de ambiente.

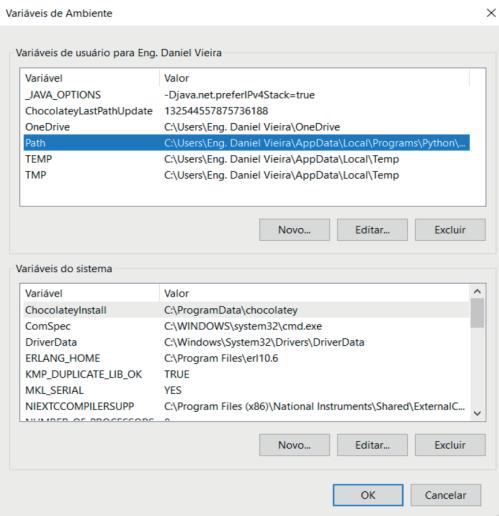


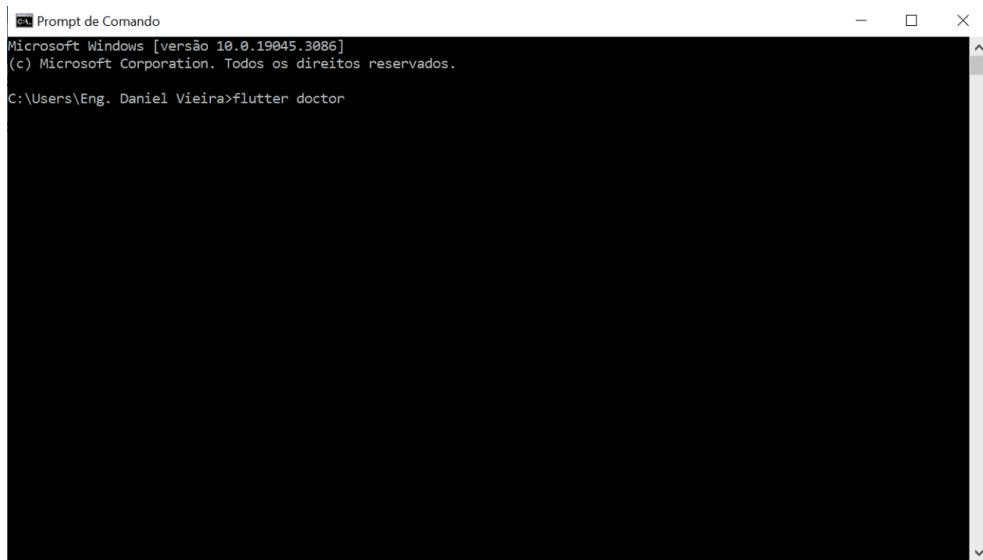
Figura 1.9 – Variáveis de ambiente.

**9º passo:** configurar variáveis de ambiente. Procurar pelo campo **Path**, clicar em **editar** > **Novo** e colar o caminho copiado anteriormente, conforme indicado a seguir: C:\src\flutter\bin. Em seguida, realizar a configuração tanto em “variáveis do usuário” quanto em “variáveis do sistema”.

Procurar pelo campo **Path**, clicar em **editar** > **Novo** e colar o caminho copiado anteriormente (C:\src\flutter\bin), conforme indicado no passo a seguir. Depois, basta clicar em **OK** e fechar as janelas.

**10º passo:** clicar em **Menu > iniciar e cmd**.

**11º passo:** digitar “Flutter Doctor” para verificar quais componentes foram instalados.

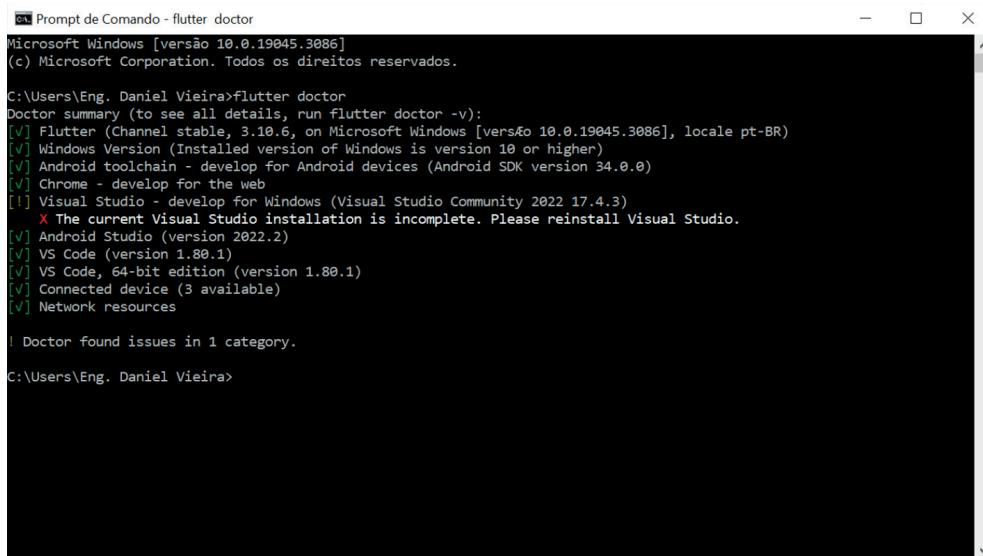


```
cmd Prompt de Comando
Microsoft Windows [versão 10.0.19045.3086]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Eng. Daniel Vieira>flutter doctor
```

**Figura 1.10** – Prompt de comando “Flutter Doctor”.

**12º passo:** se as configurações das variáveis de ambiente foram realizadas com sucesso, aparecerá a tela indicada a seguir.



```
cmd Prompt de Comando - flutter doctor
Microsoft Windows [versão 10.0.19045.3086]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Eng. Daniel Vieira>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.10.6, on Microsoft Windows [versão 10.0.19045.3086], locale pt-BR)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[✓] Chrome - develop for the web
[!] Visual Studio - develop for Windows (Visual Studio Community 2022 17.4.3)
    X The current Visual Studio installation is incomplete. Please reinstall Visual Studio.
[✓] Android Studio (version 2022.2)
[✓] VS Code (version 1.80.1)
[✓] VS Code, 64-bit edition (version 1.80.1)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 1 category.

C:\Users\Eng. Daniel Vieira>
```

**Figura 1.11** – Prompt de comando “Flutter Doctor”.

# Android Studio SDK + Emulador

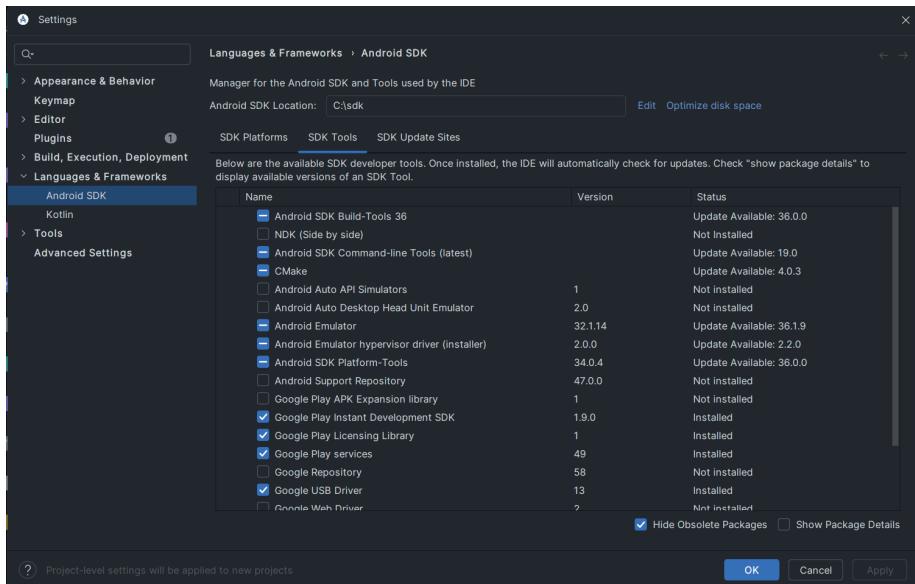
Nesta seção serão abordados o processo de instalação e a configuração do Android Studio.

## Instalação do Android Studio

O passo a passo a seguir mostra o processo de instalação do Android Studio.

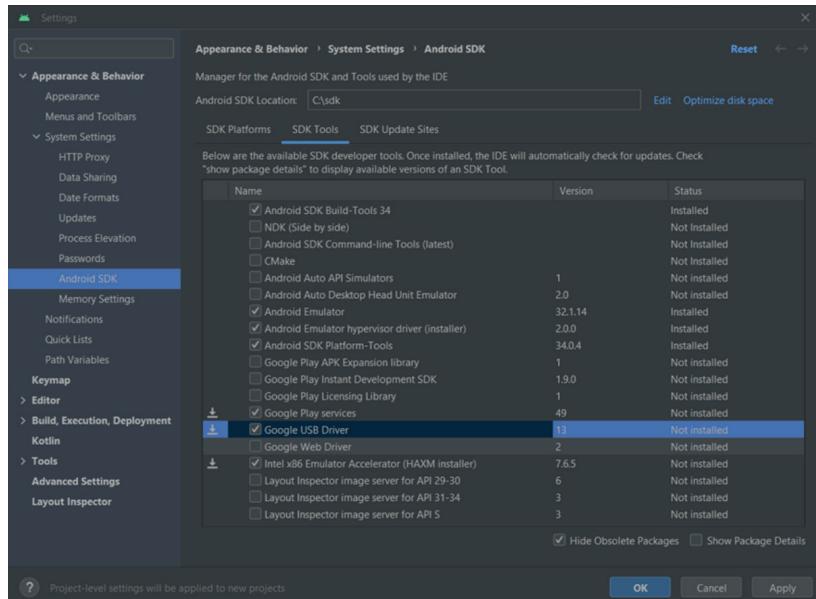
**1º passo:** download do Android Studio disponível em: <https://developer.android.com/studio?hl=pt-br>. Acesso em: 13 jul. 2025.

**2º passo:** instalar o Android Studio.



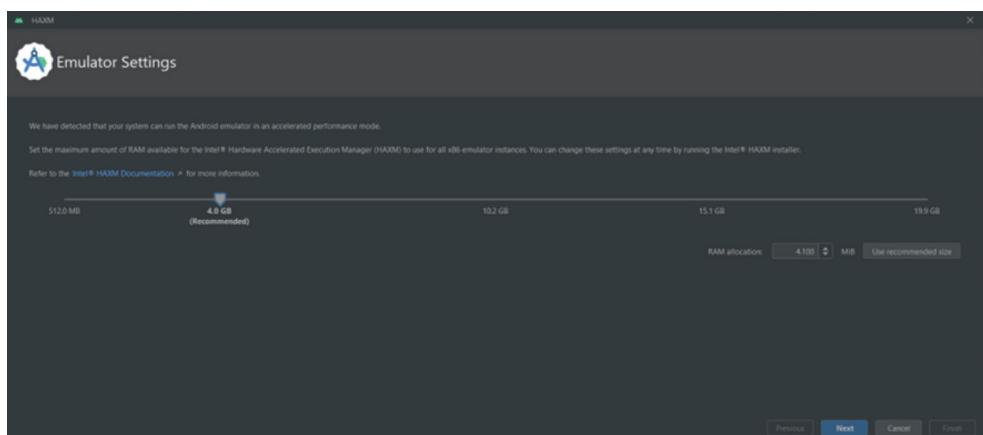
**Figura 1.12 – Instalação do Android Studio.**

### 3º passo: instalar o SDK



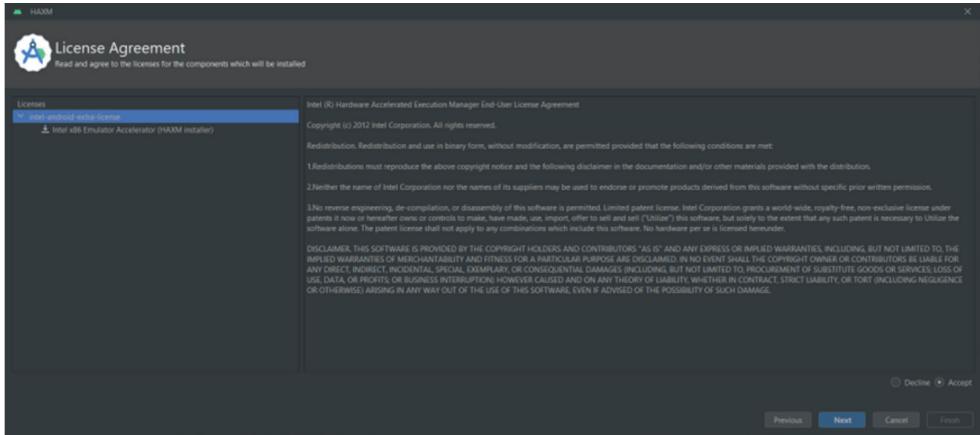
**Figura 1.13 – Instalação do SDK do Android Studio.**

### 4º passo: instalação do emulador.



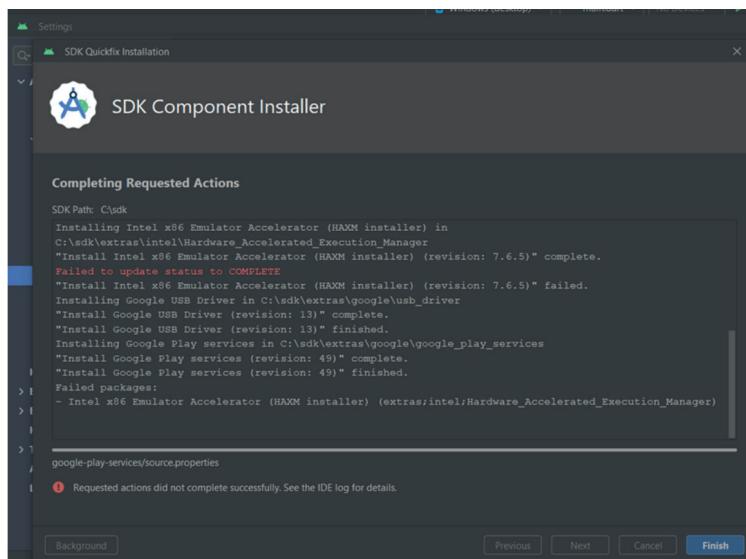
**Figura 1.14 – Configuração dos recursos do emulador do smartphone.**

## 5º passo: concordar com a licença.



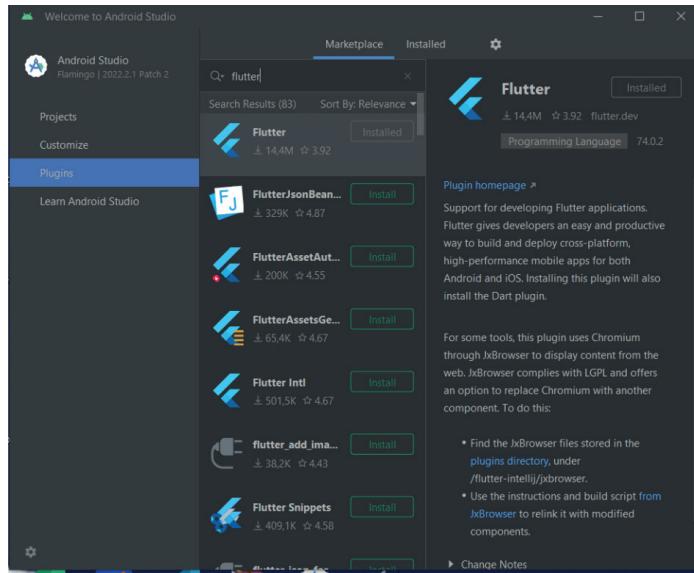
**Figura 1.15 – Aceite das licenças.**

## 6º passo: instalação do emulador.

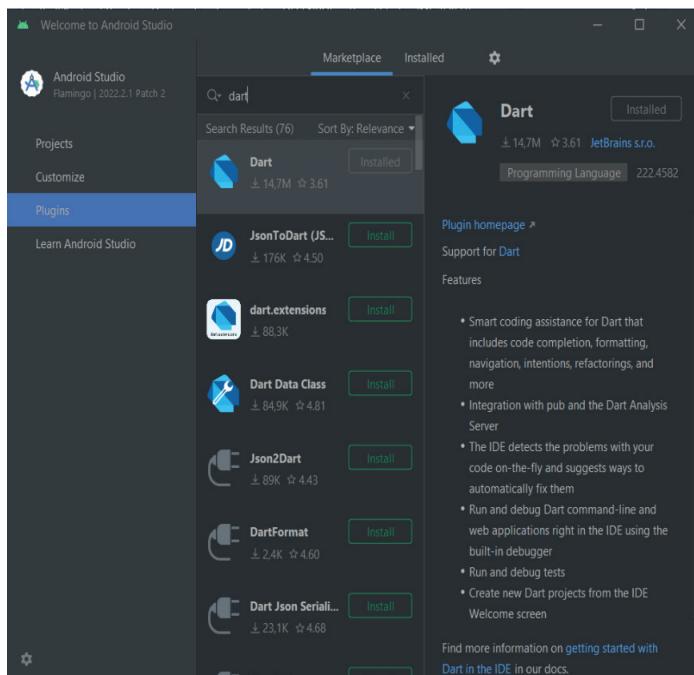


**Figura 1.16 – Instalação do emulador.**

## 7º passo: instalação dos plugins Flutter e Dart.

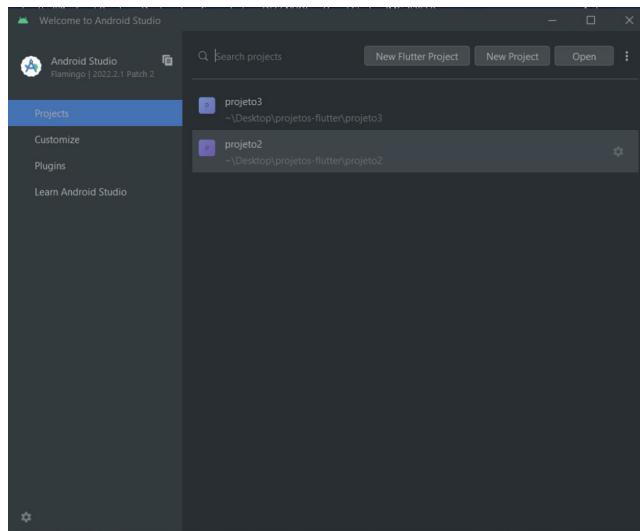


**Figura 1.17 – Instalação do plugin.**



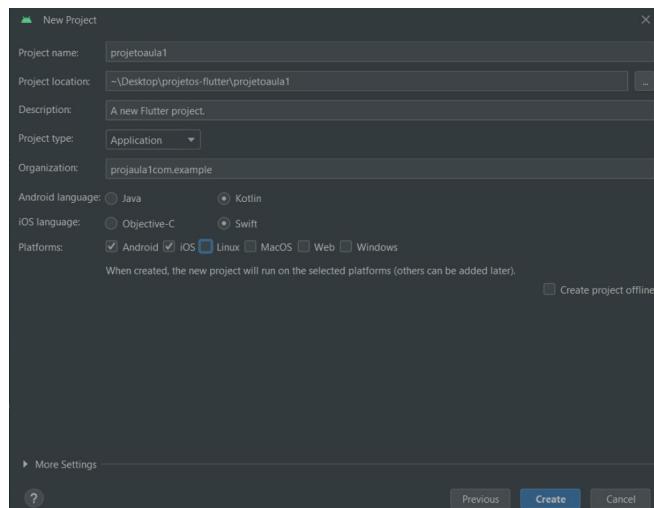
**Figura 1.18 – Instalação do plugin Dart.**

**8º passo:** criação de um projeto Flutter no Android Studio. Clicar em “New Flutter Project”



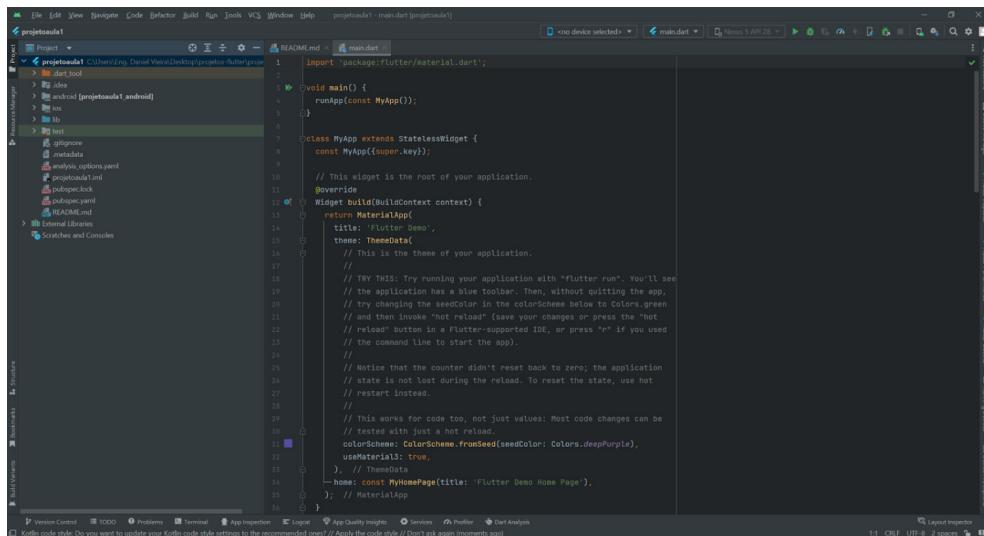
**Figura 1.19 – Tela inicial do Android Studio.**

**9º passo:** atribuir um nome para o projeto. O nome não pode começar com letra maiúscula nem com número. Em seguida, selecionar a plataforma Android e, depois, selecionar a pasta na qual será salvo o projeto. Para finalizar, clicar em **Create**.



**Figura 1.20 – Criação de pasta no Android Studio.**

Na figura a seguir é possível visualizar o projeto Flutter criado no Android Studio.



**Figura 1.21 – Projeto criado no Android Studio.**

## Linguagem Dart

A linguagem de programação Dart foi introduzida pelo Google em 2011 com a ambiciosa proposta de substituir o JavaScript como principal linguagem para desenvolvimento web. Embora esse objetivo inicial não tenha sido plenamente alcançado, Dart consolidou-se como uma ferramenta fundamental no desenvolvimento de aplicações modernas, especialmente após o lançamento do Flutter, o framework multiplataforma também desenvolvido pelo Google.

Com uma sintaxe inspirada em linguagens como C, Dart oferece uma curva de aprendizado acessível para desenvolvedores familiarizados com Java, C++ ou C#. Seguindo o paradigma da orientação a objetos, todos os elementos em Dart são tratados como objetos, herdando implicitamente da classe base `Object`. Isso proporciona uma estrutura unificada e coerente em todo o ecossistema da linguagem.

Dart é uma linguagem fortemente tipada, o que contribui para a segurança e robustez do código. No entanto, ela também oferece flexibilidade por meio da inferência de tipos, permitindo que o desenvolvedor omita declarações explícitas em muitos casos. Por exemplo, em vez de declarar `int x = 10;`, é possível simplesmente escrever `var x = 10;`, e o compilador inferirá automaticamente o tipo como inteiro.

Uma característica peculiar de Dart em relação à encapsulação é a forma como trata membros privados. Diferentemente de linguagens como Java ou C#, que utilizam a palavra-chave `private`, Dart adota uma abordagem mais simples: basta prefixar o nome do atributo, método ou classe com um sublinhado (`_`) para torná-lo privado ao arquivo onde foi definido. Essa convenção facilita o controle de visibilidade e contribui para um código mais limpo e direto.

Outro grande diferencial de Dart é sua capacidade de ser compilada tanto em ahead-of-time (AOT) quanto em just-in-time (JIT). Isso significa que o código pode ser transformado em um executável eficiente antes da execução, garantindo alto desempenho, ou compilado dinamicamente durante a execução, facilitando a depuração e o desenvolvimento.

Além disso, a funcionalidade **hot reload**, amplamente utilizada no desenvolvimento com Flutter, permite uma compilação em tempo real, atualizando imediatamente a interface do aplicativo sempre que o código é modificado. Isso acelera significativamente o ciclo de desenvolvimento e proporciona uma experiência ágil e interativa.

Com todas essas características, Dart se destaca como uma linguagem moderna, versátil e eficiente, posicionando-se como uma excelente escolha para desenvolvedores que buscam criar aplicações rápidas, seguras e escaláveis, tanto no ambiente web quanto no mobile.

Dart segue o paradigma de orientação a objetos, no qual todos os objetos herdam parâmetros da classe raiz `Object`. Apesar de ser fortemente tipada, a linguagem oferece uma abordagem flexível ao permitir a inferência de tipos, tornando opcional a declaração explícita.

## Variáveis

Uma variável em programação pode ser entendida como uma área de memória associada a um nome e destinada a armazenar valores de um tipo específico. Em Dart, as variáveis podem ser declaradas de maneira simples e com tipagem explícita ou inferida.

A declaração de variáveis em Dart pode ser feita utilizando a palavra-chave **var**, permitindo que o tipo seja inferido automaticamente pelo compilador. Por exemplo:

```
var nome = "Daniel Vieira";
String email = "danielvieira2006@gmail.com";
int numero = 50;
double preco = 19.99;
bool acesso = true; // ou false
```

### Tipos de variáveis

Em Dart, as variáveis podem ser classificadas como **mutáveis** ou **imutáveis**, dependendo da palavra-chave utilizada na declaração:

**var** = indica uma variável mutável, ou seja, seu valor pode ser alterado após a declaração.

```
var nome = "Daniel";
nome = "Vieira"; // permitido;
```

**const** = declara uma variável imutável, cujo valor deve ser conhecido em tempo de compilação e não pode ser alterado posteriormente.

```
const pi = 3.14159;
// pi = 3.14; // erro = não é possível modificar uma constante.
```

Dart diferencia entre **const** e **final**, ambos criando variáveis imutáveis, mas com uma diferença importante:

**const**: O valor é constante em tempo de compilação.

**final:** O valor é atribuído apenas uma vez, mas pode ser determinado em tempo de execução.

Veja um exemplo em que são utilizados o const e o final:

```
void main() {
    // Exemplo de `const`: o valor deve ser conhecido em tempo de
    // compilação;
    const double pi = 3.14159;
    print('O valor de pi é: $pi');

    // pi = 3.14; // ERRO: não é possível alterar uma variável const.

    // Exemplo de `final`: o valor é atribuído apenas uma vez, mas em
    // tempo de execução
    final DateTime currentTime = DateTime.now(); // O valor só é conhecido
    // em runtime
    print('O horário atual é: $currentTime');

    // currentTime = DateTime.now(); // ERRO: Não é possível reatribuir uma
    // variável final

    // Diferença prática
    // `const` exige um valor fixo e imutável em tempo de compilação.
    // `final` aceita valores que só podem ser determinados durante
    // a execução do programa
}.
```

## Função void main()

A função main() é o ponto de entrada para qualquer programa Dart. Todo código que se deseja executar começa aqui.

### const:

A variável pi é declarada como const, o que significa que seu valor deve ser conhecido em tempo de compilação. Isso quer dizer que o valor de pi não pode depender de cálculos ou operações realizadas em tempo de execução.

Aqui, pi é inicializado com o valor 3.14159, que é uma constante matemática fixa. Imutabilidade: qualquer tentativa de alterar o valor de pi resultará em erro. O comando print() exibe o valor de pi no console.

#### **final:**

A variável currentTime é declarada como final, o que significa que ela só pode ser atribuída uma vez.

Diferente de const, o valor de currentTime não precisa ser conhecido em tempo de compilação; ele é determinado durante a execução do programa (em runtime). No caso, usamos a função DateTime.now(), que retorna o horário atual no momento em que o programa é executado.

Imutabilidade: depois de atribuído, o valor de currentTime não pode ser modificado.

O comando print() exibe o valor atual da data e hora no console.

#### **const:**

Útil para valores fixos e conhecidos desde o momento em que o código é escrito, como números matemáticos, strings ou configurações constantes.

#### **final:**

Útil para valores que são imutáveis, mas cujo valor só pode ser determinado em tempo de execução, como informações dependentes do ambiente ou de cálculos dinâmicos.

No quadro a seguir é possível visualizar um mapa de memória de um computador exemplificando como cada variável fica armazenada em um endereço de memória diferente do computador.

#### **Quadro 1.1 – Mapa de memória com endereço de variáveis**

Memória do computador	
x	
nome	
pi	3.14

Os principais tipos de dados em Dart são: Int, double, string e bool. O quadro a seguir apresenta um exemplo de um programa desenvolvido na linguagem Dart com os principais tipos de variáveis utilizados.

**Quadro 1.2 – Código desenvolvido em Dart**

<b>Importa a biblioteca dart:io que permite que o usuário digite valores utilizando o teclado</b>	import “dart:io”;
<b>Função principal do código</b>	void main()
<b>Variável que armazena números inteiros</b>	{ int idade = 28;
<b>Exibe no terminal o valor da variável ligado</b>	print(“Idade: \$idade”);
<b>Variável que armazena números decimais</b>	double raio = 10.25;
<b>Exibe no terminal o valor da variável ligado</b>	print(“Raio: \$raio”);
<b>Variável que armazena caracteres e textos</b>	String nome = “Daniel”;
<b>Exibe no terminal o valor da variável nome</b>	print(“Ola \$nome, seja bem vindo”);
<b>Variável que armazena true ou false</b>	bool ligado = true;
<b>Exibe no terminal o valor da variável ligado</b>	print(“Estado \$ligado”);

## Operadores relacionais e lógicos

Os operadores lógicos são utilizados para realizar comparações entre valores ou expressões e retornar resultados booleanos (**true** ou **false**). Esses operadores são amplamente usados em estruturas de controle, como **if**, **while** e expressões condicionais. Na tabela a seguir, veremos os principais disponíveis em Dart:

**Tabela 1.6 – Operadores de comparação**

<b>Operadores de comparação</b>	<b>Função</b>	<b>Exemplo</b>
==	Comparação entre dois valores	<code>print(5 == 5); // true print(5 == 3); // false</code>
!=	Diferença entre dois valores	<code>print(5 != 3); // true print(5 != 5); // false</code>
>	Maior que Verifica se o valor à esquerda é maior que o valor à direita.	<code>print(10 &gt; 5); // true print(5 &gt; 10); // false</code>
<	Menor que Verifica se o valor à esquerda é menor que o valor à direita.	<code>print(5 &lt; 10); // true print(10 &lt; 5); // false</code>

Os operadores lógicos são ferramentas fundamentais na programação para realizar operações com expressões booleanas (verdadeiro ou falso). Eles permitem combinar, comparar ou inverter valores lógicos, sendo amplamente utilizados em condições e estruturas de controle.

Na tabela a seguir, temos os operadores lógicos disponíveis na linguagem Dart.

**Tabela 1.7 – Operadores lógicos**

<b>Operadores lógicos</b>	<b>Função</b>	<b>Exemplo</b>
&& (E)	Retorna true se ambas as condições forem verdadeiras.	<code>print(5 == 5); // true print(5 == 3); // false</code>
(OU)	Retorna true se pelo menos uma das condições for verdadeira.	<code>var notaProva = 4; var notaTrabalho = 8; print(notaProva &gt;= 6    notaTrabalho &gt;= 6); // true, pois a segunda condição é verdadeira</code>
! (NÃO/negação)	Inverte o valor lógico.	<code>bool aprovado = true; print(!aprovado); // false, pois o operador NOT inverte o valor</code>

A seguir, temos um exemplo de código completo que solicita que o usuário digite seu nome e sua idade.

**Tabela 1.8 – Exemplo de código completo solicitando que o usuário digite seu nome e idade**

Comando	Função
import 'dart:io';	Importa a biblioteca dart:io que permite que o usuário digite valores utilizando o teclado.
void main() {	Função principal do código.
print("Digite seu nome:");	Exibe mensagem no terminal solicitando ao usuário que digite seu nome.
String nome = stdin.readLineSync()!;	A variável <i>nome</i> armazena o que o usuário digitar. O comando <code>stdin.readLineSync()</code> captura o que o usuário digitar. ! indica que não está vazio e realiza a conversão para String.
print("Digite sua idade:");	Exibe mensagem no terminal solicitando ao usuário que digite sua idade.
String idadeString = stdin.readLineSync()!;	O comando <code>stdin.readLineSync()</code> captura o que o usuário digitar. ! indica que não está vazio e realiza a conversão para String.
int idade = int.parse(idadeString);	<code>int.parse</code> realiza a conversão de um dado string para inteiro.
print("Seu nome é: \$nome"); print("Sua idade é: \$idade"); }	Exibe as informações no terminal. "\$nome": o símbolo \$ permite acessar o valor armazenado na variável.

Na tabela a seguir, é possível visualizar os principais comandos utilizados para desenvolver um código utilizando a linguagem Dart.

**Tabela 1.9 – Principais comandos utilizados para desenvolver um código em Dart**

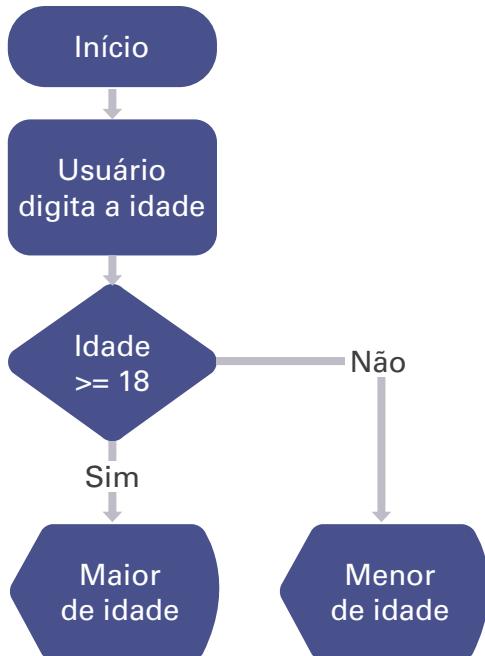
Comando	Função
import 'dart:io';	Importa a biblioteca dart:io que permite que o usuário digite valores utilizando o teclado.
void main() {}	void → Vazio, não retorna nada; main é a função principal do código: nela, todas as inicializações de bibliotecas e variáveis são executadas; () → recebe argumentos de fora.
print();	Função que imprime informações no console.
;	Necessário para terminar uma linha de código.
Para acessar informações de uma variável utilizamos o \$variável Necessário estar entre “\$variavel”	O \$ é uma interpolação de string que permite acessar o valor da variável.
int.parse(readLineSync()!); float.parse(readLineSync()!);	int.parse realiza a conversão de um dado string para int; float.parse realiza a conversão de um dado string para float.

## Estruturas condicionais

As estruturas condicionais são recursos oferecidos pelas linguagens de programação para que seja possível verificar uma condição e alterar o fluxo de execução do algoritmo.

Assim é possível definir uma ação específica para diferentes cenários e obter o resultado esperado, como no seguinte exemplo:

- exibir informações do aplicativo se o botão for pressionado;
- verificar o nome e a senha digitados pelo usuário;
- realizar login e, assim, exibir outra tela.



**Figura 1.22 – Fluxograma de estrutura condicional.**

Estruturas condicionais permitem controlar o fluxo do programa com base em condições específicas, como **if/else**.

Na tabela a seguir, é possível visualizar um exemplo de estrutura condicional com if/else para verificar a idade:

**Tabela 1.10 – Estrutura condicional com if/else**

Comando	Função
<code>int idade = 18;</code>	Declaração da variável <b>idade</b> com o valor <b>18</b> .
<code>if (idade &gt;= 18) {     print('Adulto'); } else {     print('Menor de idade'); }</code>	Estrutura condicional para verificar se a idade é maior ou igual a 18. Também exibe a mensagem no terminal. Se a condição for falsa, vai para o <b>else</b> e exibe a mensagem 'Menor de idade'.

Outra estrutura condicional é a switch case, que realiza a comparação de uma variável com diferentes valores e para no ponto em que a condição é atingida.

O switch/case realiza uma comparação direta de uma variável com valores constantes ou predefinidos. Ele para no momento que uma condição é atingida e executa o código correspondente. Suas principais características são:

- ideal para situações em que uma variável precisa ser comparada com diversos valores;
- é mais organizado e legível do que usar várias condições if/else;
- contém um caso padrão (default) que será executado quando nenhuma das condições anteriores for atendida.

Na tabela a seguir, temos um exemplo da estrutura switch/case utilizada para desenvolver um sistema de semáforo.

**Tabela 1.11 – Estrutura condicional com switch/case**

Comando	Função
<pre>String cor = 'vermelho'; switch (cor) {     case 'vermelho':         print('Pare!');         break;     case 'verde':         print('Siga!');         break;     default:         print('Atenção!'); }</pre>	<p>Declaração da variável <b>cor</b> em vermelho.</p> <p>Estrutura condicional que compara se a variável cor com diferentes valores e exibe mensagem no terminal.</p> <pre>switch (variavel) {     case valor1:         // Código a ser executado quando variavel == valor1         break; // Finaliza a execução do switch     case valor2:         // Código a ser executado quando variavel == valor2         break;     default:         // Código a ser executado quando nenhum caso é atendido }</pre> <p>O switch avalia a expressão (cor no exemplo) e compara seu valor com os casos definidos (case).</p> <p>Quando um case é satisfeito, o bloco associado é executado.</p> <p>O break é usado para evitar que outros casos sejam executados após um case ser satisfeito.</p> <p>O default é executado quando nenhum dos casos é atendido.</p>

## Estrutura de repetição

Uma estrutura de repetição é uma ferramenta poderosa na programação, pois permite executar o mesmo bloco de código várias vezes, dependendo do tipo e das condições definidas. Basicamente, ela automatiza tarefas que seriam repetitivas e demoradas se fossem feitas manualmente.

Compreender bem esses conceitos é essencial para quem está começando na programação, porque as estruturas de repetição estão no coração de diversos processos que envolvem manipulação e análise de dados.

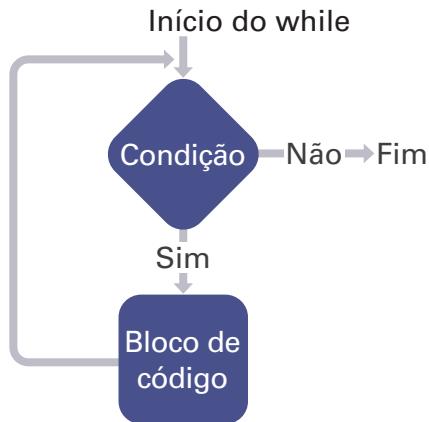
Mas afinal, quando usar uma estrutura de repetição? De forma prática, as estruturas de repetição são aliadas quando é necessário lidar com listas ou conjuntos de dados. Imagine que você tem uma lista com informações sobre pessoas que utilizam um sistema interno de uma empresa. Com uma estrutura de repetição, é possível:

- encontrar itens específicos, como todas as pessoas que compartilham um dado comum (por exemplo, idade ou cidade);
- ordenar a lista por critérios, como os nomes das pessoas ou a data de cadastro.

Além disso, há casos em que pode ser necessário interromper a repetição antes do término normal – por exemplo, se já encontrou o dado que você procurava. Esse tipo de controle, muito comum em sistemas baseados em linha de comando, permite otimizar o desempenho e garantir que a lógica do programa seja mais eficiente.

O segredo é entender que estruturas de repetição não são apenas ferramentas técnicas, e sim soluções práticas para organizar, filtrar e transformar informações no código. Entender isso é o primeiro passo para se tornar um programador ou programadora eficiente!

O fluxograma a seguir apresenta a lógica **while**.



**Figura 1.23 – Fluxograma estrutura de repetição while.**

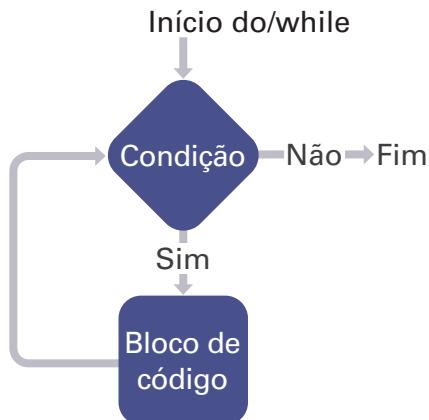
A seguir, temos a estrutura de repetição while que é utilizada quando é necessário executar uma sequência de comandos *n* vezes. É necessário verificar se a condição foi atingida dentro do loop; caso contrário, pode ocorrer o loop infinito, em que as tarefas do código são executadas sem parar e o programa pode ser travado. No exemplo a seguir, incrementamos a variável contador 5 vezes.

#### Estrutura condicional com switch/case

```

1. /*
2. Estrutura while(condição){
3. bloco de código
4. altera condição
5. }
6. */
7. //Código while
8. while
9. int contador = 0;
10. while (contador < 5) {
11.   print(contador);
12.   contador++;
13. }
  
```

Na figura a seguir, é possível visualizar o fluxograma da estrutura de repetição do/while. Essa estrutura é utilizada quando é necessário que o código seja executado pelo menos uma vez e que a condição no final do bloco de código seja verificada.



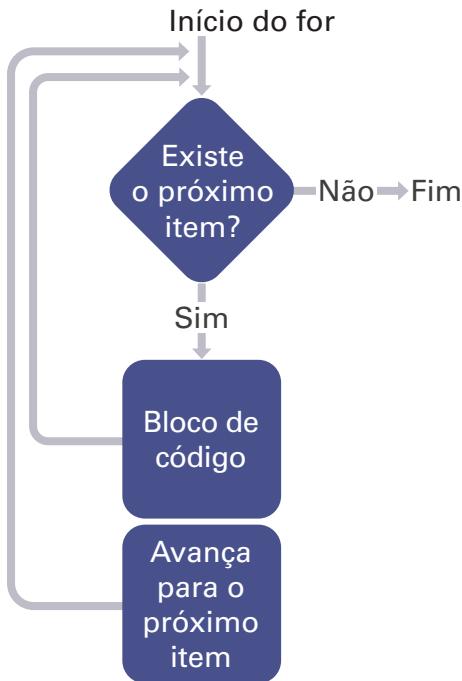
**Figura 1.24** – Fluxograma estrutura de repetição do/while.

### Estrutura de repetição while

```

1. /* Estrutura do {
2. bloco de código
3. altera condição
4. }while(condição){
5.
6.
7. } */
8.
9.
10. Código do-while
11. int contador = 0;
12. do {
13.   print(contador);
14.   contador++;
15. } while (contador < 5);
16.
  
```

Na figura a seguir, é possível visualizar o fluxograma da estrutura de repetição **for**.



**Figura 1.25** – Fluxograma da estrutura de repetição **for**.

A estrutura de repetição **for** tem três pilares principais a serem definidos:

- uma variável contadora;
- uma condicional baseada na variável;
- um pedaço de código que é executado ao final de cada repetição.

Além disso, por ser uma estrutura com uma sintaxe mais rígida, dificilmente teremos problema de repetição infinita, algo que ocorre em outros tipos de estrutura.

A estrutura de repetição **for** é utilizada quando já sabemos quantas vezes queremos repetir o bloco de código. De certo modo, é mais usada em listagens, já que, a partir do tamanho da lista, podemos percorrer cada item.

No exemplo a seguir, a variável **i** foi incrementada 5 vezes.

### Estrutura de repetição for

```
1. /*
2. Estrutura for
3. for(inicialização da variável, condição, incremento)
4. */
5. Código for
6. for (int i = 0; i < 5; i++) {
7.
8.   print(i); // 0, 1, 2, 3, 4
9. }
10.
11.
```

## Estrutura de dados

Estruturas de dados são fundamentais no desenvolvimento de software, sendo responsáveis por organizar e gerenciar dados de maneira eficiente. Em Dart, a linguagem que alimenta o Flutter, temos diversas opções de estruturas de dados que atendem às mais variadas necessidades, desde listas até mapas e conjuntos. Nas páginas seguintes, vamos explorar algumas dessas estruturas e demonstrar como utilizá-las de forma prática.

### Listas (List)

As listas em dart são coleções ordenadas de elementos, as quais permitem armazenar múltiplos valores em uma única variável, acessados por um índice. É possível criar listas fixas, com tamanho predefinido, ou listas dinâmicas, que crescem conforme são adicionados novos elementos.

## Criação de lista fixa

```

1. void main() {
2.   List<int> numeros = List.filled(5, 0); // [0, 0, 0, 0, 0]
    print(numeros);
  
```

## Criação de lista dinâmica

```

1. // Lista dinâmica
2. void main() {
3.   List<String> nomes = []; nomes.add('Daniel'); nomes.
    add('Vieira'); print(nomes); // ['Daniel', 'Vieira'] }
  
```

É possível realizar operações com as listas, como remover elementos, conforme demonstrado a seguir.

### Operações para remover um elemento da lista

```

1. // Criando lista dinâmica
2. void main() {
3.   List<int> numeros = [1, 2, 3];
4.   numeros.removeAt(1); // Remove o segundo elemento
5.   print(numeros); // [1, 3]
6. }
  
```

### Operação para iterar sobre cada elemento da lista

```

1. /*
2.  Realiza a exibição de cada elemento da lista utilizando o
  forEach que é similar ao for convencional
3. */
4. void main() {
5.   List<String> frutas = ['Maçã', 'Banana', 'Laranja'];
6.   frutas.forEach((fruta) => print(fruta));
7. }
8.
  
```

Na linguagem Dart, temos uma estrutura de dados chamada **Map**, que é similar ao formato Java Script Object Notation (JSON), no qual os dados são armazenados na forma de chave e valor.

### Estrutura de dados Map

```
1. // Estrutura de dados Map - chave-valor
2.
3. void main()
4. {
5.   Map<String, String> nomeSobrenome = {"Daniel": "Vieira",
   "Senai": "Roberto Mange"}; print(nomeSobrenome);
6. }
7.
```

Além das listas e mapas, existem outras estruturas de dados importantes que fornecem controle adicional sobre como os dados são manipulados e organizados. Essas estruturas são especialmente úteis em situações específicas, como o gerenciamento de tarefas, algoritmos e processamento de dados.

A seguir, vamos explorar as principais estruturas de dados: pilha, fila, deque, fila circular e lista ligada.

### Pilha (Stack)

A pilha segue o paradigma **LIFO** (Last In, First Out), ou seja, o último elemento a ser inserido é o primeiro a ser removido. Como exemplo, podemos imaginar uma pilha de pratos ou livros: só é possível adicionar ou remover elementos que estiverem no topo da pilha.

Operações principais:

- push – adicionar um elemento no topo;
- pop – remover um elemento do topo.

A seguir é possível visualizar a estrutura de dados do tipo pilha

### Estrutura de dados do tipo pilha

```

1. // Estrutura de dados do tipo pilha
2. void main() {
3.     List<int> pilha = [];
4.     // Adicionando elementos (Push) pilha.add(10); pilha.add(20);
        pilha.add(30); print(pilha); // [10, 20, 30]
5.     // Removendo o elemento do topo (Pop) int removido = pilha.
        removeLast();
6.     print(removido);
7.     // 30 print(pilha); // [10, 20]
8. }
9.
```

### Usos comuns:

- pilha de chamadas (Call Stack) em programas;
- navegação no histórico de páginas do navegador.

### Fila (Queue)

A fila segue o paradigma **FIFO** (First In, First Out), ou seja, o primeiro elemento a entrar é o primeiro a sair. É como uma fila de pessoas em uma bilheteria: a pessoa que chegou primeiro será atendida primeiro.

### Operações principais:

- **enqueue** – inserir um elemento no final da fila;
- **dequeue** – remover o elemento do início da fila.

A seguir é possível visualizar a estrutura de dados do tipo fila.

### Estrutura de dados do tipo fila

```

1. // Estrutura de dados do tipo fila
2. void main() {
3.     List<int> fila = []; // Inserindo elementos (Enqueue)
4.     fila.add(10);
```

```

5. fila.add(20);
6. fila.add(30); print(fila); // [10, 20, 30] // Removendo o
   elemento do início (Dequeue)
7. int removido = fila.removeAt(0); print(removido);
8. // 10 print(fila); // [20, 30]
9. }

```

### Usos comuns:

- fila de impressão;
- processamento de requisições em servidores.

### Deque (Double-Ended Queue)

O deque é uma estrutura de dados mais flexível, que permite adicionar e remover elementos tanto do início quanto do final. Ele combina características de pilha e fila.

#### Operações principais:

- inserir no início ou no final;
- remover do início ou do final.

A seguir, é possível visualizar a estrutura de dados do tipo deque.

#### Estrutura de dados do tipo deque

```

1. // Estrutura de dados do tipo Deque
2. void main() {
3.     List<int> deque = [];
4.
5.     // Inserindo no final
6.     deque.add(10);
7.     deque.add(20);
8.
9.     // Inserindo no início
10.    deque.insert(0, 5);
11.    print(deque); // [5, 10, 20]
12.

```

```
13. // Removendo do início
14. deque.removeAt(0);
15. print(deque); // [10, 20]
16.
17. // Removendo do final
18. deque.removeLast();
19. print(deque); // [10]
20. }
```

### Usos comuns:

- algoritmos que requerem manipulação em duas pontas;
- sistemas de edição, nos quais desfazer e refazer são permitidos.

As atividades a seguir auxiliam o entendimento dos conceitos básicos sobre linguagem Dart como estrutura condicional e repetição.

## Atividades

1. Crie um programa utilizando a linguagem Dart que receba informações digitadas pelo usuário: nome, idade, curso.

A seguir, o código a ser usado nesse primeiro exercício.

### Código da Atividade 1

```
1. import 'dart:io';
2.
3. // Solicita o nome do usuário
4. void main() {
5.   // Solicita o nome do usuário
6.
7.   print("Digite seu nome: ");
8.   String? nome = stdin.readLineSync();
9.
10. // Solicita a idade do usuário
```

```

11.
12. print("Digite sua idade: ");
13.   String? idadeInput = stdin.
        readLineSync();
14.   int idade = int.tryParse(idadeInput ??
        "0") ?? 0;
15.
16. // Solicita o curso do usuário
17.
18. print("Digite o curso: ");
19. String? curso = stdin.readLineSync();
20.
21. // Exibe as informações coletadas
22.
23. print("\nInformações coletadas:");
24. print("Nome: ${nome ?? 'Não
        informado'}");
25. print("Idade: ${idade > 0 ? idade : 
        'Não informada ou inválida'}");
26. print("Curso: ${curso ?? 'Não
        informado'}");
27.

```

Para executar o código desenvolvido em Dart, ele deve ser salvo como **nomedoexercicio.dart**. Ao executar **dart nomedoexercicio.dart**, o resultado será o demonstrado na figura a seguir.

```

PS C:\Users\dsadm\Desktop\CodigosDart> dart ex1.dart
Digite seu nome:
Daniel
Digite sua idade:
29
Digite o curso:
Analise e Desenvolvimento de Sistemas

Informações coletadas:
Nome: Daniel
Idade: 29
Curso: Analise e Desenvolvimento de Sistemas
PS C:\Users\dsadm\Desktop\CodigosDart>

```

**Figura 1.26** – Exibição das informações da Atividade 1.

**2. Crie um programa que receba a nota de dois alunos, calcule a média e, conforme o resultado, informe se o aluno está aprovado ou reprovado.**

- Se a média for maior ou igual a 7: aprovado.
- Maior ou igual a 4 e menor do que 7: exame.
- Menor do que 4: reprovado.

**3. Crie um programa que receba a idade de duas pessoas e insira o print na tela indicando qual é a pessoa mais velha.**

**4. Crie um programa que receba o valor médio de três modelos de carro e indique qual é mais caro e o mais barato.**

**5. Crie um programa que receba notas de 10 alunos e calcule a média das notas.**

**6. Um posto de combustíveis oferece descontos variados com base no tipo de combustível adquirido e na quantidade comprada. O desconto é aplicado diretamente sobre o valor total, e as condições específicas para cada tipo de combustível devem ser consideradas.**

Escreva um programa que:

a) solicite ao usuário:

- a **quantidade de litros** comprada.
- o **tipo de combustível**, sendo:
  - E para etanol;
  - D para diesel;
  - G para gasolina.

b) calcule:

- o **valor do desconto** utilizando a fórmula **desconto = preço do litro × quantidade de litros × percentual de desconto**.
- o **valor total a ser pago** utilizando a fórmula **valor total = (preço do litro × quantidade de litros) – desconto**.

c) exiba o valor a ser pago pelo cliente.

**Observação:** o programa deve tratar corretamente os diferentes tipos de combustíveis e a fórmula do desconto depende do preço do litro e do percentual aplicável para cada combustível.

Na tabela a seguir, temos o valor de cada combustível e a porcentagem de desconto.

**Tabela 1.12 –** Valor dos combustíveis e desconto

<b>Tipo de combustível</b>	<b>Valor</b>	<b>Desconto</b>
Etanol	R\$ 1,70	Compra $\geq$ 15L Desconto de 4% por litro Compra $<$ 15L Desconto de 3% por litro
Diesel	R\$ 2,00	Compra $\geq$ 15L Desconto de 5% por litro Compra $<$ 15L por litro Desconto de 3% por litro
Gasolina	R\$ 4,50	Compra $\geq$ 20L Desconto de 3% por litro Compra $<$ 20L sem desconto

A seguir, temos o código da Atividade 6.

#### Código para a Atividade 6

```

1. import 'dart:io';
2.
3. /*
4. Função principal do código e declaração
   de constantes com o valor dos
   combustíveis
5. */
6. void main() {
7.   // Preços dos combustíveis
8.   const double precoEtanol = 1.70;
9.   const double precoDiesel = 2.00;
10.  const double precoGasolina = 4.50;

```

```
11.  
12. // Solicita a quantidade de litros.  
13.  
14. print("Digite a quantidade de litros  
comprados: ");  
15. String? litrosInput = stdin.  
    readLineSync();  
16. double litros = stdin.readLineSync();  
17. litros = double.tryParse(litrosInput ??  
    "0") ?? 0;  
18. // Solicita o tipo de combustível.  
19. O Comando toUpperCase converte a  
    letra digitada pelo usuário para letra  
    maiúscula.  
20.  
21. print("Digite o tipo de combustível  
    (E para Etanol, D para Diesel, G para  
    Gasolina): ");  
22. String? tipoCombustivel = stdin.  
    readLineSync()?.toUpperCase();  
23.  
24. // Variáveis para armazenar o preço e  
    desconto.  
25. double precoPorLitro = 0;  
26. double percentualDesconto = 0;  
27.  
28.  
29. switch (tipoCombustivel) {  
30.     case 'E':  
31.         precoPorLitro = precoEtanol;  
32.         if (litros >= 15) {  
33.             percentualDesconto = 0.04; //  
            4% de desconto  
34.         } else {
```

```
35.         percentualDesconto = 0.03; // 3%
    de desconto
36.     }
37.     break;
38. case 'D':
39.     precoPorLitro = precoDiesel;
40.     if (litros >= 15) {
41.         percentualDesconto = 0.05; //
    5% de desconto
42.     } else {
43.         percentualDesconto = 0.03; // 3%
    de desconto
44.     }
45.     break;
46. case 'G':
47.     precoPorLitro = precoGasolina;
48.     if (litros >= 20) {
49.         percentualDesconto = 0.03; // 3%
    de desconto
50.     } else {
51.         percentualDesconto = 0.00; //
    Sem desconto
52.     }
53.     break;
54. default:
55.     print("Tipo de combustível
    inválido.");
56.     return;
57. }
58. double desconto = precoPorLitro *
    litros * percentualDesconto;
59. double valorTotal = (precoPorLitro *
    litros) - desconto;
60. // Exibe o valor total a ser pago
```

```

61. print("Valor total a ser pago:  

       R\$\{valorTotal.toStringAsFixed(2)\}");  

62. }

```

```

Quantidade de litros: 12.0 L
Preço por litro: R$ 1.70
Desconto aplicado: R$ 0.61
Valor total a ser pago: R$ 19.79
PS C:\Users\dsadm\Desktop\CodigosDart> dart ex2.dart
Digite a quantidade de litros comprados: 20
Digite o tipo de combustível (E para Etanol, D para Diesel, G para Gasolina): E

Resumo da compra:
Tipo de combustível: E
Quantidade de litros: 20.0 L
Preço por litro: R$ 1.70
Desconto aplicado: R$ 1.36
Valor total a ser pago: R$ 32.64
PS C:\Users\dsadm\Desktop\CodigosDart> []

```

**Figura 1.27** – Exibição das informações da Atividade 6.

**7. Escreva um programa que calcule o preço a pagar pelo fornecimento de energia elétrica.**

- Pergunte para o usuário a quantidade de kWh consumida e o tipo de instalação: Residência (R), Indústria (I), Comércio (C).
- Calcule o preço da energia com base na tabela a seguir.
- O preço a pagar pelo fornecimento da energia elétrica deve ser calculado preço unitário do KWh \* quantidade de KWh inserido pelo usuário.

**Tabela 1.13** – Dados para a Atividade 7

<b>_tipo</b>	<b>Faixa(KWh)</b>	<b>Preço (R\$)</b>
Residencial	Até 500	0,50
	Acima de 500	0,70
Comercial	Até 1 000	0,65
	Acima de 1 000	0,60
Industrial	Até 5 000	0,55
	Acima de 5 000	0,50

**8. Crie um programa que solicite ao usuário um número e calcule sua tabuada.**

**9. Crie um programa que receba dois valores numéricos digitados pelo usuário. Depois, permita que ele escolha a operação que deseja realizar entre as seguintes opções: soma (+), subtração (-), multiplicação (\*) e divisão (/).**

**10. Crie um programa que receba 20 valores de temperatura ambiente simulando um sensor de temperatura cujo range de medição é de 0 a 100°C; calcule sua média e imprima o maior e o menor valor de temperatura digitados pelo usuário.**

## Funções

No contexto da programação, função é um bloco de código projetado para realizar uma tarefa específica. Ela encapsula uma lógica que pode ser reutilizada em diferentes partes do programa, tornando o código mais organizado, modular e manutenível. As funções desempenham um papel fundamental no desenvolvimento de aplicativos móveis, especialmente em frameworks como Flutter, no qual funções são amplamente usadas na manipulação de dados e na construção de interfaces.

Vejamos agora as características das funções:

- reutilização – uma vez definida, uma função pode ser chamada várias vezes, evitando duplicação de código;
- modularidade – funções ajudam a dividir o programa em partes menores e mais gerenciáveis;
- legibilidade – nomear funções de forma descritiva torna o código mais fácil de entender;
- manutenção – alterações em uma função afetam todas as suas chamadas, facilitando atualizações e correções.

Na tabela a seguir, é possível visualizar a estrutura de uma função em Dart.

**Tabela 1.14 – Estrutura de uma função em Dart**

Código	Função
<pre>tipoRetorno nomeDaFuncao(parâmetros) { // Corpo da função return valor; // Opcional }</pre>	<p><b>tipoRetorno:</b> Especifica o tipo do valor retornado pela função (por exemplo, int, String, void). Se a função não retorna nada, usamos void.</p> <p><b>nomeDaFuncao:</b> Nome único que identifica a função.</p> <p><b>parâmetros:</b> Valores que a função recebe para realizar sua tarefa (opcional).</p> <p><b>return:</b> Usado para retornar um valor da função (opcional).</p>

Na tabela a seguir é possível visualizar os tipos de função em Dart.

**Tabela 1.15 – Tipos de função em Dart**

Código	Função
<pre>void saudar(String nome) { print("Olá, \$nome! Bem-vindo ao Flutter."); }  void main() { saudar("Daniel"); }</pre>	<p><b>Função sem retorno</b> Usada para executar uma tarefa sem retornar um valor.</p>
<pre>int somar(int a, int b) { return a + b; }  void main() { int resultado = somar(5, 10); print("A soma é: \$resultado"); // Saída: A soma é: 15 }</pre>	<p><b>Função com retorno</b> Usada quando é necessário obter um valor como resultado da função.</p>
<pre>void exibirMensagem(String mensagem, [String remetente = "Anônimo"]) { print("Mensagem de \$remetente: \$mensagem"); }  void main() { exibirMensagem("Bem-vindo ao curso!"); // Remete: Anônimo exibirMensagem("Parabéns pelo progresso!", "Professor"); }</pre>	<p><b>Função com passagem de parâmetros opcionais</b> Dart permite declarar parâmetros opcionais, que podem ter valores-padrão.</p>

Código	Função
<pre>void criarUsuario({required String nome, int idade = 18}) {     print("Usuário: \$nome, Idade: \$idade"); }  void main() {     criarUsuario(nome: "Ana");     criarUsuario(nome: "Carlos", idade: 25); }</pre>	<p><b>Função com passagem de parâmetros nomeados</b> Os parâmetros podem ser nomeados, tornando o código mais legível.</p>
<pre>(){     print("Desenvolvimento Mobile") }</pre>	<p><b>Função anônima</b> Funções sem nome são úteis quando é necessário a lógica temporária.</p>
<pre>Future&lt;void&gt; carregarDados() async {     print(&lt;&lt;"Carregando..."&gt;&gt;);     await Future.delayed(Duration(seconds: 2));     print(&lt;&lt;"Dados carregados!"&gt;&gt;); }  void main() async {     await carregarDados(); }</pre>	<p><b>Função assíncrona</b> Usada para operações que demoram para ser concluídas, como chamadas de APIs.</p>

## Funções em aplicações móveis

As funções são frequentemente usadas para:

- manipular dados exibidos na interface do usuário;
- realizar cálculos complexos;
- chamar apis para buscar ou enviar dados;
- modularizar a construção de widgets no Flutter.

Os principais benefícios de utilizar funções no desenvolvimento de código são:

- funções bem estruturadas;
- reutilização de código – evita duplicação e simplifica alterações;
- manutenção facilitada – alterações em uma função afetam todas as suas chamadas;

- melhor desempenho – funções assíncronas otimizam operações demoradas;
- colaboração – funções bem nomeadas e documentadas facilitam o trabalho em equipe.

## Atividades

**1. Crie uma função que receba as informações de um usuário inseridas por meio de digitação no teclado: nome, curso, idade.**

**2. Crie uma função que calcule a área de um triângulo a partir de dados digitados pelo usuário.  $A = (b * h) / 2$  e retorne esse valor.**

**3. Crie uma função que calcule o salário líquido do usuário a partir dos valores digitados pelo teclado, considerando um desconto de 10% de impostos e bonificação de 20% em cima do salário.**

**4. Crie um programa de transações bancárias com os seguintes itens:**

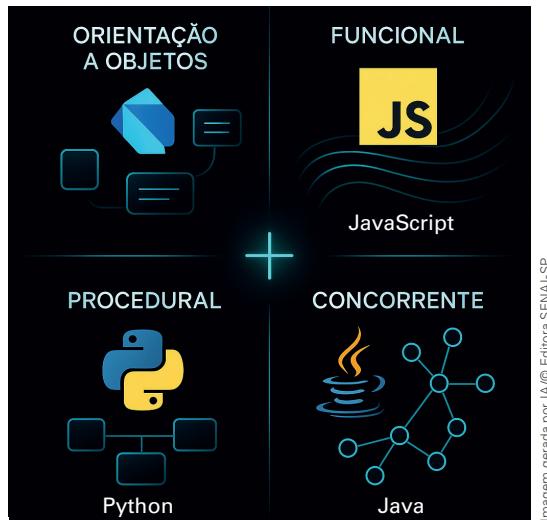
- saque;
- pix;
- empréstimos;
- transferências.

Para cada opção do menu, pergunte ao usuário o valor para realizar a transação e passe o valor por função.

**5. Crie um programa que converta valores de reais (R\$) para outras moedas de acordo com a escolha do usuário: euro (EUR), dólar (US\$), franco suíço (CHF).**

# Paradigmas de linguagem de programação

Na figura a seguir é possível visualizar os principais paradigmas de linguagem de programação, destacando-se: programação orientada a objeto, funcional, procedural e concorrente.



**Figura 1.28 – Paradigmas de programação.**

## Paradigma imperativo

No paradigma imperativo, como o próprio nome sugere, o desenvolvedor descreve passo a passo as instruções que a máquina deve seguir para executar uma tarefa. Dentro dele, existem duas abordagens principais:

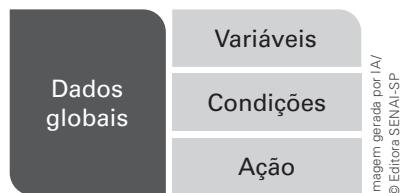
**programação procedural** – esse paradigma é ideal para tarefas gerais de programação, pois se baseia em uma sequência de instruções que o computador executa de forma linear, uma de cada vez. A maioria das linguagens de programação tradicionais, como C, C++ e Java, são procedurais.

**programação orientada a objetos (POO)** – esse paradigma é um dos mais aplicados por conta das vantagens que traz para o processo, como a **modularidade do código**, além da função de criar relações entre problemas reais dentro dos termos de código.

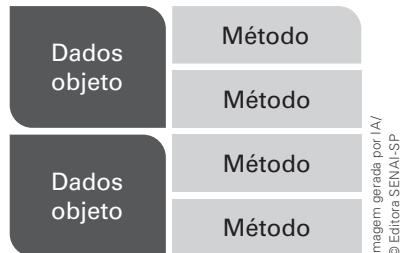
# Programação orientada a objeto

A programação orientada a objeto (POO) é um paradigma de programação que organiza o software em objetos. Esses objetos são entidades que combinam dados (atributos) e comportamentos (métodos), representando elementos do mundo real ou conceitual no código.

Nas figuras a seguir, é possível visualizar a estrutura de uma programação estruturada e da programação orientada a objeto.



**Figura 1.29 – Programação estruturada.**



**Figura 1.30 – Programação orientada a objeto.**

Na programação estruturada, temos os conceitos indicados a seguir.

**Dados globais** – informações que podem ser acessadas em qualquer parte do código, ou seja, têm um escopo global.

**Variáveis** – espaços de memória para armazenar valores. Elas permitem que o programa guarde e manipule dados durante sua execução.

**Condições** – verificações de valores ao longo do código que ajudam a tomar decisões e direcionar o fluxo de execução dependendo do resultado da comparação.

**Ação** – refere-se à execução de comportamentos específicos, como exibir informações na tela e salvar itens em um banco de dados, entre outras operações.

Na programação orientada a objetos, cada objeto é composto por dados (também chamados de variáveis ou atributos) e métodos (funções associadas ao objeto). Isso significa que cada objeto possui seus próprios atributos e métodos, o que contribui para a modularidade e a organização do código.

Os dados de um objeto representam os valores associados ao objeto específico que foi instanciado a partir de uma classe; já os métodos são as funções definidas na classe, que podem ser executadas pelos objetos criados a partir dessa classe.

## Por que utilizar programação orientada a objeto?

A programação orientada a objetos surgiu para resolver problemas comuns enfrentados por desenvolvedores em paradigmas anteriores, como a programação estruturada. Entre os principais motivos de sua criação estão os itens relacionados a seguir.

**Complexidade do software:** Com o crescimento do software, tornou-se difícil organizar e manter programas estruturados, especialmente os com muitos trechos de código interdependentes. A POO oferece uma abordagem modular, na qual o software é dividido em objetos, cada um com sua responsabilidade.

**Reutilização de código:** em paradigmas anteriores, o código muitas vezes precisava ser reescrito para ser reutilizado. Com a POO, conceitos como herança e polimorfismo permitem reaproveitar funcionalidades existentes.

**Manutenção e escalabilidade:** a modularidade da POO facilita encontrar e corrigir erros, além disso, classes e objetos podem ser adicionados ou modificados sem afetar outras partes do sistema.

**Modelagem do mundo real:** a POO permite mapear entidades do mundo real diretamente para o código. Um carro, por exemplo, pode ser representado por uma classe com atributos, como marca e modelo; e métodos, como acelerar.

**Segurança e controle de acesso:** o encapsulamento protege os dados do objeto, permitindo acesso apenas por métodos específicos; isso reduz erros causados por alterações não controladas nos dados.

**Colaboração em equipes:** projetos grandes envolvem múltiplos desenvolvedores. A POO facilita a divisão do trabalho em módulos independentes (classes e objetos), promovendo a colaboração.

## Aplicações da POO

A programação orientada a objeto tem as seguintes aplicações:

- desenvolvimento de software corporativo – sistemas de gestão (ERP, CRM, por exemplo) frequentemente modelam entidades como clientes, produtos e pedidos;
- jogos – jogos digitais usam objetos como personagens, armas e ambientes;
- desenvolvimento mobile – em Flutter, o conceito de widgets é baseado na POO;
- sistemas orientados a serviços – APIs e serviços web modelam entidades e suas interações usando objetos.

## Principais elementos da POO

### Paradigma

Um paradigma é um conjunto de crenças, valores, técnicas e práticas que orientam o modo de pensar, agir e resolver problemas em determinada área ou contexto. Ele funciona como um modelo ou padrão aceito que serve de referência para a compreensão e a abordagem de determinado assunto.

## Classe

É um molde ou modelo para criar objetos. Define os atributos (dados) e métodos (comportamentos) que um objeto terá. **Classe** e **objeto** são dois conceitos extremamente importantes na programação orientada a objetos. O objeto é uma instância da classe. Ele é criado a partir da classe e comportamentos definidos pela classe. Cada objeto tem seus atributos definidos pela classe, mas compartilha os métodos.

Na figura a seguir, é possível visualizar a planta de uma casa e seu objeto.

Classe: Casa



Objeto: Casa



Imagem gerada por IA/© Editora SENAI-SP

**Figura 1.31 – Classe e objeto.**

A seguir é possível visualizar a estrutura de uma classe em Dart.

### Estrutura de uma classe em Dart

```

1. /*
2. Cria uma classe chamada carro que tem dois atributos: a
   marca e o ano; e um método chamado acelerar que escreverá
   a mensagem no terminal o carro está acelerando.
3. */
4. class Carro {
5.     String marca;
6.     int ano;
7.
8.     void acelerar() {

```

```

9.     print("O carro está acelerando!");
10.    }
11. }
12.
13.
14.
15.

```

## Objeto

É uma instância de uma classe. Um objeto herda os atributos e métodos definidos na classe. A seguir, é possível visualizar um objeto instanciado da classe **carro**.

### Objeto instanciado da classe carro

```

1. Carro meuCarro = Carro(); // Cria objeto carro
2. meuCarro.marca = "Toyota";
3. meuCarro.ano = 2020;
4. meuCarro.acelerar();
5.

```

## Encapsulamento

Protege os dados de um objeto, permitindo acesso apenas através de métodos controlados. Promove a segurança e modularidade do código.

A seguir é possível visualizar uma classe pessoa com encapsulamento para proteger os dados.

### Classe pessoa com encapsulamento

```

1. // Classe pessoa com atributos privados que garantem a
   segurança dos dados.
2. class Pessoa {
3.     String _nome; // Atributo privado
4.

```

```
5.     void setNome(String nome) {  
6.         _nome = nome;  
7.     }  
8.  
9.     String getNome() {  
10.        return _nome;  
11.    }  
12. }
```

## Herança

Permite que uma classe herde atributos e métodos de outra classe e promove a reutilização de código.

A seguir é possível visualizar o código utilizando o recurso de herança.

### Exemplo de código de herança

```
1. /*  
2. Cria uma classe denominada animal com o método dormir.  
3. */  
4. class Animal {  
5.     void dormir() {  
6.         print("Animal dormindo...");  
7.     }  
8. }  
9. // Cria uma classe denominada cachorro herdando o método  
// dormir da classe animal.  
10.  
11. class Cachorro extends Animal {  
12.     void latir() {  
13.         print("Cachorro latindo...");  
14.     }  
15. }
```

## Polimorfismo

Permite que objetos de classes diferentes sejam tratados como objetos de uma mesma superclasse. O comportamento do método pode variar dependendo do objeto.

A seguir é possível visualizar o exemplo de código utilizando polimorfismo.

### Exemplo de código com polimorfismo

```
1. /*
2. Classe forma com o método desenhar
3. Classe circulo herda método desenhar com método.
4. O comportamento do método pode variar de acordo com o
   objeto.
5. */
6.
7. class Forma {
8.     void desenhar() {
9.         print("Desenho genérico");
10.    }
11. }
12.
13. class Circulo extends Forma {
14.     @override
15.     void desenhar() {
16.         print("Desenhando um círculo");
17.    }
18. }
19.
20. void desenharForma(Forma forma) {
21.     forma.desenhar();
22. }
```

## Abstração

Foca nos aspectos essenciais de um objeto, escondendo os detalhes internos, e usa classes abstratas e interfaces para definir comportamentos comuns. A seguir, é possível visualizar um exemplo de código com abstração.

### Exemplo de código com abstração

```

1. // Classe abstrata animal
2.
3. abstract class Animal {
4.   void emitirSom();
5.
6.
7. class Gato extends Animal {
8.   @override
9.   void emitirSom() {
10.     print("Miau!");
11.   }
12. }
13.

```

## Construtor

Na programação orientada a objetos em Dart, um construtor é um método especial usado para inicializar uma classe. Ele permite passar parâmetros ao criar um objeto e configura os atributos da classe com os valores fornecidos. Assim, garante que os objetos sejam instanciados com valores consistentes. **Definição:** o construtor tem o mesmo nome da classe. **Parâmetros:** os atributos da classe podem ser inicializados diretamente dentro do construtor usando a sintaxe **this.atributo**.

## Uso do this

O this refere-se aos atributos da instância atual da classe, diferenciando-os de variáveis ou parâmetros com o mesmo nome. A seguir é possível visualizar o exemplo de código utilizando construtor.

## Exemplo de código com construtor

```
1. /*
2. Construtor com this:
3.
4. Fruta(this.sabor, this.nome, this.cor, this.peso, this.
   diasDesdeColheita);
5. O uso de this associa os parâmetros do construtor
   diretamente aos atributos da classe, reduzindo a
   necessidade de inicializá-los manualmente.
6.
7. Atributo Opcional:
8. O atributo isMadura é marcado como nulo opcional (bool?), 
   o que significa que ele pode ou não ter um valor atribuído.
9.
10. Métodos:
11. A classe possui um método verificaMaturidade que realiza
    uma ação com base nos valores dos atributos.
12. */
13. class Fruta {
14.     // Atributos da classe
15.     String sabor;
16.     String nome;
17.     String cor;
18.     double peso;
19.     int diasDesdeColheita;
20.     bool? isMadura; // Opcional (pode ser nulo)
21.
22.     // Construtor: inicializa os atributos da classe
23.     Fruta(this.sabor, this.nome, this.cor, this.peso, this.
   diasDesdeColheita);
24.
25.     // Método para verificar se a fruta está madura
26.     void verificaMaturidade(int diasParaMaturidade) {
27.         if (diasDesdeColheita >= diasParaMaturidade) {
28.             print("A $nome está madura.");
```

```
29.     } else {
30.         print("A $nome ainda não está madura.");
31.     }
32. }
33.
34.
35. void main() {
36.     // Criando um objeto da classe Fruta
37.     Fruta manga = Fruta("Doce", "Manga", "Amarela", 1.2, 5);
38.
39.     // Chamando o método para verificar maturidade
40.     manga.verificaMaturidade(4); // Saída: "A Manga está
        madura."
41.     manga.verificaMaturidade(6); // Saída: "A Manga ainda não
        está madura."
42. }
43.
```

Entre os benefícios do construtor, estão:

- simplificar a inicialização – os atributos são configurados diretamente na criação do objeto;
- oferecer um código mais legível – reduz a necessidade de atribuições manuais;
- proporcionar flexibilidade – permite criar objetos com valores personalizados.

## Atividades

1. Crie uma classe chamada “pessoa” com os seguintes atributos: nome, idade, profissão, salário. Utilize o método exibetrabalho(String nomeempresa, int tempo de trabalho) Print(nome da empresa, tempo de trabalho).

**2. Crie uma classe chamada “automóvel”, que deve ser a classe mãe e ter os seguintes parâmetros: cor do automóvel, modelo, tipo de combustível, quantidade de rodas. Crie classes filhas denominadas: carro, moto e caminhão, herdando características da classe automóvel. Método a ser utilizado: ligar carro, desligar carro, abrir vidro, descer vidro.**

**3. Crie um programa que permita ao usuário informar dados pessoais e, em seguida, realizar transações bancárias com base nas opções escolhidas. Nome, profissão, saldo.**

Os métodos são:

- pix (double valor);
- empréstimo (double valor);
- saque (double valor);
- extrato (double valor).

**4. Crie uma classe denominada “máquinas” com os seguintes atributos: nome da máquina, quantidade de eixos, rotações por minuto, consumo de energia elétrica. Essa classe deve ser a mãe de outras classes.**

Criar classe filha denominada “furadeira”, herdando o nome da máquina, rotações por minuto, consumo de energia elétrica. Criar métodos para ligar e desligar a máquina e um método para ajustar a velocidade de rotação da máquina.

**5. Crie uma classe denominada “produtos” com os seguintes parâmetros: nome do produto, quantidade, preço do produto, tipo de comunicação, consumo de energia elétrica. Essa classe de produtos deve ser a mãe de outras classes, como fritadeira, televisão, ar-condicionado.**

As classes filhas devem ter o seguinte método: ligar, desligar, ajuste de temperatura com passagem de parâmetros para setpoint.

**6. Crie uma classe denominada “componentes eletrônicos” com os seguintes parâmetros: nome do componente, valor (por exemplo, 1k, 200R, 330, 10uF, 100uF), quantidade de componentes. Essa classe de produtos deve ser a mãe de outras classes, como resistor, capacitor, indutor, diodo, led.**

As classes filhas devem ter o seguinte método: exibir informações dos componentes e associar componentes. Nesse método, deve-se somar os valores dos componentes da classe e mostrar o valor da associação.

**7. Crie uma classe chamada “carrinho de compras” com os seguintes atributos: itens e quantidades.**

Método:

- adicionar\_itens – adiciona um item ao carrinho;
- remover\_item – remove um item do carrinho;
- calcular\_o\_total – retorna o valor total do carrinho.

**8. Crie uma classe chamada “carro” com os seguintes atributos: marca, modelo, ano, motor ligado.**

Método:

- ligar\_motor(): – método que liga o motor do carro e atualiza o atributo motor\_ligado para True;
- desligar\_motor(): – método que desliga o motor do carro e atualiza o atributo motor\_ligado para False;
- status\_motor(): – método que retorna uma mensagem indicando se o motor está ligado ou desligado.

Teste sua classe criando um objeto carro, ligando e desligando o motor e verificando seu status.

## Get e Set

Os getters e setters em Dart são métodos especiais usados para acessar e modificar os atributos (ou propriedades) de uma classe, fornecendo maior controle sobre como os dados são manipulados. Eles têm as utilidades indicadas a seguir.

- **encapsulamento:** protegem os atributos da classe, permitindo acesso e manipulação controlados;
- » **validação e lógica:** permitem incluir lógica ao acessar ou alterar os valores, como verificar se o novo valor é válido antes de atribuí-lo;

- » **interface limpa:** escondem a complexidade oferecendo uma interface de acesso parecida com a de atributos normais;
- » **imutabilidade controlada:** podem impedir alterações diretas em atributos que precisam ser apenas lidos (getter sem setter).

A sintaxe de getters e setters em Dart ocorre da maneira indicada a seguir.

- **getter:** um método que retorna o valor de um atributo;
- **setter:** um método que define (ou altera) o valor de um atributo.

A palavra-chave `get` é usada para definir um getter, e `set` é usada para definir um setter. A seguir é possível visualizar o exemplo de código da classe `produto` utilizando o getter e setter.

#### Código com getter e setter

```
1. /*
2. Encapsulamento:
3.
4. O atributo _preco é privado (só pode ser acessado
dentro da classe). O getter e setter permitem acessá-lo e
modificá-lo de forma controlada.
5.
6. Validação no Setter:
7.
8. Antes de alterar o valor de preco, o setter verifica
se o valor é maior que zero. Se não for, ele rejeita a
alteração.
9.
10. Uso prático:
11.
12. O acesso ao preço é feito de forma transparente com o
getter (produto.preco), como se fosse um atributo normal.
13. A atribuição do preço utiliza o setter (produto.preco
= valor), permitindo que a validação seja aplicada
automaticamente.
14. */
```

```
15. class Produto {  
16.     // Atributo privado (convenção: usa-se ‘_’ para indicar  
     // privado)  
17.     double _preco = 0.0;  
18.  
19.     // Getter: Retorna o valor do atributo _preco  
20.     double get preco => _preco;  
21.  
22.     // Setter: Define um novo valor para _preco com  
     // validação  
23.     set preco(double novoPreco) {  
24.         if (novoPreco > 0) {  
25.             _preco = novoPreco;  
26.         } else {  
27.             print("O preço deve ser maior que zero.");  
28.         }  
29.     }  
30. }  
31.  
32. void main() {  
33.     // Criando um objeto da classe Produto  
34.     Produto produto = Produto();  
35.  
36.     // Usando o setter para definir o preço  
37.     produto.preco = 50.0; // Valor válido  
38.     print("Preço do produto: R\${produto.preco}"); // Saída:  
     // R$50.0  
39.  
40.     produto.preco = -10.0; // Valor inválido  
41.     // Saída: “O preço deve ser maior que zero.”  
42.  
43.     print("Preço do produto: R\${produto.preco}"); // Saída:  
     // R$50.0 (valor anterior não foi alterado)  
44. }
```

Vantagens de usar getters e setters:

- **segurança** – impedem que dados sejam alterados diretamente sem passar por regras ou validações;
- **flexibilidade** – permitem implementar lógica adicional sem mudar a interface pública da classe;
- **manutenção** – facilitam a evolução do código, pois as mudanças podem ser feitas nos getters e setters sem alterar a interface da classe.

## Atividades

**1. Crie uma classe mãe chamada “animal” com os atributos: string nome, int idade, string cor, string raça.**

**2. Crie uma classe filha com o tipo de animal “pássaro, cachorro, tigre, peixe” e o atributo peso. Utilize os métodos “acordou, dormiu”.**

**3. Crie uma classe abstrata denominada “alimentar” com três métodos:**

- separar ingredientes;
- pegar tigela;
- preparar comida.

Essa classe abstrata deve ser implementada na classe filha.

**4. Crie uma classe abstrata denominada “máquina industrial” com os seguintes métodos:**

- nome – nome da máquina;
- potência da máquina;
- status – booleano.

Métodos abstratos

- Ligar() – método abstrato que define como a máquina é ligada;
- Desligar () – método abstrato que define como a máquina é desligada.

5. Crie duas subclasses concretas de máquinas industriais **prensa** e **robô solda**. A **prensa** deve ter um atributo adicional chamado “**pressão em toneladas**” e os métodos **ligar** e **desligar** devem exibir mensagens adequadas. **Robô solda** deve ter um atributo chamado **tipo de solda(String)** para especificar o tipo de solda que realiza. Os métodos **ligar** e **desligar** devem exibir mensagens adequadas.
6. Crie uma classe chamada “**transportador**”, que herda os parâmetros **nome**, **potência da máquina**, **status**, métodos **ligar** e **desligar** de máquina industrial. Essa classe deve ter um atributo adicional chamado **velocidade (float )m/s**.
7. Crie instâncias das classes **prensa**, **robosolda** e **transportador**. Ligue e desligue cada uma delas para testar seus métodos.
8. Crie uma lista chamada “**máquinas**” e adicione todas as instâncias das máquinas anteriormente a essa lista. Em seguida, percorra a lista e, para cada máquina, exiba nome, potência e status.
9. Crie uma classe **FornecedorDeEnergia** com um método **fornecerEnergia(maquina: MaquinalIndustrial)**, que receba uma instância de **MaquinalIndustrial** e ligue a máquina se houver energia suficiente (suponha que a energia seja um recurso limitado). Implemente esse método de forma que ele só ligue a máquina se houver energia suficiente e exiba uma mensagem informando se a máquina foi ligada ou não.
10. Crie uma classe chamada “**produto**”, com **nome**, **preço** e **quantidade**, e utilize o comando **get** para passar o **nome** para o **produto**, o **preço** e a **quantidade**, e um método para exibir as informações do **produto**.
11. Crie a classe “**pessoa**” com **getters** e **setters** para o **nome** e **idade** da pessoa. Os **setters** permitem configurar esses atributos com verificações de validade. A classe também possui um método para exibir as informações da pessoa.

**12. Crie uma classe abstrata chamada “automóveis”, com nome, cor e ano como atributos.**

**13. Crie uma classe chamada “carro”, herdando características da classe abstrata “automóveis”.**

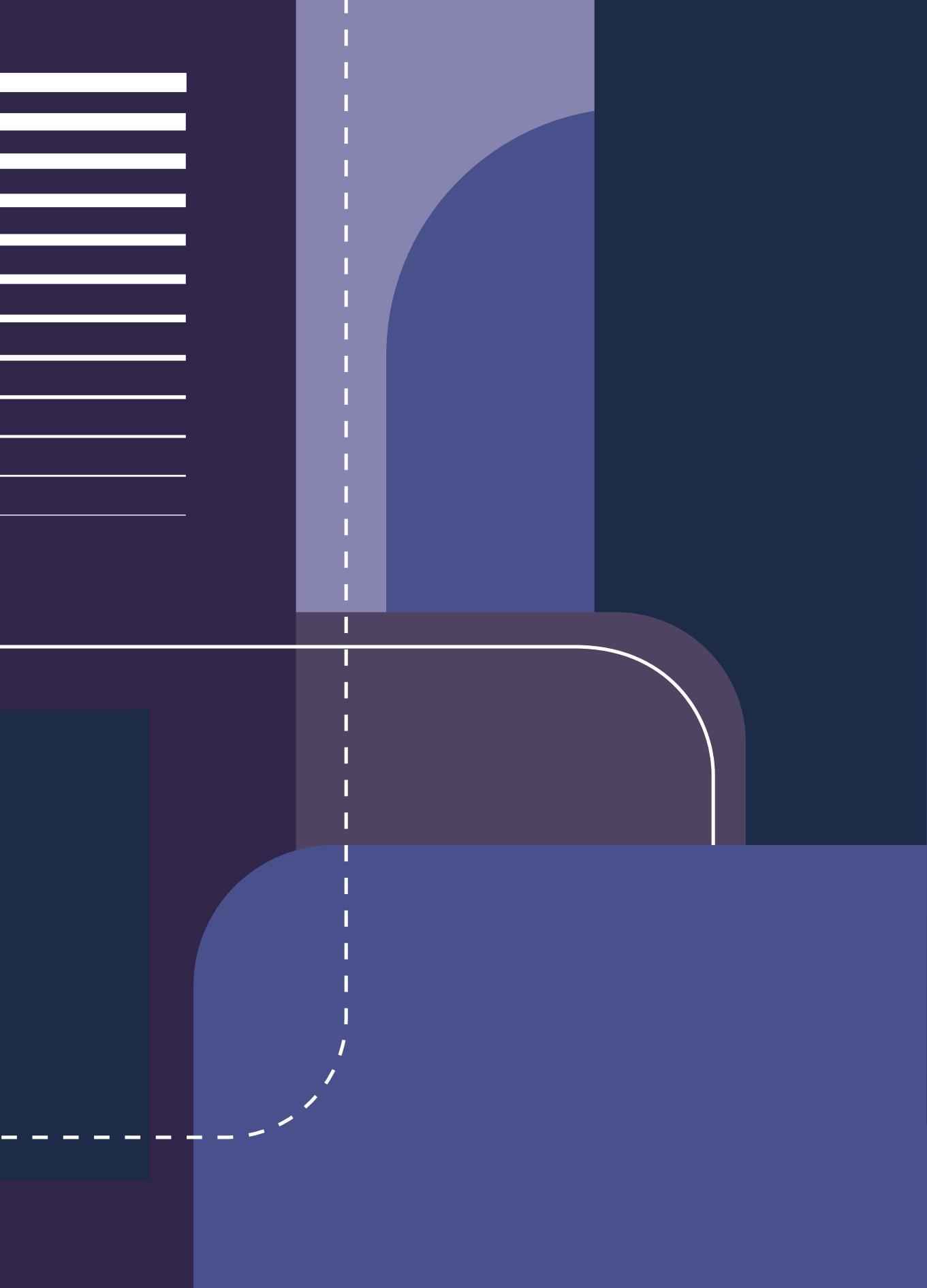
**14. Crie métodos abstratos na classe abstrata, como colocar o cinto, ligar o carro, desligar o carro e dirigir.**

**15. Crie uma classe concreta “carro”, implementando os métodos abstratos e exibindo mensagens adequadas.**

## Conclusão

Neste primeiro capítulo, foram apresentados os conceitos históricos sobre dispositivos móveis, as diferenças entre o desenvolvimento nativo e o híbrido, suas vantagens e desvantagens, além das principais empresas que utilizam o Flutter como framework para desenvolvimento mobile. Também aprendemos a fazer o download do Git, do SDK do Flutter e do Android Studio, e discutimos conceitos sobre a linguagem Dart, tipos de variáveis, estruturas condicionais, estruturas de repetição, estruturas de dados e paradigmas de programação, acompanhados de exercícios práticos.

No próximo capítulo, abordaremos os componentes utilizados no Flutter para criar um aplicativo, o layout da tela, o posicionamento de componentes, entre outros tópicos.





## CAPÍTULO 2

# CRIAÇÃO DE INTERFACE

### Introdução

Neste capítulo, vamos explorar os princípios fundamentais da criação de interfaces no Flutter. Abordaremos tópicos essenciais como componentes visuais, estrutura e tipos de widgets, organização de layouts, gerenciamento de estado, recursos e construção de interfaces dinâmicas.

Você aprenderá a trabalhar com menus, diálogos, barras de ação, controle de elementos na tela, além do tratamento de eventos e exceções. Também veremos como manipular listas, realizar entrada, processamento e saída de dados e implementar navegação entre telas com passagem de parâmetros. Por fim, discutiremos o uso de gestos e o gerenciamento de dependências para tornar seus aplicativos mais interativos e escaláveis.

Prepare-se para criar interfaces incríveis e levar suas habilidades em Flutter a um novo nível!

O framework Flutter oferece uma vasta gama de componentes de UI (User Interface) conhecidos como widgets. Eles são a base de qualquer aplicação Flutter e incluem:

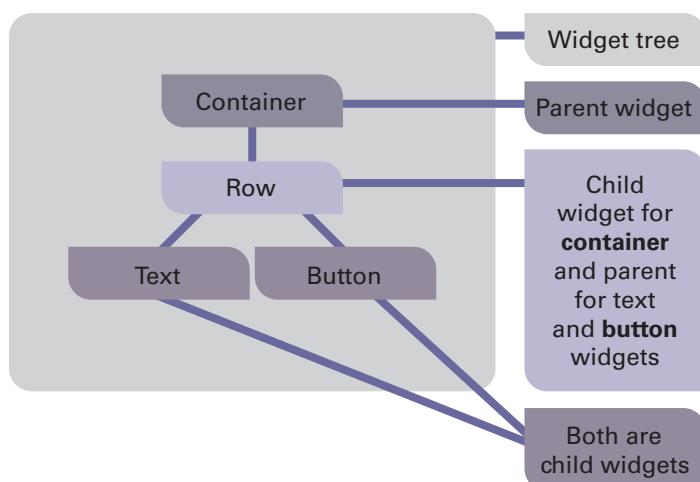
**widgets básicos:** text, image, icon, container. Esse tipo de widget é utilizado para estilizar e organizar layouts etc.;

**widgets de layout:** row, column, stack, listview (gerar listas de rolagem), gridview, entre outros;

**widgets interativos:** elevated button, gesture detector, switch, checkbox etc.;

**custom widgets:** cria widgets personalizados para reutilização de código e consistência de design.

A figura a seguir apresenta a estrutura de componentes do Flutter, representada por uma típica árvore de widgets (widget tree). Essa visualização ilustra a organização hierárquica dos elementos que compõem o layout de uma interface, destacando como os widgets se relacionam e se aninham para formar a estrutura da aplicação.



**Figura 2.1 – Estrutura da árvore de widgets de componentes do Flutter.**

# Widget tree (árvore de widgets)

Toda a interface em Flutter é organizada como uma árvore de widgets. Nessa estrutura hierárquica, os widgets pais contêm widgets filhos, criando uma organização aninhada e modular.

## Container

O container é o widget pai (parent widget) nesta árvore. Ele atua como o contêiner principal da interface, encapsulando e organizando outros widgets dentro de sua área, servindo como base para a composição do layout.

### Row

Dentro do container, há um widget row, que funciona como seu widget filho (child widget) e, simultaneamente, como o widget pai de outros elementos. O row é responsável por organizar seus widgets filhos horizontalmente, alinhando-os lado a lado em uma única linha.

### Text e button

Os widgets text e button são filhos do widget row, sendo posicionados lado a lado na interface. Eles representam os elementos terminais da árvore de widgets nessa estrutura, responsáveis por exibir diretamente o conteúdo e permitir a interação do usuário com a aplicação.

## Relações de parentesco

Vejamos agora como são as relações de parentesco entre os itens que compõem um widget:

- **container**: atuação como widget pai para o row;
- **row**: atua como widget filho do container e, ao mesmo tempo, como widget pai para os widgets text e button;

- **text e button:** são os widgets filhos de row e, portanto, representam os elementos de interface visíveis na tela;

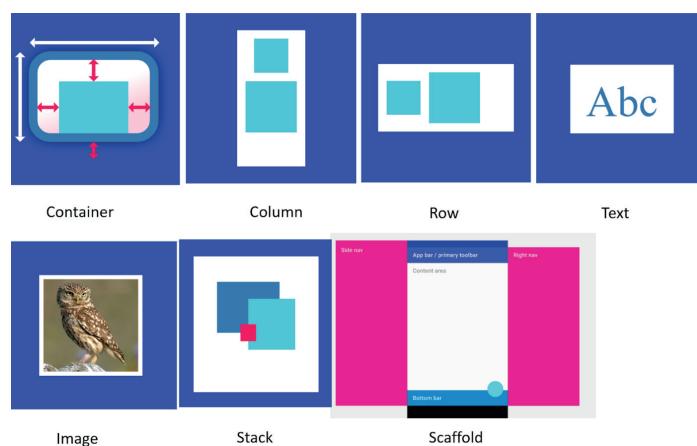
Essa estrutura evidencia como o Flutter adota uma organização hierárquica, na qual os widgets podem atuar como pais ou filhos. Esse modelo permite a construção de layouts complexos a partir do simples aninhamento de widgets, promovendo flexibilidade e modularidade no desenvolvimento da interface.

## 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 predefinidas que facilitam a organização de widgets na tela.

A figura a seguir mostra os **widgets básicos** utilizados no Flutter.



**Figura 2.2 – Widgets básicos utilizados no Flutter.**

**SAIBA MAIS**

Os códigos de cada widget podem ser consultados na própria documentação do Flutter. Disponível em: <https://docs.flutter.dev/ui/widgets/basics>. Acesso em: 20 jul. 2025. Também é possível acessar o arquivo pelo QR Code.

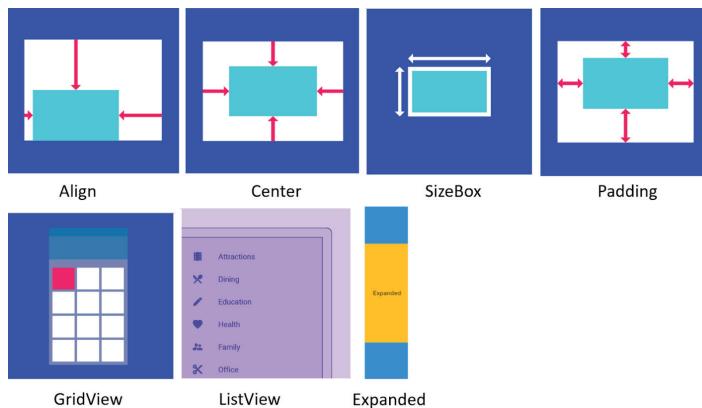


## Widgets de layout

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 uma 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.

Os principais **widgets de layout** do Flutter, utilizados para organizar outros widgets da interface, são mostrados na figura a seguir.



**Figura 2.3 – Widgets de layout do Flutter.**

**SAIBA MAIS**

Para mais informações sobre os widgets de layout, acessar o link disponível em: <https://docs.flutter.dev/ui/widgets/layout>. Acesso em: 20 jul. 2025. Também é possível acessar o arquivo pelo QR Code.



## Widgets interativos

Os widgets interativos são widgets que respondem a interações do usuário.

### 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. O código exemplo para o ElevatedButton é apresentado a seguir.

Código exemplo para o ElevatedButton

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

## TextButton

Um botão representado apenas por texto, geralmente usado para ações secundárias. Não tem elevação ou fundo destacado. O código exemplo para o TextButton é mostrado a seguir.

Código exemplo para o TextButton

```
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. )
```

## IconButton

Um botão que usa um ícone como visual principal e é ideal para ações rápidas, como botões de voltar ou de configurações.

O código exemplo para o IconButton é indicado a seguir.

Código exemplo do IconButton

```
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.
```

## FloatingActionButton

O FloatingActionButton é um componente proeminente; um botão flutuante geralmente usado para a principal ação de uma tela.

Código exemplo para FloatingActionButton

```
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.
```

## GestureDetector

Deteta gestos do usuário, como toques, deslizes, longos pressionamentos, entre outros, em qualquer widget.

## Código exemplo para o GestureDetector

```

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.  ),)

```

## Switch

O switch é um botão de alternância utilizado para representar uma escolha entre dois estados, geralmente “ligado” e “desligado”. Ele permite ao usuário ativar ou desativar uma funcionalidade de forma simples e intuitiva.

## Código exemplo para o switch

```

1. /*
2. bool isSwitched = false;
3.
4. Representa o estado inicial do botão Switch.
5. O estado é false, indicando que o botão começa na posição
   desligada.
6. Switch:
7.

```

```
8. value: Define o estado atual do Switch. Aqui ele usa a  
variável isSwitched.  
9. onChanged: Callback acionado ao alternar o estado do  
botão. Atualiza o estado de isSwitched e imprime o novo  
valor no console.  
10. */  
11.  
12. bool isSwitched = false;  
13.  
14. Switch(  
15.   value: isSwitched,  
16.   onChanged: (value) {  
17.     isSwitched = value;  
18.     print('Switch: $isSwitched');  
19.   },  
20. )  
21.
```

## Slider

É um controle deslizante usado para selecionar um valor dentro de um intervalo contínuo. Ele permite ao usuário ajustar configurações, como volume ou brilho, de forma precisa e intuitiva, arrastando o controle horizontalmente (ou verticalmente, dependendo da orientação).

Código exemplo para o 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.
```

## CheckBox

CheckBox é uma caixa de seleção que pode estar marcada ou desmarcada, usada para opções booleanas.

### Exemplo de código para o CheckBox

```
1. /*
2. bool isChecked = false;
3.
4. Define o estado inicial do checkbox como desmarcado
   (false).
5. Checkbox:
6.
7. value: Controla o estado atual do checkbox (true para
   marcado, false para desmarcado).
```

```
8. onChanged: Callback acionado quando o estado do checkbox  
é alterado. O novo estado é passado como parâmetro  
(value).  
9. value!:  
10.  
11. 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.  
12. */  
13.  
14. bool isChecked = false;  
15.  
16. Checkbox(  
17.   value: isChecked,  
18.   onChanged: (value) {  
19.     isChecked = value!;  
20.     print('Checkbox: $isChecked');  
21.   },  
22. )  
23.
```

## Radio

Um botão de seleção exclusiva, usado em conjunto para criar opções mutuamente exclusivas.

Exemplo de código para o radio

```
1. /*  
2. int? selectedOption = 1;  
3.  
4. A variável selectedOption armazena a opção selecionada.  
Inicialmente, ela é definida como 1, marcando o primeiro  
botão de rádio.
```

```
5. Radio:  
6.  
7. value: O valor associado ao botão de rádio específico.  
8. groupValue: O valor atualmente selecionado no grupo.  
    Apenas o botão de rádio cujo value coincide com groupValue  
    estará marcado.  
9. onChanged: Callback acionado quando o botão de rádio é  
    selecionado. Aqui, a variável selectedOption é atualizada  
    para refletir a nova seleção.  
10. */  
11. int? selectedOption = 1;  
12.  
13. Column(  
14.   children: [  
15.     Radio<int>(  
16.       value: 1,  
17.       groupValue: selectedOption,  
18.       onChanged: (value) {  
19.         selectedOption = value;  
20.         print('Radio: $selectedOption');  
21.       },  
22.     ),  
23.     Radio<int>(  
24.       value: 2,  
25.       groupValue: selectedOption,  
26.       onChanged: (value) {  
27.         selectedOption = value;  
28.         print('Radio: $selectedOption');  
29.       },  
30.     ),  
31.   ],  
32. )  
33.
```

## TextField

Um campo de entrada de texto que permite ao usuário inserir e editar informações.

### Exemplo de código para o TextField

```
1. /*
2. onChanged:
3.
4. 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.
5. decoration:
6.
7. Personaliza a aparência do TextField. Aqui está sendo
   usado o widget InputDecoration com:
8. labelText: Exibe um rótulo acima do campo de texto quando
   ele está em foco ou preenchido.
9. border: Define um contorno ao redor do campo de texto
   usando OutlineInputBorder.
10. */
11.
12. TextField(
13.   onChanged: (text) {
14.     print('Texto inserido: $text');
15.   },
16.   decoration: InputDecoration(
17.     labelText: 'Digite algo',
18.     border: OutlineInputBorder(),
19.   ),
20. )
21.
```

## Widgets de apresentação (Display Widgets)

Widgets de apresentação são utilizados para exibir informações para o usuário.

### Widget Card

Usado para criar um contêiner estilizado que destaca seu conteúdo.

#### Exemplo de código para o Card

```
1. /*
2. elevation: 4;
3.
4. Define a altura da sombra do card, criando um efeito de
   profundidade. Valores maiores aumentam a intensidade da
   sombra.
5. margin: EdgeInsets.all(16);
6.
7. Define uma margem de 16 pixels ao redor do card,
   espaçando-o dos elementos vizinhos.
8. shape: RoundedRectangleBorder:
9.
10. Configura o formato do card, neste caso, com bordas
    arredondadas usando um raio de 16 pixels.
11. Padding:
12.
13. Espaçamento interno de 16 pixels entre o conteúdo do card
    e suas bordas.
14. Column:
15.
16. Organiza o conteúdo verticalmente, incluindo:
17. Text: Um título estilizado com tamanho de fonte maior e
    peso negrito.
18. SizedBox: Adiciona um espaço vertical de 8 pixels.
19. Outro Text com o conteúdo descritivo
```

```
20. */
21. Card(
22.   elevation: 4, // Altura da sombra
23.   margin: EdgeInsets.all(16), // Margem ao redor do card
24.   shape: RoundedRectangleBorder(
25.     borderRadius: BorderRadius.circular(16), // Bordas
26.     arredondadas
27.   ),
28.   child: Padding(
29.     padding: EdgeInsets.all(16),
30.     child: Column(
31.       mainAxisAlignment: MainAxisAlignment.min,
32.       children: [
33.         Text(
34.           'Título do Card',
35.           style: TextStyle(fontSize: 18, fontWeight:
36.             FontWeight.bold),
37.           ),
38.           SizedBox(height: 8),
39.           Text('Este é um exemplo de conteúdo dentro de um
40.           Card.'),
41.         ],
42.       ),
43.     )
```

## Widget Chip

A seguir é possível visualizar o código exemplo para o Chip, que é um pequeno elemento que guarda informações resumidas. O widget Chip é usado para criar um contêiner estilizado que destaca seu conteúdo.

## Código exemplo para o Chip

```
1. /*
2. label:
3.
4. Texto exibido no chip, neste caso: “Exemplo de Chip”.
5. avatar:
6.
7. Um ícone ou avatar exibido à esquerda do texto. No
   exemplo, é usado um CircleAvatar com o texto “E” e fundo
   azul.
8. onDeleted:
9.
10. Callback acionado quando o botão de exclusão (ícone) é
    pressionado. Aqui, imprime “Chip removido!” no console.
11. deleteIcon:
12.
13. Ícone exibido à direita do chip, representando a ação de
    exclusão. O ícone usado é o Icons.close.
14.
15. */
16.
17. Chip(
18.   label: Text('Exemplo de Chip'),
19.   avatar: CircleAvatar(
20.     backgroundColor: Colors.blue,
21.     child: Text('E'),
22.   ),
23.   onDeleted: () {
24.     print('Chip removido!');
25.   },
26.   deleteIcon: Icon(Icons.close),
27. )
```

## Widget ToolTip

A seguir é possível visualizar o código exemplo para o ToolTip, que exibe uma dica quando o usuário passa o cursor sobre o widget.

### Exemplo de código para o ToolTip

```

1. Tooltip(
2. /*
3. message:
4.
5. Define o texto que será exibido no tooltip. Neste caso, o
   texto é “Este é um ícone de configurações”.
6. child:
7.
8. É o widget ao qual o tooltip está associado. Aqui, é um
   ícone de configurações (Icons.settings) com tamanho 40.
9.
10.*/
11. message: 'Este é um ícone de configurações',
12. child: Icon(
13.   Icons.settings,
14.   size: 40,
15. ),
16. )
17.

```

## Widget ProgressIndicator

A seguir, temos um exemplo de código para o ProgressIndicator. O ProgressIndicator exibe barras de progresso e pode ser circular ou linear.

### Exemplo de código para o ProgressIndicator

```

1. CircularProgressIndicator(
2.   value: 0.7, // Valor de progresso (0.0 a 1.0)
3.   strokeWidth: 6, // Largura da borda
4.   backgroundColor: Colors.grey[300],

```

```
5.   valueColor: AlwaysStoppedAnimation<Color>(Colors.blue),
6. )
7.
8. /*
9. value:
10.
11. Representa o progresso atual.
12. Valor entre 0.0 (0%) e 1.0 (100%).
13. Aqui, 0.7 indica que 70% da tarefa foi concluída.
14. Se omitido, o indicador será indeterminado (um círculo
    girando continuamente).
15. strokeWidth:
16.
17. Define a espessura da borda do indicador.
18. Aqui, a borda tem uma largura de 6 pixels.
19. backgroundColor:
20.
21. Cor do fundo do círculo que representa a parte não
    preenchida (os 30% restantes, neste caso).
22. valueColor:
23.
24. Define a cor da parte preenchida do indicador.
25. AlwaysStoppedAnimation<Color> é usado para manter a cor
    constante (neste caso, azul).
26. */
27.
28. LinearProgressIndicator(
29.   value: 0.5, // Valor de progresso (0.0 a 1.0)
30.   backgroundColor: Colors.grey[300],
31.   valueColor: AlwaysStoppedAnimation<Color>(Colors.green),
32. )
33.
34.
35.
36.
37.
```

# Widgets de navegação (Navigation Widgets)

São widgets utilizados para gerenciar a navegação na interface.

A seguir, temos um exemplo de código para o BottomNavigationBar. Esse widget alterna entre diferentes telas ou conteúdos quando o usuário clica em um dos itens da barra.

Exemplo de código para o BottomNavigationBar

```
1. /*
2. BottomNavigationBar:
3.
4. Usado para criar a barra de navegação inferior.
5. Contém uma lista de itens (BottomNavigationBarItem) que
   representam as abas.
6. Aba Selecionada:
7.
8. A aba ativa é controlada pela variável _selectedIndex.
9. A tela correspondente é exibida a partir da lista _pages.
10. Método onTap:
11.
12. Atualiza o índice da aba selecionada com setState,
    garantindo que a interface seja atualizada.
13. Propriedades do BottomNavigationBar:
14.
15. currentIndex: Define qual aba está ativa.
16. selectedItemColor: Cor do item ativo.
17. onTap: Função que é chamada ao clicar em uma aba.
18. */
19. import 'package:flutter/material.dart';
20.
21. void main() {
22.   runApp(MyApp());
23. }
```

```
24.  
25. class MyApp extends StatelessWidget {  
26.   @override  
27.   Widget build(BuildContext context) {  
28.     return MaterialApp(  
29.       home: BottomNavBarExample(),  
30.     );  
31.   }  
32. }  
33.  
34. class BottomNavBarExample extends StatefulWidget {  
35.   @override  
36.   _BottomNavBarExampleState createState() =>  
37.   _BottomNavBarExampleState();  
38. }  
39. class _BottomNavBarExampleState extends  
40.   State<BottomNavBarExample> {  
41.   int _selectedIndex = 0; // Índice da aba selecionada  
42.   // Lista de widgets para cada aba  
43.   static const List<Widget> _pages = <Widget>[  
44.     Center(child: Text('Página Inicial', style:  
45.       TextStyle(fontSize: 24))),  
46.     Center(child: Text('Buscar', style: TextStyle(fontSize:  
47.       24))),  
48.     Center(child: Text('Perfil', style: TextStyle(fontSize:  
49.       24))),  
50.   ];  
51.   // Método para atualizar o índice quando o usuário clica  
52.   // em um item  
53.   void _onItemTapped(int index) {  
54.     setState(() {  
55.       _selectedIndex = index;  
56.     });  
57.   }  
58. }
```

```
54.    }
55.
56.    @override
57.    Widget build(BuildContext context) {
58.        return Scaffold(
59.            appBar: AppBar(
60.                title: Text('Exemplo de BottomNavigationBar'),
61.            ),
62.            body: _pages[_selectedIndex], // Exibe o conteúdo da
   aba selecionada
63.            bottomNavigationBar: BottomNavigationBar(
64.                items: const <BottomNavigationBarItem>[
65.                    BottomNavigationBarItem(
66.                        icon: Icon(Icons.home),
67.                        label: 'Início',
68.                    ),
69.                    BottomNavigationBarItem(
70.                        icon: Icon(Icons.search),
71.                        label: 'Buscar',
72.                    ),
73.                    BottomNavigationBarItem(
74.                        icon: Icon(Icons.person),
75.                        label: 'Perfil',
76.                    ),
77.                ],
78.                currentIndex: _selectedIndex, // Índice da aba
   selecionada
79.                selectedItemColor: Colors.blue, // Cor do item
   selecionado
80.                onTap: _onItemTapped, // Função ao clicar em um
   item
81.            ),
82.        );
83.    }
84. }
85.
```

**AppBar** é um widget usado no Flutter para criar uma barra superior que geralmente contém o título da página, ícones para ações rápidas (como um menu ou botão de pesquisa), e suporte para navegação. Ele é altamente personalizável e essencial para aplicativos modernos. É possível visualizar a seguir um exemplo de código para o AppBar.

#### Exemplo de código para o AppBar

```
1. /*
2. title:
3.
4. Exibe um título centralizado ou alinhado à esquerda
   (dependendo do tema).
5. No exemplo, o título é um widget Text com o texto “Título
   da Página”.
6. actions:
7.
8. Lista de widgets posicionados à direita da barra de
   título.
9. Geralmente usado para ícones de ações rápidas, como
   busca, compartilhamento ou configurações.
10. IconButton:
11.
12. Um botão que exibe um ícone.
13. icon: Define o ícone exibido.
14. onPressed: Callback que define a ação executada ao
   pressionar o botão.
15. */
16.
17. AppBar(
18.   title: Text('Título da Página'),
19.   actions: [
20.     IconButton(
21.       icon: Icon(Icons.search),
22.       onPressed: () {
```

```
23.         print('Botão de busca pressionado');
24.     },
25. ),
26. IconButton(
27.     icon: Icon(Icons.more_vert),
28.     onPressed: () {
29.         print('Mais ações pressionadas');
30.     },
31. ),
32. ],
33. )
34.
```

O **Drawer** no Flutter é um widget usado para criar um menu lateral deslizante, que pode ser acessado deslizando a partir da borda esquerda ou clicando no ícone do menu no AppBar. É amplamente usado em aplicativos para navegação entre diferentes seções ou exibição de opções importantes. A seguir é possível visualizar um exemplo de código para o Drawer.

#### Exemplo de código para o Drawer

```
1. /*
2. drawer:
3.
4. O widget que cria o menu lateral.
5. Geralmente colocado dentro do Scaffold.
6. DrawerHeader:
7.
8. Um cabeçalho decorativo para o menu.
9. Pode conter informações como o nome do usuário, foto de
   perfil, ou outras informações importantes.
10. ListView:
11.
12. Usado para listar os itens do menu.
```

```
13. ListTile:  
14.  
15. Cada item do menu é um ListTile, com:  
16. leading: Ícone exibido à esquerda.  
17. title: Texto do item.  
18. onTap: Callback acionado ao clicar no item.  
19.  
20. */  
21. import 'package:flutter/material.dart';  
22.  
23. void main() {  
24.   runApp(MyApp());  
25. }  
26.  
27. class MyApp extends StatelessWidget {  
28.   @override  
29.   Widget build(BuildContext context) {  
30.     return MaterialApp(  
31.       home: Scaffold(  
32.         appBar: AppBar(  
33.           title: Text('Exemplo de Drawer'),  
34.         ),  
35.         drawer: Drawer(  
36.           child: ListView(  
37.             padding: EdgeInsets.zero,  
38.             children: <Widget>[  
39.               DrawerHeader(  
40.                 decoration: BoxDecoration(  
41.                   color: Colors.blue,  
42.                 ),  
43.                 child: Text(  
44.                   'Cabeçalho do Menu',  
45.                   style: TextStyle(  
46.                     color: Colors.white,
```

```
47.                     fontSize: 24,
48.                 ),
49.             ),
50.         ),
51.     ListTile(
52.         leading: Icon(Icons.home),
53.         title: Text('Início'),
54.         onTap: () {
55.             print('Início pressionado');
56.         },
57.     ),
58.     ListTile(
59.         leading: Icon(Icons.settings),
60.         title: Text('Configurações'),
61.         onTap: () {
62.             print('Configurações pressionado');
63.         },
64.     ),
65.     ListTile(
66.         leading: Icon(Icons.exit_to_app),
67.         title: Text('Sair'),
68.         onTap: () {
69.             print('Sair pressionado');
70.         },
71.     ),
72.     ],
73.   ),
74.   ),
75.   body: Center(
76.       child: Text('Conteúdo Principal'),
77.   ),
78.   ),
79. );
80. }
81. }
82.
83.
```

Os widgets basics, de layout, interativos e de apresentação são fundamentais na construção das interfaces dos aplicativos. O exemplo a seguir apresenta um aplicativo utilizando o widget **Scaffold**.

### Exemplo de código para o widget 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`.
- 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.
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.

```
52. // Pacote importado para criar a interface utilizando o
      Material Design
53. import 'package:flutter/material.dart';
54.
55. // Função principal do aplicativo, ponto de entrada da
      aplicação
56. void main() {
57.   // Inicializa o aplicativo e chama a classe MyApp, que
      define a interface do app
58.   runApp(MyApp());
59. }
60.
61. // Classe principal do aplicativo, do tipo
      StatelessWidget
62. // StatelessWidget significa que esta classe não possui
      estado dinâmico (imutável)
63. class MyApp extends StatelessWidget {
64.   @override
65.   Widget build(BuildContext context) {
66.     // Método build constrói a interface do aplicativo
67.     return MaterialApp(
68.       // Define a tela inicial do aplicativo
69.       home: Scaffold(
70.         // AppBar é a barra superior da tela
71.         appBar: AppBar(
72.           // Define o título exibido na AppBar
73.           title: Text("Meu App"),
74.           // Define a cor de fundo da AppBar
75.           backgroundColor: Colors.blue,
76.           // Define os ícones de ação na AppBar
77.           actions: [
78.             IconButton(
```

```
79.          // Ícone de busca
80.          icon: Icon(Icons.search),
81.          // Define a ação que será executada ao
     pressionar o botão
82.          onPressed: () {
83.              // Ação de busca (a ser implementada)
84.              print('Busca acionada');
85.          },
86.      ),
87.      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.
```

A figura a seguir mostra o app com o Scaffold.



**Figura 2.4 –** App com o widget Scaffold.

## Layouts de tela e estrutura

- Flutter utiliza uma abordagem baseada em widgets aninhados, em que cada elemento da UI é um widget.
- Estruturas comuns incluem a combinação de widgets de layout, como Scaffold, AppBar, Drawer e BottomNavigationBar.
- A árvore de widgets permite criar uma interface responsiva e reativa. Widgets como MediaQuery ajudam a adaptar a interface ao tamanho da tela.

### Tipos de layouts

- **Single-Child Layout:** contém um único widget filho, como Container e Center.
- **Multi-Child Layout:** contém múltiplos widgets filhos, como Column, Row, Stack e ListView.

A seguir, é possível visualizar o código exemplo do Widget Single Child Layout.

#### Exemplo de código para o Widget Single Child Layout

```
1. /*
2. 1. class MyApp extends
3. StatelessWidget
4.
5. MyApp é uma classe que estende StatelessWidget, que
   significa que ela não possui estado interno e não muda ao
   longo do tempo.
6. Esta classe é a estrutura principal do aplicativo e
   define a interface básica.
7.
8. 2. @override Widget build(BuildContext context)
9. 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.
10.
11. 3. MaterialApp
12. MaterialApp é um widget que configura o aplicativo para
   seguir o estilo Material Design, fornecendo temas,
   navegação, e outras funcionalidades globais.
13. A propriedade home define a tela inicial do aplicativo.
   Aqui, estamos passando um Scaffold como a tela inicial.
14.
15. 4. Scaffold
16. Scaffold é um widget de layout básico que fornece uma
   estrutura de página padrão com suporte para AppBar,
   Drawer, FloatingActionButton, entre outros.
17. Aqui, estamos usando Scaffold para estruturar a tela com
   uma barra de aplicativo (AppBar) e um corpo (body).
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.
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  
45. import 'package:flutter/material.dart';  
46.  
47. // Função principal do aplicativo, ponto de entrada da  
      aplicação  
48. void main() {  
49.   // Inicializa o aplicativo e chama a classe MyApp, que  
      define a interface do app  
50.   runApp(MyApp());  
51. }  
52.  
53. // Classe principal do aplicativo, do tipo  
      StatelessWidget  
54. // StatelessWidget significa que esta classe não possui  
      estado dinâmico (imutável)  
55. class MyApp extends StatelessWidget {  
56.   @override  
57.   Widget build(BuildContext context) {  
58.     // Método build constrói a interface do aplicativo  
59.     return MaterialApp(  
60.       // Define a tela inicial do aplicativo  
61.       home: Scaffold(  
62.         // AppBar é a barra superior da tela  
63.         appBar: AppBar(  
64.           // Define o título exibido na AppBar  
65.           title: Text("Meu App"),  
66.           // Define a cor de fundo da AppBar  
67.           backgroundColor: Colors.blue,  
68.           // Define os ícones de ação na AppBar  
69.           actions: [
```

```
70.          IconButton(
71.              // Ícone de busca
72.              icon: Icon(Icons.search),
73.              // Define a ação que será executada ao
74.              onPressed: () {
75.                  // Ação de busca (a ser implementada)
76.                  print('Busca acionada');
77.              },
78.          ),
79.          IconButton(
80.              // Ícone de “mais opções”
81.              icon: Icon(Icons.more_vert),
82.              // Define a ação que será executada ao
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
94.          // texto simples no centro
95.          ),
96.      ),
97.  );
98. }
99.
100.
```

A figura a seguir apresenta o app com o layout Single Child Layout.



**Figura 2.5 – Tela exemplo do Single Child Layout.**

A seguir é possível visualizar o código para o Multi Child Layout.

#### Exemplo de código para o Multi Child Layout

1. `/*`
2. `AppBar:`
- 3.
4. `Barra superior com o título “Exemplo de Multi-Child Layout”.`
5. `Column:`
- 6.
7. `Organiza os widgets verticalmente.`
8. `Propriedades:`
9. `crossAxisAlignment: Define o alinhamento horizontal dos filhos.`
10. `children: Contém os widgets filhos (Row, Stack, ListView, etc.).`
11. `Row:`
- 12.
13. `Organiza os widgets horizontalmente.`

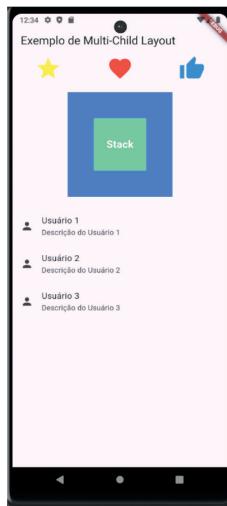
```
14. Propriedades:  
15. mainAxisAlignment: Espaça os widgets igualmente.  
16. Contém três ícones (star, favorite, thumb_up).  
17. SizedBox:  
18.  
19. Espaçamento entre widgets.  
20. Stack:  
21.  
22. Sobrepõe widgets uns aos outros.  
23. Contém dois Containers e um Text centralizado.  
24. 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. */  
33. import 'package:flutter/material.dart';  
34.  
35. void main() {  
36.   runApp(MyApp());  
37. }  
38.  
39. class MyApp extends StatelessWidget {  
40.   @override  
41.   Widget build(BuildContext context) {  
42.     return MaterialApp(  
43.       home: Scaffold(  
44.         // AppBar:  
45.         // Barra superior com o título “Exemplo de Multi-  
Child Layout”.
```

```
46.         appBar: AppBar(
47.             title: Text("Exemplo de Multi-Child Layout"),
48.         ),
49.         body: Column(
50.             // Column:
51.             // Organiza os widgets verticalmente.
52.             // Propriedades:
53.             // - mainAxisAlignment: Define o alinhamento
54.             // horizontal dos filhos.
55.             // - children: Contém os widgets filhos (Row,
56.             Stack, ListView, etc.).
57.             mainAxisAlignment: CrossAxisAlignment.stretch,
58.             children: [
59.                 // Row:
60.                 // Organiza os widgets horizontalmente.
61.                 // Propriedades:
62.                 // - mainAxisAlignment: Espaça os widgets
63.                 // igualmente.
64.                 // Contém três ícones (star, favorite,
65.                 thumb_up).
66.                 Row(
67.                     mainAxisAlignment: MainAxisAlignment.
68.                     spaceAround,
69.                     children: [
70.                         Icon(Icons.star, color: Colors.yellow,
71.                         size: 50),
72.                         Icon(Icons.favorite, color: Colors.red,
73.                         size: 50),
74.                         Icon(Icons.thumb_up, color: Colors.blue,
75.                         size: 50),
76.                     ],
77.                 ),
78.                 // SizedBox:
79.                 // Espaçamento entre widgets.
80.                 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.              ),  
89.              Text(  
90.                «Stack»,  
91.                style: TextStyle(  
92.                  color: Colors.white,  
93.                  fontSize: 20,  
94.                  fontWeight: FontWeight.bold,  
95.                ),  
96.                ),  
97.              ],  
98.            ),  
99.            // SizedBox:  
100.           // Espaçamento entre widgets.  
101.           SizedBox(height: 20),  
102.           // Expanded:  
103.           // Permite que o ListView preencha o espaço  
    disponível na tela.  
104.           Expanded(  
105.             // ListView:  
106.             // Um widget rolável que lista elementos.
```

```
107.           // Contém vários ListTile 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. }
```

A figura a seguir mostra a tela do aplicativo utilizando o Multi Child Layout.

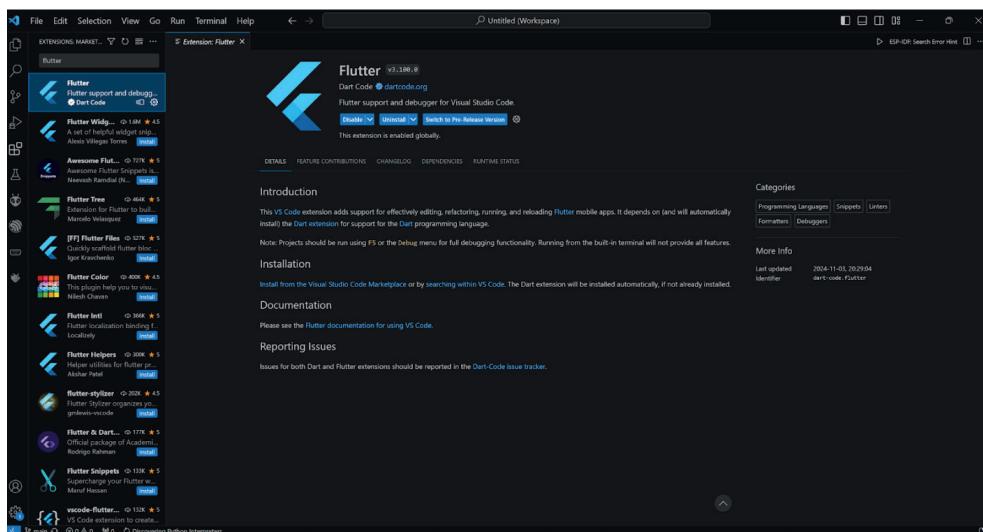


**Figura 2.6 – Tela do aplicativo Multi Child layout.**

Criação do primeiro aplicativo com o Flutter utilizando VSCode

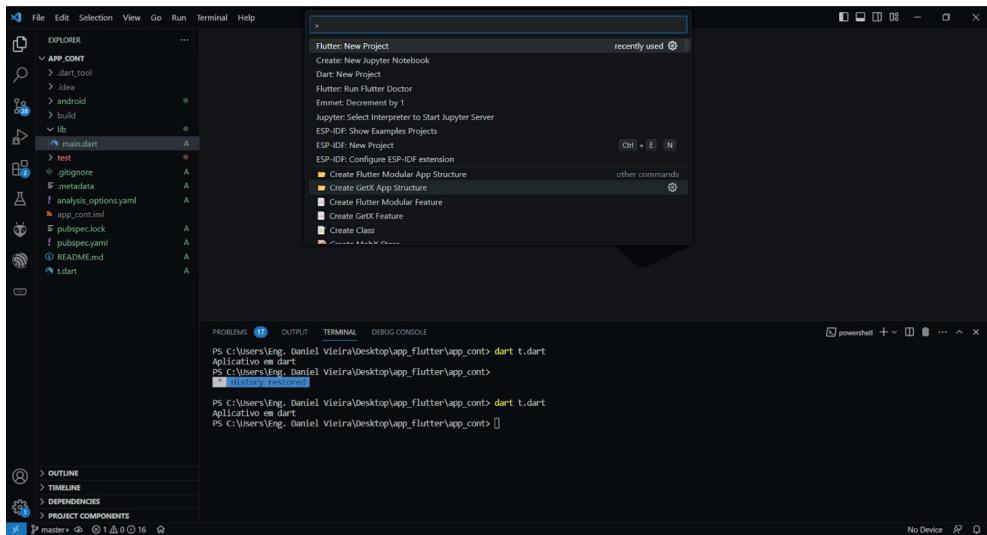
Para criar um projeto de aplicativo mobile utilizando o framework Flutter com a IDE VSCode, é necessário seguir os passos indicados:

1. Abrir o VSCode;
2. Instalar os plugins conforme a figura a seguir;



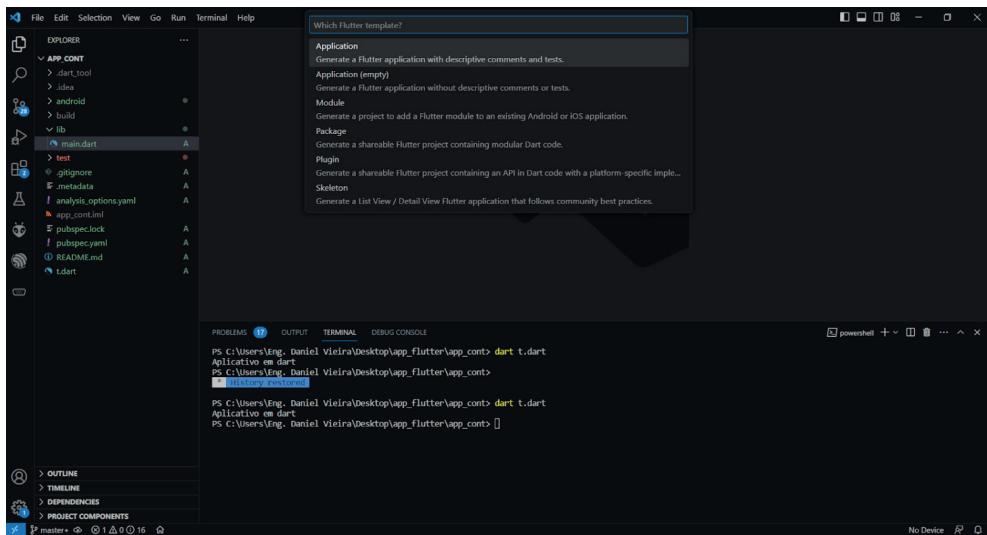
**Figura 2.7 – IDE VSCODE instalação das extensões Flutter e Dart.**

3. Apertar a tecla F1; em seguida teclar F1 e clicar em Flutter New Project, conforme a figura a seguir;



**Figura 2.8 – IDE VS CODE criando projeto Flutter.**

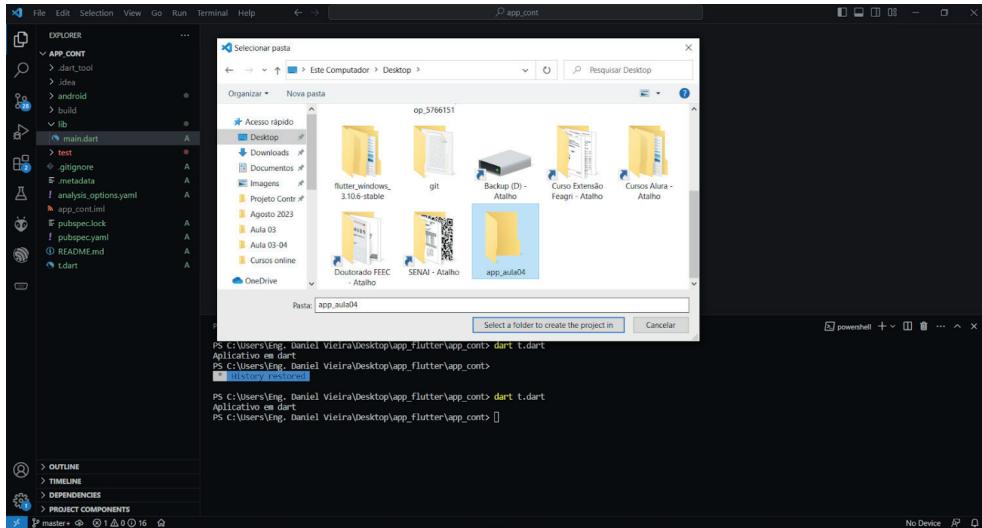
4. Selecionar a opção Application, conforme a figura a seguir;



**Figura 2.9 – IDE VS CODE criando projeto Flutter.**

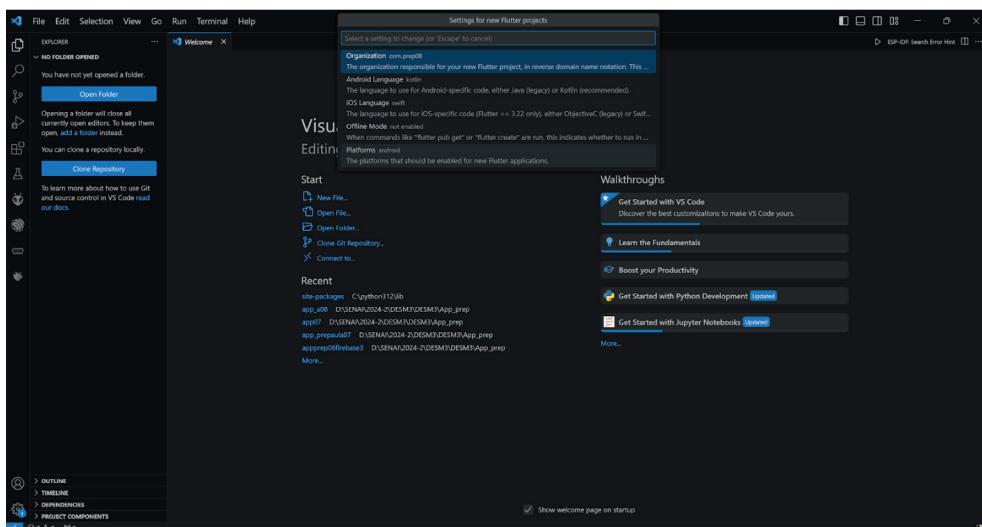
5. Selecionar uma pasta para salvar o projeto conforme a figura a seguir.

A pasta não deve começar com letras maiúsculas ou números e não pode haver espaço no nome da pasta;



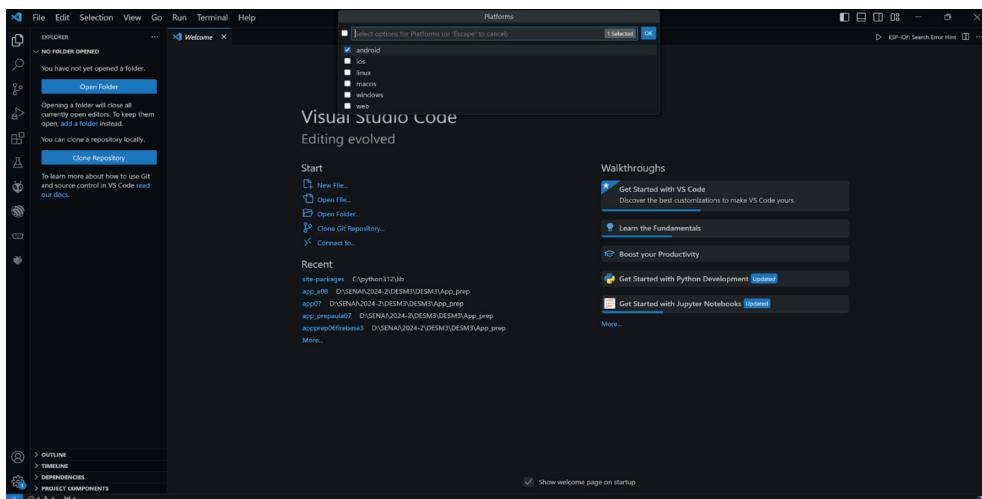
**Figura 2.10 – IDE VS CODE selecionando a pasta para salvar o projeto.**

6. Clicar na engrenagem para configurar as plataformas mobile para que o projeto seja executado conforme demonstrado na figura a seguir;



**Figura 2.11 – IDE VS CODE selecionando as plataformas do projeto.**

7. Selecionar a opção Android e, em seguida, teclar em Enter, conforme demonstrado na figura a seguir;

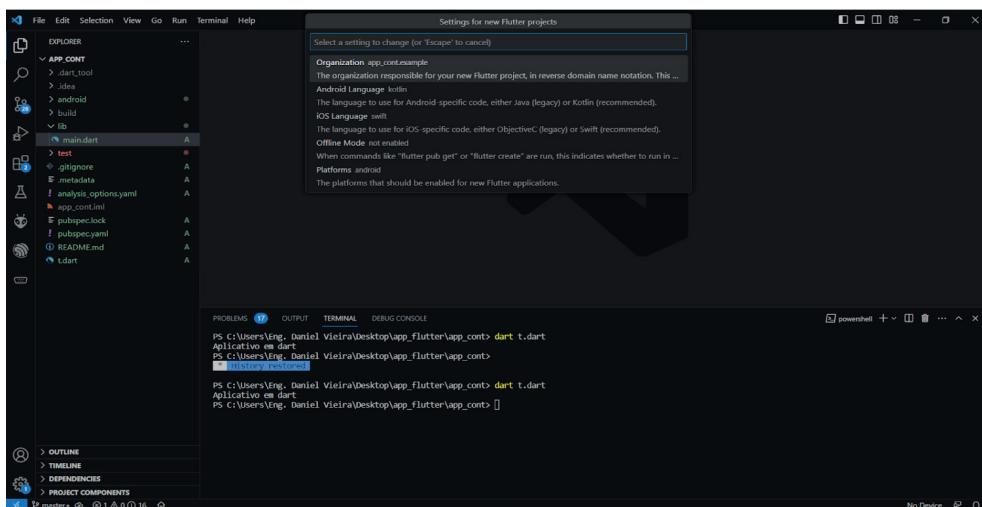


**Figura 2.12 – IDE VS CODE selecionando a plataforma Android.**

### CURIOSIDADE

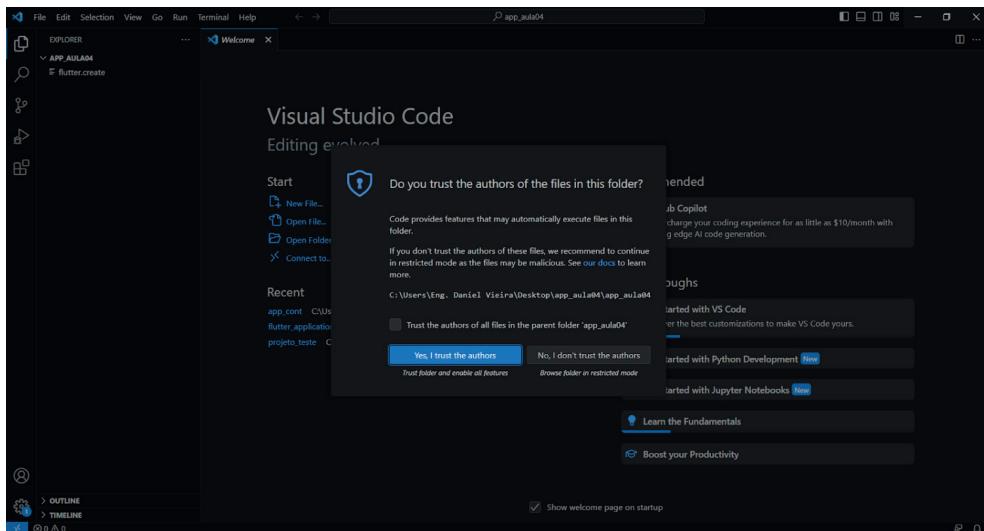
Organization é o parâmetro que indica o domínio da empresa que desenvolveu o app; é um parâmetro importante para publicação do APP em lojas virtuais.

8. Clicar em Organization, conforme mostra a figura a seguir.



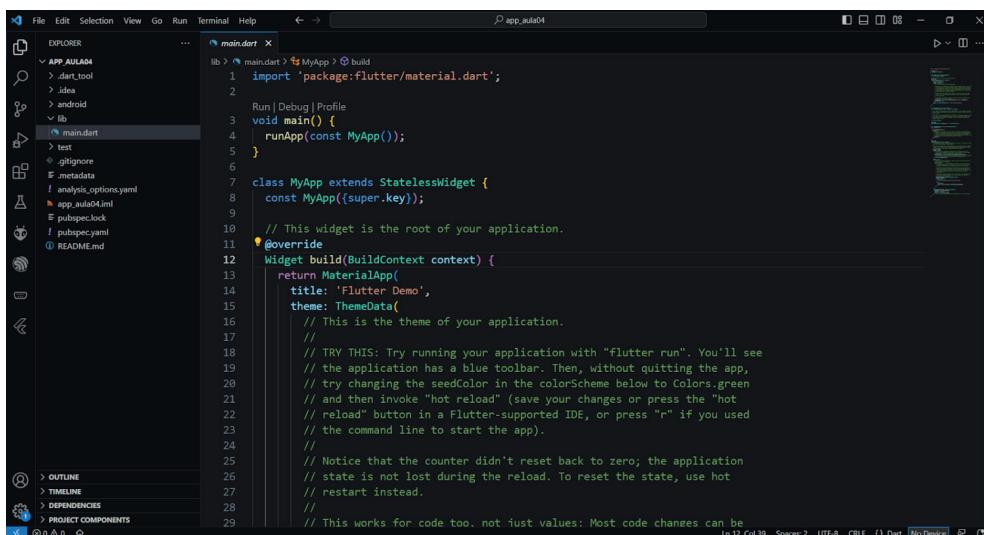
**Figura 2.13 – IDE VS CODE selecionando o organization.**

Após realizar as configurações, apertar Esc e Enter. O projeto Flutter será criado. Em seguida, marcar a opção Yes, I Trust the authors, conforme a figura a seguir.



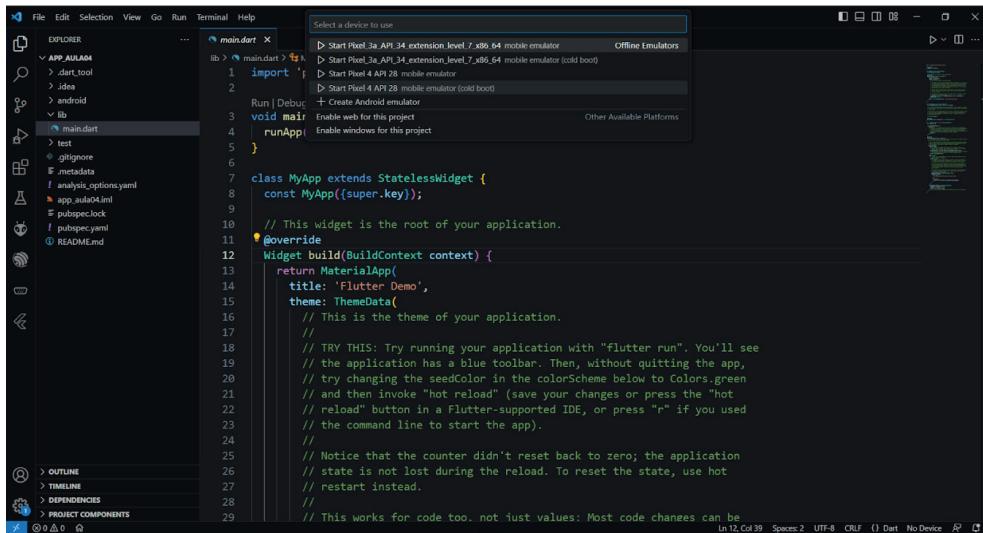
**Figura 2.14 – IDE VS CODE com projeto criado.**

Para executar o aplicativo no emulador, o primeiro passo é clicar em “escolheremos o device para emular nosso telefone”, clicar em No Device e selecionar o emulador desejado conforme figura a seguir.



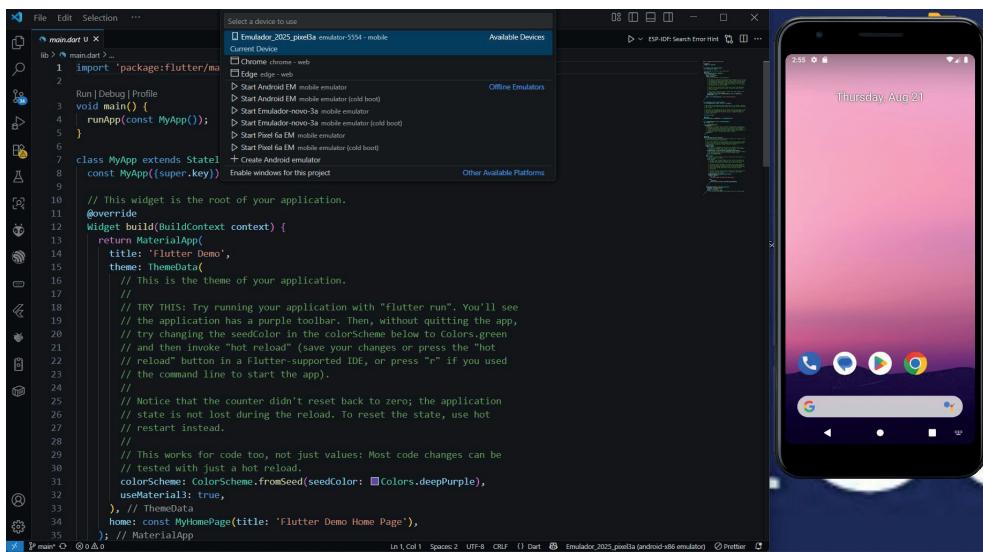
**Figura 2.15 – IDE VS CODE para escolha do emulador.**

Após clicar em No Device no editor VSCode, aparecerá a opção de escolha do emulador (conforme a figura a seguir), o qual executa o aplicativo. Caso não apareça nenhum emulador, então é necessário criá-lo. Para isso, consulte o item “Criando emulador para executar aplicativos”.



**Figura 2.16 – Escolha do emulador utilizando a IDE do VSCode.**

A figura a seguir mostra que, após selecionar o device, ele aparecerá na tela.



**Figura 2.17 – Visualização do device.**

Para executar o app, clicar em Run without debug e o aplicativo será aberto no emulador, conforme a figura a seguir.



**Figura 2.18** – Emulador Android executando aplicativo.

O código do aplicativo pode ser visualizado a seguir.

Código do aplicativo mostrado na Figura 2.18.

```
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. }
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. }
33. /*
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.  
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. */  
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.                  ),  
96. Row(  
97.                 mainAxisAlignment: MainAxisAlignment.  
spaceEvenly,  
98.                 crossAxisAlignment: CrossAxisAlignment.end,
```

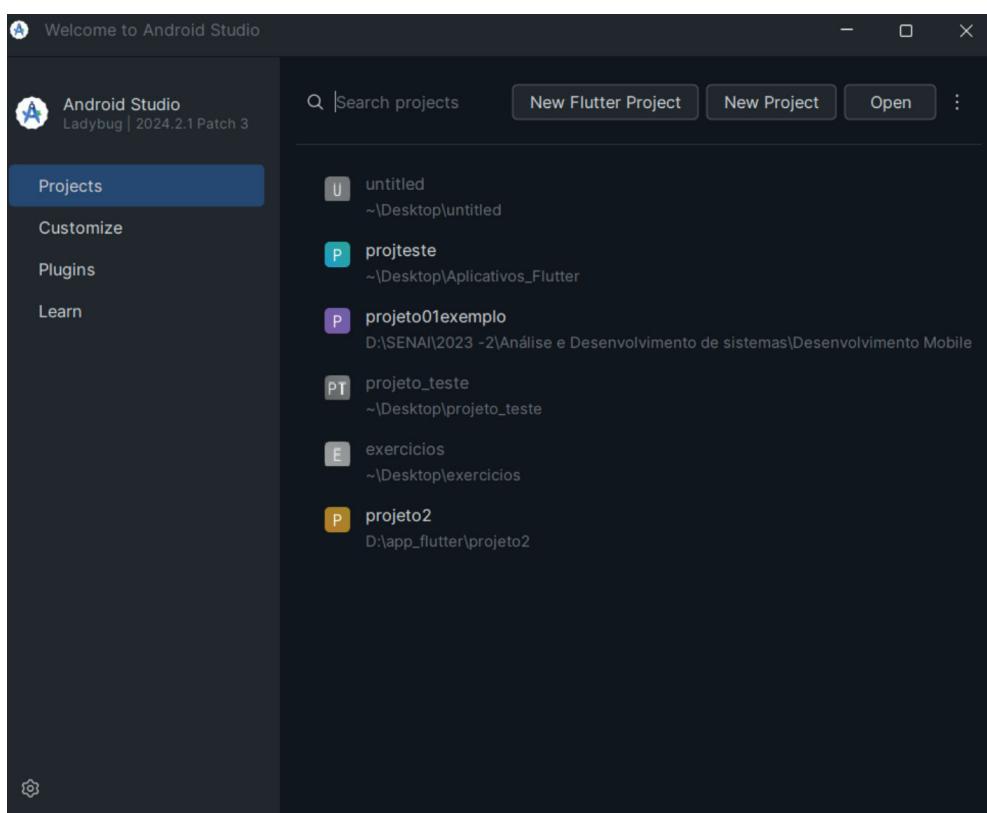
```
99.           children: [
100.             Container(color:Colors.blue,height:50,
101.               width:50),
102.             Container(color:Colors.green,height:50,
103.               width:50),
104.           ],
105.           Container(color:Colors.amber,height:50,width:300,
106.             child:Text(«Senai»,style:TextStyle(color:Colors.
107.               black,fontSize:28,
108.                 decoration:TextDecoration.none),textAlign:
109.                   TextAlign.center),
110.                 ),
111.               ElevatedButton(
112.                 onPressed: (){
113.                   print("Você apertou o botão 1");
114.                   },
115.                   child: Text("Botão 1"),),
116.                   ElevatedButton(
117.                     onPressed: _msg,
118.                     child: Text("Mensagem 2")),
119.                     ],
120.                     ),
121.                     ),
122.                     ),
123.                     ),
124.                     );
125.           );
126.         }
127.       }
128.
```

- 129.
- 130.
- 131.
- 132.
- 133.

## Criando emulador para executar aplicativos

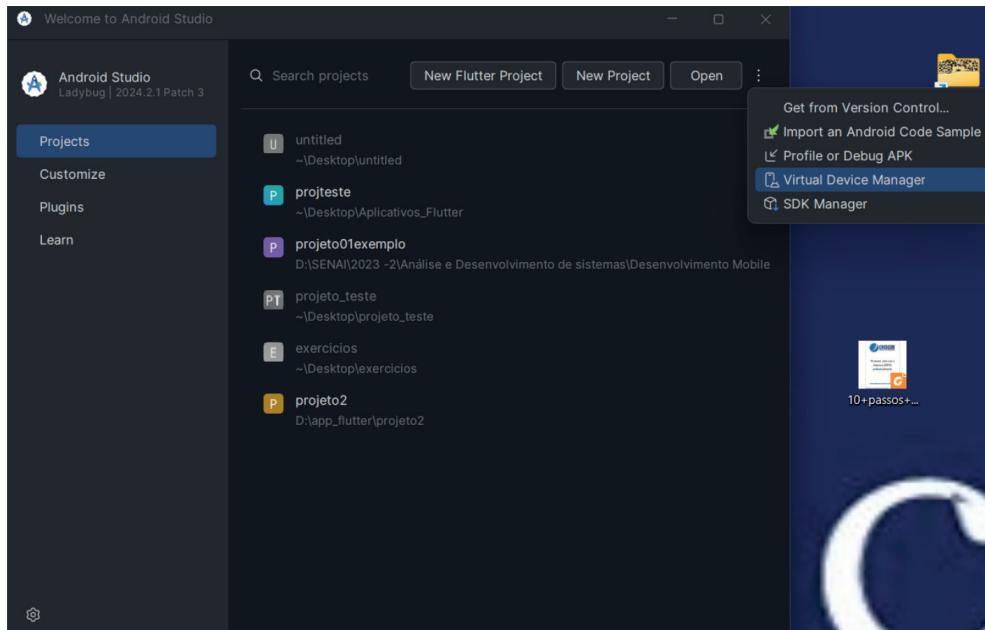
Uma forma muito eficiente e rápida de testar os aplicativos desenvolvidos é utilizar o emulador do Android Studio. Para isso, é necessário criá-lo no Android Studio, conforme indicado a seguir.

Abrir o Android Studio e clicar nos três pontos ao lado do botão Open.



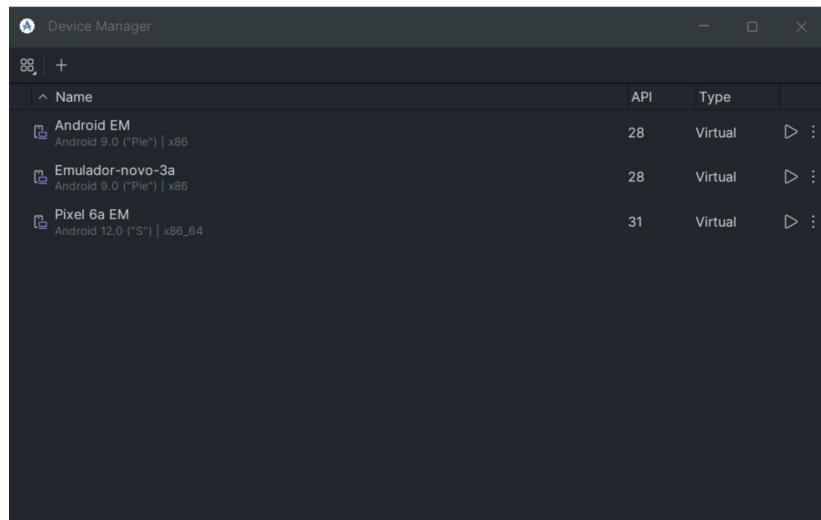
**Figura 2.19 – Tela Android Studio.**

Após clicar nos três pontos ao lado do botão Open, clique em Virtual Device Manager, conforme demonstrado na figura a seguir.



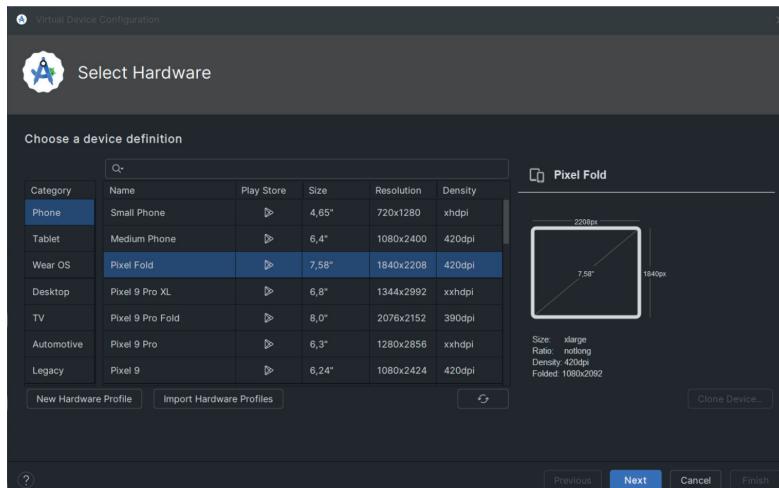
**Figura 2.20 – Selecionando o Virtual Device Manager.**

Após clicar em Virtual Device Manager, a tela com os emuladores criados aparecerá, conforme mostra a figura a seguir.



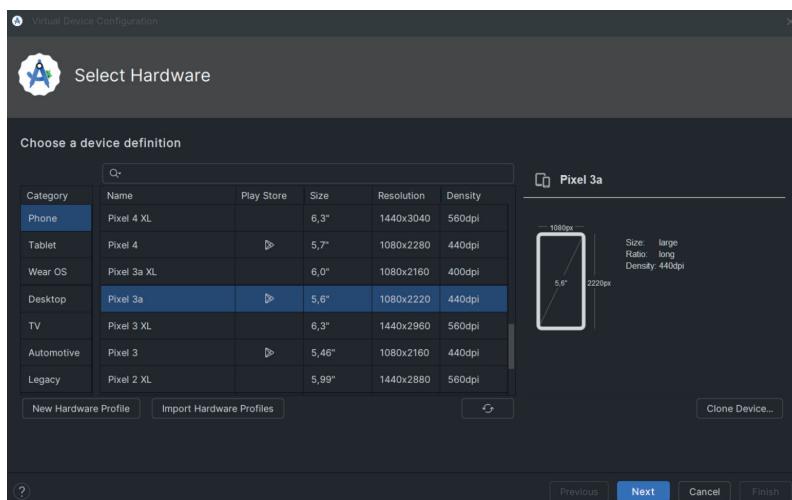
**Figura 2.21 – Emuladores criados.**

Para criar um novo emulador, é necessário clicar no ícone +. Após clicar, aparecerá a lista dos dispositivos emuladores disponíveis para serem escolhidos, conforme a figura a seguir.



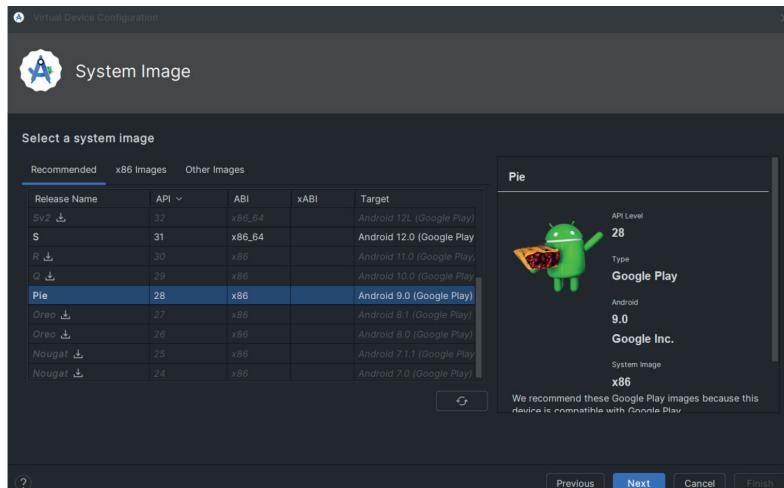
**Figura 2.22 – Emuladores disponíveis para serem escolhidos.**

Deve-se escolher o emulador Pixel 3a, conforme a Figura 2.22, ou qualquer outro emulador que esteja com o ícone da Play Store ativado, caso o objetivo seja hospedar o aplicativo na Play Store.



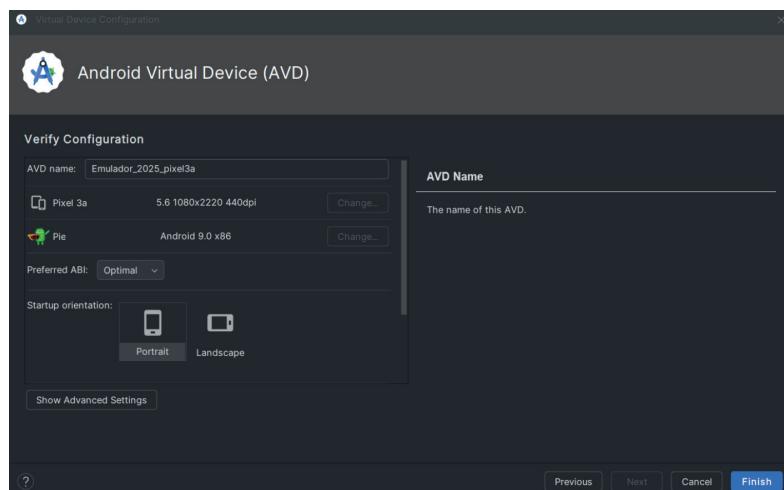
**Figura 2.23 – Tela para escolher o emulador Pixel 3a.**

Com o emulador Pixel 3a escolhido, o próximo passo será a escolha da versão do sistema operacional, conforme a figura a seguir. A versão escolhida será a versão Pie, por ser mais leve para ser executada em nosso dispositivo.



**Figura 2.24** – Escolha da versão do sistema operacional.

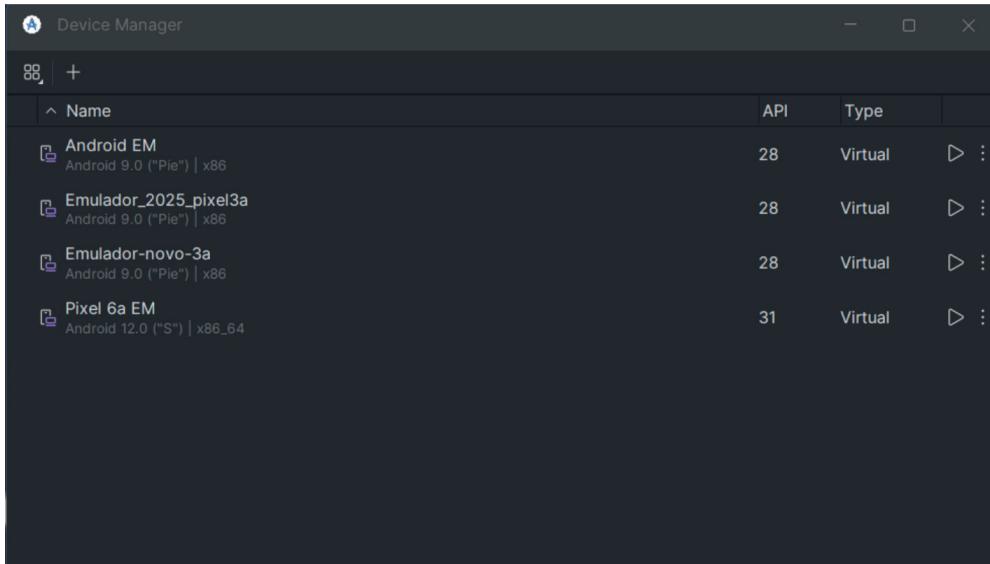
Clicar em Next para nomear o emulador, conforme demonstrado na figura a seguir.



**Figura 2.25** – Atribuindo um nome ao emulador Android.

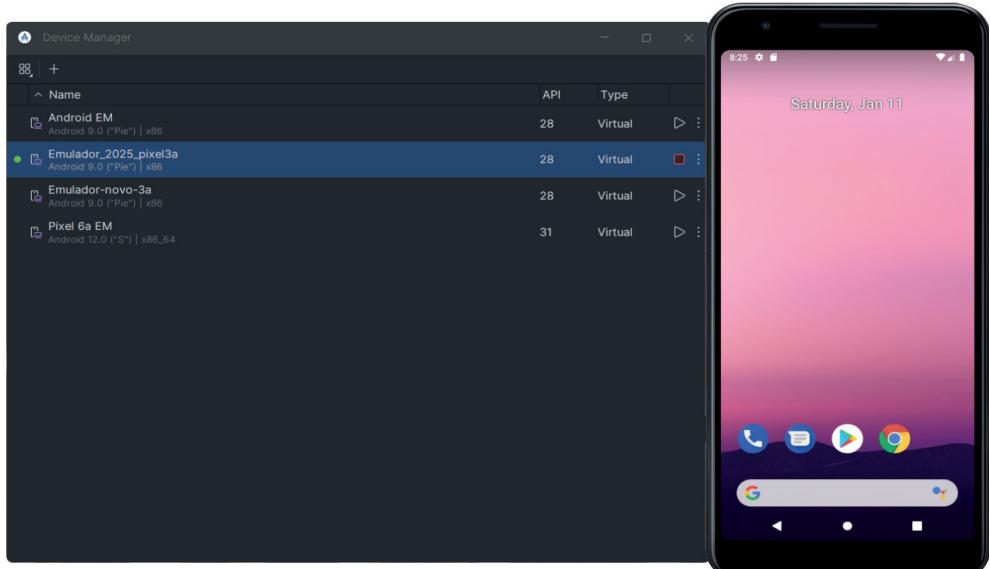
Após atribuir o nome Emulador\_2025\_pixel3a ao emulador, clicar em Finish para criá-lo.

Na figura seguinte, é possível visualizar o emulador criado; é só clicar em Play para o emulador ser inicializado e abrir a tela no celular.



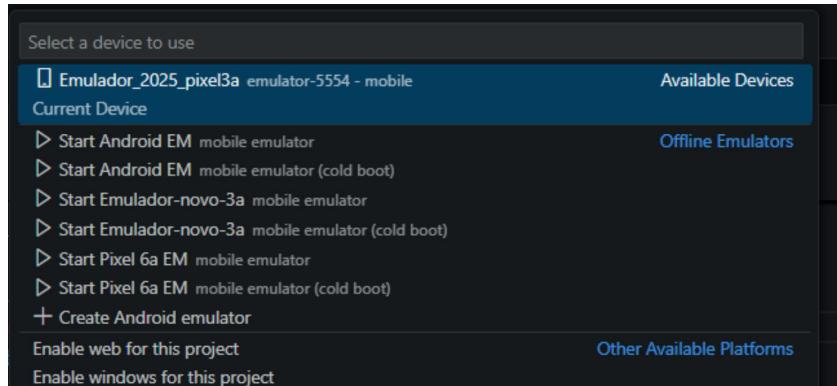
**Figura 2.26 – Emulador criado.**

Na figura a seguir, é possível ver o emulador sendo executado; agora, basta utilizá-lo para executar os aplicativos.



**Figura 2.27 – Emulador sendo executado.**

Com o emulador criado no Android Studio, feche esse software e abra o VSCode. Em seguida, clique em No Device e o emulador novo criado aparecerá como opção, conforme a figura a seguir.



**Figura 2.28** – Emulador disponível no VSCode.

Pronto! O emulador criado para executar os nossos aplicativos está disponível.

A seguir trataremos de um tema muito importante, que é o gerenciamento de estados em aplicativos móveis.

## Gerenciadores de estado

O gerenciamento de estado é crucial para sincronizar a interface com os dados subjacentes. Seus principais métodos são:

- **stateful widgets:** mantém um estado interno que muda ao longo do tempo;
- **provider:** biblioteca que facilita o gerenciamento de estado e a comunicação entre widgets;
- **riverpod, BLoC e GetX:** outras opções avançadas para aplicações de grande porte.

## StatefulWidget

Os StatefulWidget são a abordagem mais básica e integrada ao Flutter para gerenciar estados que mudam ao longo do tempo. Seu funcionamento ocorre da seguinte forma:

- o estado é armazenado dentro do widget;
- ele usa o método setState para notificar o Flutter de que algo mudou e precisa ser reconstruído.

A seguir, é possível visualizar o código do gerenciador de estado StatefulWidget.

### Código StatefulWidget

```
1. /*
2. CounterApp: É um widget do tipo StatefulWidget. Isso
   significa que ele tem um estado interno que pode ser
   alterado durante a execução.
3. createState: Cria uma instância do estado associado a este
   widget (_CounterAppState), onde a lógica e os dados são
   definidos.
4.
5. _CounterAppState: É o estado da aplicação e contém a
   lógica e os dados do widget.
6. counter: É uma variável que armazena o valor atual do
   contador (inicializado como 0).
7.
8. increment: Incrementa o valor do contador (counter).
9. setState: Notifica o Flutter que o estado mudou. Isso faz
   com que o widget seja reconstruído e a interface seja
   atualizada.
10.
11. build: Descreve como o widget deve ser exibido.
12. Scaffold: Estrutura básica para o layout da tela (inclui
   suporte a barra de título, corpo, etc.).
```

```
13. Center: Centraliza os widgets no eixo principal.  
14. Column: Organiza os widgets filhos em uma coluna.  
15. mainAxisAlignment: MainAxisAlignment.center: Centraliza os  
    elementos verticalmente.  
16. Text: Mostra o valor atual do contador.  
17. ElevatedButton: Um botão elevado.  
18. onPressed: increment: Chama o método increment quando o  
    botão é pressionado.  
19. child: Define o texto exibido no botão (“Incrementar”).  
20.  
21. */  
22. class CounterApp extends StatefulWidget {  
23.   @override  
24.   _CounterAppState createState() => _CounterAppState();  
25. }  
26.  
27. class _CounterAppState extends State<CounterApp> {  
28.   int counter = 0;  
29.  
30.   void increment() {  
31.     setState(() {  
32.       counter++;  
33.     });  
34.   }  
35.  
36.   @override  
37.   Widget build(BuildContext context) {  
38.     return Scaffold(  
39.       body: Center(  
40.         child: Column(  
41.           mainAxisAlignment: MainAxisAlignment.center,  
42.           children: [  
43.             Text('Contador: $counter'),  
44.             ElevatedButton(  
45.               onPressed: increment,  
46.               child: Text('Incrementar'),
```

```

47.           ),
48.           ],
49.           ),
50.           ),
51.       );
52.   }
53. }
54.
55.

```

**Vantagens:**

- fácil de implementar;
- ideal para estados locais (widgets isolados).

**Desvantagens:**

- difícil de escalar para aplicações grandes;
- pode causar reconstruções desnecessárias em widgets filhos.

## Provider

O Provider é uma biblioteca oficial do Flutter que facilita o compartilhamento de estados entre widgets. Seu funcionamento ocorre da seguinte forma:

- cria um estado global ou escopo específico que pode ser acessado por widgets;
- usa ChangeNotifier para notificar mudanças.

A seguir, é possível visualizar o código do gerenciador de estado Provider.

**Código Provider**

```

1. /*
2. CounterProvider: É a classe que gerencia o estado do
   contador. Ela herda de ChangeNotifier, que permite notificar
   os widgets dependentes quando há mudanças no estado.

```

```
3. counter: Variável que armazena o valor do contador.
4. increment: Método que incrementa o valor do contador e
   chama notifyListeners().
5. notifyListeners: Notifica todos os widgets ouvintes
   que houve uma alteração no estado, desencadeando a
   reconstrução desses widgets.
6.
7. ChangeNotifierProvider: É um widget do pacote provider que
   disponibiliza uma instância de CounterProvider para todos
   os widgets filhos.
8. create: Cria a instância de CounterProvider.
9. child: O widget filho que terá acesso ao CounterProvider
   (neste caso, o MyApp).
10.
11. final counterProvider = Provider<
        ChangeNotifierProvider<CounterProvider>>(context);
12. Obtém a instância de CounterProvider disponibilizada pelo
    ChangeNotifierProvider.
13. Essa instância é usada para acessar o estado e os métodos
    do provedor.
14.
15.
16. Estrutura da Interface
17.
18. MaterialApp: Configura o aplicativo.
19. Scaffold: Proporciona a estrutura básica da tela.
20. Center: Centraliza os widgets no eixo principal.
21. Column: Organiza os widgets em uma coluna.
22. Text: Exibe o valor atual do contador acessado de
   counterProvider.counter.
23. ElevatedButton:
24. onPressed: counterProvider.increment: Chama o método
   increment do provedor para atualizar o estado.
25. */
26. class CounterProvider extends ChangeNotifier {
```

```
27.     int counter = 0;
28.
29.     void increment() {
30.         counter++;
31.         notifyListeners();
32.     }
33. }
34.
35. void main() {
36.     runApp(
37.         ChangeNotifierProvider(
38.             create: (_) => CounterProvider(),
39.             child: MyApp(),
40.         ),
41.     );
42. }
43.
44. class MyApp extends StatelessWidget {
45.     @override
46.     Widget build(BuildContext context) {
47.         final counterProvider = Provider.
48.             of<CounterProvider>(context);
49.
50.         return MaterialApp(
51.             home: Scaffold(
52.                 body: Center(
53.                     child: Column(
54.                         mainAxisAlignment: MainAxisAlignment.center,
55.                         children: [
56.                             Text('Contador: ${counterProvider.counter}'),
57.                             ElevatedButton(
58.                                 onPressed: counterProvider.increment,
59.                                 child: Text('Incrementar'),
60.                             ),
61.                         ],
62.                     ),
63.                 ),
64.             ),
65.         );
66.     }
67. }
```

```

61.           ),
62.           ),
63.           ),
64.       );
65.   }
66. }
67.

```

### **Vantagens:**

- escalável para projetos de médio porte;
- simples de integrar e usar com widgets Flutter;
- controla reconstruções de forma eficiente.

### **Desvantagens:**

- curva de aprendizado inicial para novos desenvolvedores;
- pode ficar confuso em projetos muito grandes.

## Riverpod

O Riverpod é uma evolução do Provider, projetado para resolver limitações e melhorar a experiência. Seu funcionamento ocorre da seguinte forma:

- não depende diretamente do contexto (sem BuildContext);
- usa provedores declarativos.

### **Vantagens:**

- melhor performance que o Provider;
- mais flexível e escalável;
- adequado para grandes projetos.

### **Desvantagens:**

- curva de aprendizado um pouco mais alta;
- não é a biblioteca oficial do Flutter.

## BLoC (Business Logic Component)

O BLoC implementa o padrão de arquitetura reativa para separar lógica de negócios da interface. Seu funcionamento ocorre da seguinte forma:

- usa Streams para transmitir estados;
- eventos acionam mudanças no estado.

A seguir, é possível visualizar o código do gerenciador de estado Bloc.

### Gerenciamento de estado com Bloc

```

1. /*
2. CounterBloc: Classe que encapsula a lógica de negócios
   para o contador.
3. _counterController: Um StreamController que gerencia um
   fluxo de dados (Stream) do tipo int.
4. É usado para emitir valores do contador para os ouvintes
   (listeners).
5. counter: Uma variável que mantém o valor atual do
   contador, iniciada em 0.
6.
7. counterStream: Um getter que expõe o fluxo de valores
   (Stream) do _counterController.
8. Widgets podem ouvir este fluxo para receber atualizações
   do valor do contador em tempo real.
9.
10. Incrementa o contador:
11. O valor de counter é aumentado em 1.
12. Emite o novo valor:
13. O método sink.add(counter) envia o novo valor do contador
   para todos os ouvintes do fluxo (counterStream).
14. */
15. class CounterBloc {
16.   final _counterController = StreamController<int>();
17.   int counter = 0;

```

```

18.
19.     Stream<int> get counterStream => _counterController.
    stream;
20.
21.     void increment() {
22.         counter++;
23.         _counterController.sink.add(counter);
24.     }
25.
26.     void dispose() {
27.         _counterController.close();
28.     }
29.
30.

```

### Vantagens:

- excelente para aplicativos grandes e complexos;
- facilita testes unitários;
- padroniza a separação de camadas.

### Desvantagens:

- pode ser verboso e complicado para pequenos projetos;
- requer compreensão avançada de Streams.

## GetX

O GetX é uma biblioteca simples e eficiente que gerencia estado, rotas e dependências. Seu funcionamento ocorre da seguinte forma:

- gerencia estado reativo com observáveis;
- não exige contexto para acessar estados.

A seguir, é possível visualizar o código do gerenciador de estado GetX.

## Código gerenciador de estado GetX

```
1. /*
2. CounterController: Classe que gerencia o estado do
   contador, estendendo GetxController, que é a classe base
   para controladores no GetX.
3.
4. counter: Uma variável reativa (RxInt) inicializada com o
   valor 0. O .obs transforma o valor em um objeto reativo,
   que notifica automaticamente os widgets ouvintes sobre
   mudanças no valor.
5.
6. increment: Método que incrementa o valor do contador.
   Como o counter é reativo, qualquer alteração será refletida
   na interface do usuário automaticamente.
7.
8. GetMaterialApp: Uma versão de MaterialApp fornecida
   pelo GetX. Ela é necessária para que o GetX funcione
   corretamente, como para navegação e gerenciamento de
   dependências.
9. home: Define o widget inicial da aplicação (MyApp).
10.
11. Componentes
12. Get.put(CounterController()):
13.
14. Injeta uma instância do CounterController no sistema de
   dependências do GetX.
15. Torna o controlador acessível em qualquer lugar do
   aplicativo onde ele for requisitado.
16. Aqui, a instância é armazenada em controller.
17. Obx:
18.
19. Um widget reativo fornecido pelo GetX que reconstrói
   automaticamente a interface sempre que a variável
   observada (counter) mudar.
20. No exemplo, o Text é reconstruído sempre que counter for
   atualizado.
```

```
21. Estrutura da Interface:  
22.  
23. Scaffold: Fornece a estrutura básica da tela.  
24. Center: Centraliza os widgets no eixo principal.  
25. Column:  
26. Exibe o valor atual do contador usando Text.  
27. Exibe um botão ElevatedButton que chama o método  
    increment para atualizar o contador.  
28. */  
29. class CounterController extends GetxController {  
30.   var counter = 0.obs;  
31.  
32.   void increment() {  
33.     counter++;  
34.   }  
35. }  
36.  
37. void main() {  
38.   runApp(GetMaterialApp(home: MyApp()));  
39. }  
40.  
41. class MyApp extends StatelessWidget {  
42.   final CounterController controller = Get.  
      put(CounterController());  
43.  
44.   @override  
45.   Widget build(BuildContext context) {  
46.     return Scaffold(  
47.       body: Center(  
48.         child: Column(  
49.           mainAxisAlignment: MainAxisAlignment.center,  
50.           children: [  
51.             Obx(() => Text('Contador: ${controller.  
              counter}')),  
52.             ElevatedButton(  
53.               onPressed: controller.increment,
```

```

54.           child: Text('Incrementar'),
55.           ),
56.           ],
57.           ),
58.           ),
59.           );
60.       }
61.   }
62.

```

### Vantagens:

- sintaxe simples e intuitiva;
- muito eficiente em gerenciamento de estados reativos;
- integração de rotas e dependências.

### Desvantagens:

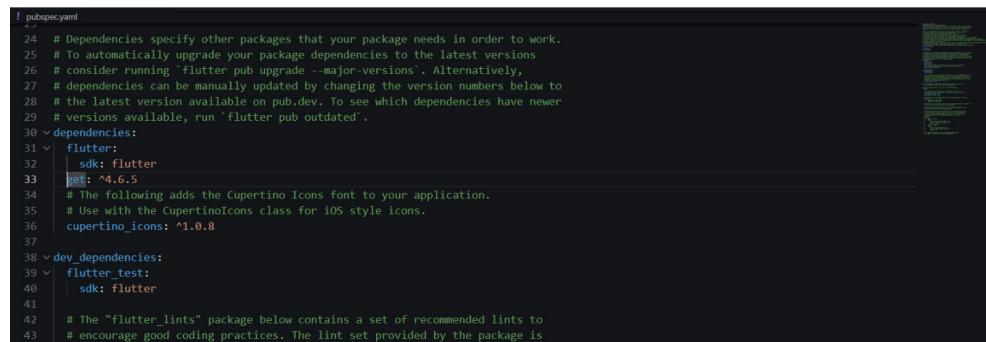
- uso indiscriminado pode levar a tight coupling (forte acoplamento);
- alguns desenvolvedores consideram menos adequado para projetos extremamente grandes.

Na tabela a seguir, é possível visualizar uma comparação entre as vantagens e desvantagens de cada gerenciador de estados, suas aplicações e o método implementado.

**Tabela 2.1 – Comparação entre os gerenciadores de estado**

Método	Aplicação	Vantagens	Desvantagens
Stateful	Estados locais pequenos.	Simples de usar.	Difícil de escalar.
Provider	Projetos médios.	Oficial e fácil de integrar.	Complexo em projetos grandes.
Riverpod	Projetos grandes.	Flexível e eficiente.	Curva de aprendizado maior.
BloC	Aplicações complexas.	Padronizável e testável.	Verboso para projetos pequenos.
GetX	Rápida prototipagem, reatividade.	Simples e poderoso.	Acoplamento forte em uso excessivo.

Para utilizar o gerenciador de estado Getx, é necessário adicionar sua dependência ao projeto, conforme mostra a figura a seguir.



```
! pubspec.yaml
24 # Dependencies specify other packages that your package needs in order to work.
25 # To automatically upgrade your package dependencies to the latest versions
26 # consider running `flutter pub upgrade --major-versions`. Alternatively,
27 # dependencies can be manually updated by changing the version numbers below to
28 # the latest version available on pub.dev. To see which dependencies have newer
29 # versions available, run `flutter pub outdated`.
30 dependencies:
31   flutter:
32     sdk: flutter
33   get: ^4.6.5
34   # The following adds the Cupertino Icons font to your application.
35   # Use with the CupertinoIcons class for iOS style icons.
36   cupertino_icons: ^1.0.8
37
38 dev_dependencies:
39   flutter_test:
40     sdk: flutter
41
42   # The "flutter_lints" package below contains a set of recommended lints to
43   # encourage good coding practices. The lint set provided by the package is
```

**Figura 2.29** – Adição de dependência ao projeto.

A seguir, é possível visualizar o código do gerenciador de estado GetX.

### Código utilizando o gerenciador de estado Getx

1. `/*`
2. `SumController:`
- 3.
4. `Gerencia o estado da aplicação.`
5. `Observáveis (obs) são usados para monitorar mudanças nos números (number1, number2) e no resultado (result).`
6. `Injeção do Controlador:`
- 7.
8. `O Get.put(SumController()) cria e disponibiliza o controlador para os widgets.`
9. `Reatividade com Obx:`
- 10.
11. `Atualiza automaticamente a interface sempre que o valor de result muda.`
12. `TextField:`
- 13.
14. `Permite ao usuário digitar números.`

```
15. O método setNumber1 ou setNumber2 converte o valor  
    digitado em double e atualiza o estado.  
16. Botão de Soma:  
17.  
18. Chama o método calculateSum no controlador para calcular  
    a soma e atualizar o resultado.  
19. */  
20. import 'package:flutter/material.dart';  
21. import 'package:get/get.dart';  
22.  
23. void main() {  
24.   runApp(GetMaterialApp(  
25.     home: SumApp(),  
26.   ));  
27. }  
28.  
29. class SumController extends GetxController {  
30.   var number1 = 0.0.obs; // Observável para o primeiro  
    número  
31.   var number2 = 0.0.obs; // Observável para o segundo  
    número  
32.   var result = 0.0.obs; // Observável para o resultado da  
    soma  
33.  
34.   void setNumber1(String value) {  
35.     number1.value = double.tryParse(value) ?? 0.0; //  
      Atualiza o número 1  
36.   }  
37.  
38.   void setNumber2(String value) {  
39.     number2.value = double.tryParse(value) ?? 0.0; //  
      Atualiza o número 2
```

```
40.    }
41.
42.    void calculateSum() {
43.        result.value = number1.value + number2.value; //
    Calcula a soma
44.    }
45. }
46.
47. class SumApp extends StatelessWidget {
48.     final SumController controller =
49.         Get.put(SumController()); // Injeta o controlador
50.
51.     @override
52.     Widget build(BuildContext context) {
53.         return Scaffold(
54.             appBar: AppBar(
55.                 title: Text('Somar Números - GetX'),
56.             ),
57.             body: Padding(
58.                 padding: const EdgeInsets.all(16.0),
59.                 child: Column(
60.                     mainAxisAlignment: MainAxisAlignment.center,
61.                     children: [
62.                         TextField(
63.                             keyboardType: TextInputType.number,
64.                             decoration: InputDecoration(
65.                                 labelText: 'Número 1',
66.                                 border: OutlineInputBorder(),
67.                             ),
68.                             onChanged: (value) =>
69.                                 controller.setNumber1(value), // Atualiza
    número 1
70.                         ),
```

```
71.           SizedBox(height: 16),  
72.           TextFormField(  
73.             keyboardType: TextInputType.number,  
74.             decoration: InputDecoration(  
75.               labelText: 'Número 2',  
76.               border: OutlineInputBorder(),  
77.             ),  
78.             onChanged: (value) =>  
79.               controller.setNumber2(value), // Atualiza  
    número 2  
80.           ),  
81.           SizedBox(height: 16),  
82.           ElevatedButton(  
83.             onPressed: controller.calculateSum, //  
    Realiza a soma  
84.             child: Text('Somar'),  
85.           ),  
86.           SizedBox(height: 16),  
87.           Obx(() => Text(  
88.             'Resultado: ${controller.result}', //  
    Exibe o resultado  
89.             style: TextStyle(fontSize: 20,  
90.               fontWeight: FontWeight.bold),  
91.             )),  
92.           ],  
93.         ),  
94.       );  
95.     }  
96.   }  
97.  
98.
```

Na figura a seguir, é possível visualizar o aplicativo com o gerenciador de estados GetX.



**Figura 2.30** – Aplicativo com o Gerenciador de estados GetX.

## Recursos e interfaces

Os recursos incluem imagens, fontes e temas, que são organizados no arquivo pubspec.yaml.

**ThemeData:** Define cores, fontes e estilos de widget para manter uma aparência consistente.

A seguir, é possível visualizar o código personalizando o tema do aplicativo, estilizando a fonte, tamanho etc.

### Código personalizando o ThemeData

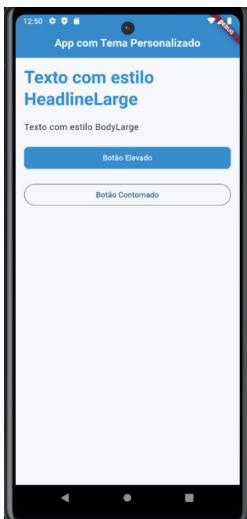
1. `/*`
2. `ThemeData:`
- 3.
4. `Define o tema global do aplicativo, incluindo esquema de cores, estilos de texto e configurações para widgets como AppBar e botões.`

```
5. TextTheme:  
6.  
7. Personaliza estilos de texto globais que podem ser  
    aplicados em diferentes partes do aplicativo.  
8. AppBarTheme:  
9.  
10. Configura a aparência da barra de aplicativo (AppBar), como  
    cor de fundo e estilo do título.  
11. ElevatedButtonThemeData:  
12.  
13. Personaliza os botões elevados (ElevatedButton), incluindo  
    cor e bordas arredondadas.  
14. Página Inicial:  
15.  
16. Demonstra como usar estilos definidos no tema para textos  
    e botões.  
17. */  
18. // Importa o pacote Material do Flutter, que fornece os  
    widgets baseados no Material Design  
19. import 'package:flutter/material.dart';  
20.  
21. // Função principal do aplicativo, ponto de entrada  
22. void main() {  
23.     // Inicializa o aplicativo e chama a classe MyApp  
24.     runApp(MyApp());  
25. }  
26.  
27. // Classe principal do aplicativo, do tipo StatelessWidget  
28. // StatelessWidget indica que esta interface não possui  
    estado dinâmico  
29. class MyApp extends StatelessWidget {  
30.     @override  
31.     Widget build(BuildContext context) {  
32.         return MaterialApp(  
            // Personalização do tema do aplicativo
```

```
34.     theme: ThemeData(  
35.         // Ativa o Material Design 3 para usar os recursos  
36.         // mais recentes  
37.         useMaterial3: true,  
38.         // Define o esquema de cores com base em uma cor  
39.         // principal  
40.         colorScheme: ColorScheme.fromSeed(seedColor:  
41.             Colors.blue),  
42.             // Configura o tema de texto global  
43.             textTheme: TextTheme(  
44.                 headlineLarge: TextStyle(  
45.                     fontSize: 32, // Tamanho da fonte para títulos  
46.                     grandes  
47.                     fontWeight: FontWeight.bold, // Peso da fonte  
48.                     em negrito  
49.                     color: Colors.blue, // Cor do texto  
50.                 ),  
51.                 bodyLarge: TextStyle(  
52.                     fontSize: 16, // Tamanho da fonte para texto  
53.                     padrão  
54.                     color: Colors.black87, // Cor do texto  
55.                 ),  
56.             ),  
57.             // Configura o tema da AppBar  
58.             appBarTheme: AppBarTheme(  
59.                 backgroundColor: Colors.blue, // Cor de fundo da  
60.                 AppBar  
61.                 centerTitle: true, // Centraliza o título na  
62.                 AppBar  
63.                 elevation: 4, // Define a sombra da AppBar  
64.                 titleTextStyle: TextStyle(  
65.                     fontSize: 20, // Tamanho da fonte do título  
66.                     fontWeight: FontWeight.bold, // Peso da fonte  
67.                     color: Colors.white, // Cor do texto  
68.                 ),
```

```
61.          ),
62.          // Configura o tema para botões elevados
63.          (ElevatedButton)
64.          elevatedButtonTheme: ElevatedButtonThemeData(
65.              style: ElevatedButton.styleFrom(
66.                  backgroundColor: Colors.blue, // Cor de fundo
67.                      do botão
68.                  foregroundColor: Colors.white, // Cor do texto
69.                      do botão
70.                  shape: RoundedRectangleBorder(
71.                      borderRadius: BorderRadius.circular(8), //
72.                          Bordas arredondadas
73.                  ),
74.                  ),
75.          ); }
```

A seguir, é possível visualizar o aplicativo com o tema personalizado.

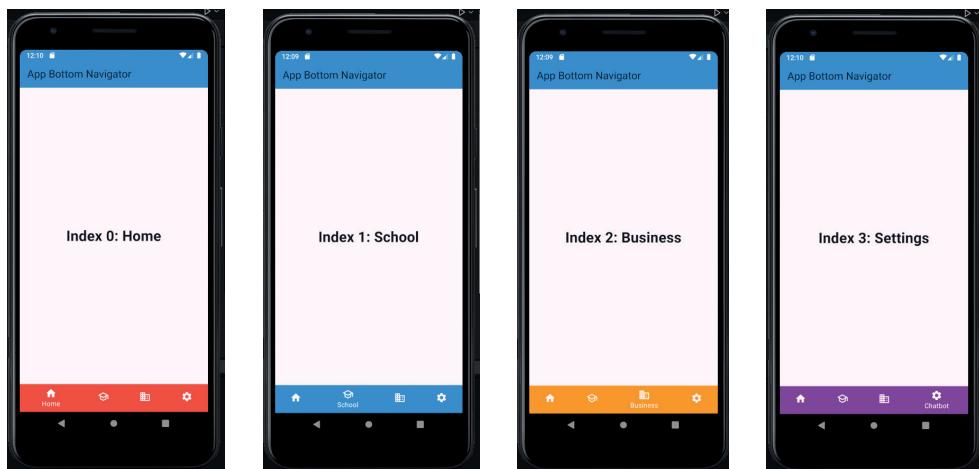


**Figura 2.31** – Aplicativo com tema personalizado.

## Componentes de tela: menus, diálogos e barra de ação

- **Menu:** Drawer para menus laterais, BottomNavigationBar para navegação inferior.
- **Diálogos:** AlertDialog para mensagens simples, Snackbar para notificações breves.
- **AppBar:** barra superior que permite adicionar ações, ícones de navegação e títulos.

Na figura a seguir, é apresentado o aplicativo utilizando o componente BottomNavigationBar.



**Figura 2.32 – Emulador Android executando aplicativo com campo texto.**

O código utilizado no aplicativo com o BottomNavigator pode ser visualizado a seguir.

Código do aplicativo com BottonNavigator

```
1. /*  
2. Função main():
```

```
3. Inicializa o aplicativo chamando runApp() e passando o
   widget NavButton como argumento.
4. */
5. import 'package:flutter/material.dart';
6.
7. void main() {
8.   runApp(NavButton());
9. }
10. /*
11. Classe NavButton:
12.
13. A classe NavButton é um StatelessWidget que define a tela
   principal, que é a Navigator_screen. O MaterialApp contém
   essa tela como home.
14. */
15. class NavButton extends StatelessWidget {
16.   const NavButton({super.key});
17.
18.   @override
19.   Widget build(BuildContext context) {
20.     return MaterialApp(
21.       home: Navigator_screen(),
22.     );
23.   }
24. }
25. /*
26. Classe Navigator_screen:
27. Navigator_screen é um StatefulWidget, já que o estado
   da tela muda de acordo com o item selecionado no
   BottomNavigationBar.
28. A variável selectedIndex controla qual widget será exibido
   na tela principal com base na escolha do usuário no menu
   de navegação inferior.
29.
```

```
30. */
31.
32. class Navigator_screen extends StatefulWidget {
33.   const Navigator_screen({super.key});
34.
35.   @override
36.   State<Navigator_screen> createState() =>
37.     _NavigatorScreenState();
38. }
39.
40.
41.
42.
43. class _NavigatorScreenState extends State<Navigator_
44.   screen> {
45.   int selectedIndex =0; // variavel pela escolha do widget
46.   bar
47.   // Constante para definir caracteristicas do texto
48.   static const TextStyle optionStyle =
49.     TextStyle(fontSize: 30, fontWeight: FontWeight.bold);
50.
51.   // Cria lista de widgets
52.   static const List<Widget> _widgetOptions = <Widget>[
53.
54.     /*Text(
55.       'Index 0: School',
56.       style: optionStyle,
57.     ), */
```

```
58.    // Tela1() ,
59.    TelaHome(),
60.    /*Text(
61.        'Index 1: Business',
62.        style: optionStyle,
63.    ),*/
64.    CheckboxExample(),
65.    RadioButton(),
66.    /*Text(
67.        'Index 2: School',
68.        style: optionStyle,
69.    ),
70.    */
71.    Text(
72.        'Index 3: Settings',
73.        style: optionStyle,
74.    ),
75.
76.
77.    ];
78. // O método onItemTapped() altera o valor de selectedIndex
    para refletir o item //selecionado.
79.
80. void onItemTapped(int index) {
81.     setState(() {
82.         selectedIndex = index;
83.
84.
85.     });
86. }
87. @override
88. Widget build(BuildContext context) {
89.     return Scaffold(
90.         appBar: AppBar(
```

```
91.           title: Text("Aplicativo aula 04"),
92.           ),
93.           body: Center(
94.             child: _widgetOptions.elementAt(selectIndex),
95.           ),
96.
97. /*BottomNavigationBar:
98. Contém quatro itens: "Home", "Business", "School"
99. e "Settings", com ícones e cores definidas. O item
100. selecionado é destacado, e o conteúdo da tela muda de
101. acordo com a escolha do usuário.
102. */
103. bottomNavigationBar: BottomNavigationBar(
104.   items: const <BottomNavigationBarItem>[
105.     BottomNavigationBarItem(
106.       icon: Icon(Icons.home),
107.       label: 'Home',
108.       backgroundColor: Colors.red,
109.     ),
110.     BottomNavigationBarItem(
111.       icon: Icon(Icons.business),
112.       label: 'Business',
113.       backgroundColor: Colors.blue,
114.     ),
115.     BottomNavigationBarItem(
116.       icon: Icon(Icons.school),
117.       label: 'School',
118.       backgroundColor: Colors.orange,
119.     ),
120.     BottomNavigationBarItem(
121.       icon: Icon(Icons.settings),
122.       label: 'Settings',
123.       backgroundColor: Colors.purple,
124.     ),
125.   ],
126.   selectedItemIndex: 0,
127.   onTap: (int index) {
128.     setState(() {
129.       selectIndex = index;
130.     });
131.   },
132. 
```

```
122.           ],
123.           currentIndex: selectedIndex,
124.           selectedItemColor: Colors.white,
125.           onTap: onItemClick,
126.         ),
127.
128.
129.
130.
131.
132.       );
133.     }
134.   }
135.
136. /*
137. Telas Individuais:
138. Tela Home (TelaHome): Uma tela simples com um texto
    centralizado, indicando “Mobile 2”.
139.
140. */
141. class TelaHome extends StatelessWidget {
142.   const TelaHome({super.key});
143.   static const TextStyle styletext = TextStyle(fontSize:
    30, fontWeight: FontWeight.bold);
144.   @override
145.   Widget build(BuildContext context) {
146.     return Scaffold(
147.
148.       body: Column(
149.         mainAxisAlignment: MainAxisAlignment.center,
150.         children: [
151.
152.           Center(child: Text("Mobile 2",style: styletext)),
153.         ],

```

```
154.         ),
155.     );
156.   }
157. }
158.
159. /*
160. Checkbox (CheckboxExample): Exibe um Checkbox que altera
161. o estado de um valor booleano (ischeck) que é exibido na
162. tela.
163. Radio Button (RadioButton): Exibe três opções de
164. RadioListTile, e o valor selecionado é armazenado na
165. variável selectOption.
166.
167. */
168. class CheckboxExample extends StatefulWidget {
169.   const CheckboxExample({super.key});
170.
171.   @override
172.   State<CheckboxExample> createState() => CheckBoxState();
173.
174.   bool ischeck = false;
175.   @override
176.   Widget build(BuildContext context) {
177.     return Scaffold(
178.       body: Column(
179.         children: [
180.           Text("Check Box 1"),
181.           Checkbox(
182.             // Value é o parametro do check box
```

```
183.           // ischeck variavel booleana que recebe o
    parametro value do check box
184.           value:  ischeck,
185.           onChanged:(bool? value){
186.               setState(() {
187.                   ischeck = value!;
188.
189.               });
190.           }
191.
192.
193.
194.           ),
195.           Text(«Status do checkbox $ischeck»),
196.           ],
197.           ),
198.           );
199.       }
200.   }
201.
202. class RadioButton extends StatefulWidget {
203.     const RadioButton({super.key});
204.
205.     @override
206.     State<RadioButton> createState() => _RadioButtonState();
207. }
208.
209. class _RadioButtonState extends State<RadioButton> {
210.     int ? selectOption;
211.     @override
212.     Widget build(BuildContext context) {
213.         return Column(
214.             mainAxisAlignment: MainAxisAlignment.center,
```

```
215.         children: [
216.             RadioListTile<int>(
217.                 title: Text(«Opção 1»),
218.                 value: 1,
219.                 groupValue: selectOption,
220.                 onChanged: (value){
221.                     setState(() {
222.                         selectOption = value;
223.
224.                     });
225.                 },
226.
227.             ),
228.             RadioListTile<int>(
229.                 title: Text(«Opção 2»),
230.                 value: 2,
231.                 groupValue: selectOption,
232.                 onChanged: (value){
233.                     setState(() {
234.                         selectOption = value;
235.
236.                     });
237.                 },
238.
239.             ),
240.             RadioListTile(
241.                 title: Text(«Opção 3»),
242.                 value: 3,
243.                 groupValue: selectOption,
244.                 onChanged: (value){
245.                     setState(() {
246.                         selectOption = value;
247.
```

```
248.           });
249.         },
250.
251.       ),
252.       Text(«Radio button opção $selectOption»),
253.
254.     ],
255.   );
256. }
257. }
258.
```

## Atividades

**1. Crie um aplicativo simples com navegação por meio de uma barra inferior (Navigator Bar). Cada tela deve exibir conteúdo relacionado ao tema (um texto simples ou imagem, por exemplo).**

Exemplo de saída:

- indústria – texto explicando o que é uma indústria;
- agricultura – imagem de plantações;
- saúde – texto sobre saúde pública;
- economia – texto sobre PIB.

**2. Desenvolva um aplicativo com um Navigator Bar que contenha 4 opções representando diferentes países (por exemplo: Brasil, Itália, Japão, México).**

**Em cada tela, exiba informações sobre um prato típico de cada um dos países escolhidos, incluindo o nome da comida e uma imagem que a represente.**

Exemplo de saída:

- Brasil – feijoada (com imagem);
- Itália – pizza (com imagem);
- Japão – sushi (com imagem);
- México – tacos (com imagem).

**3. Crie um aplicativo para coletar informações de um usuário.**

Perguntas obrigatórias:

Sexo: use Radio Buttons para as opções Masculino e Feminino.

Estado civil: use Radio Buttons para as opções Solteiro, Casado, Divorciado e Viúvo.

Após o preenchimento, exiba os dados coletados em uma tela separada (usando o Navigator).

Exemplo de fluxo:

- o usuário seleciona as opções desejadas e pressiona o botão “Concluir”;
- na próxima tela, os dados escolhidos são exibidos.

**4. Adicione uma tela de login com entrada de nome de usuário e senha. Valide que o nome e a senha não estão vazios antes de permitir o acesso.**

Após o login bem-sucedido, exiba um Navigator Bar com as opções:

- área do triângulo – permita calcular a área a partir da base e da altura;
- área do trapézio – permita calcular a área usando as bases maior, menor e a altura;
- 1<sup>a</sup> Lei de Ohm – permita calcular tensão (V), corrente (I) ou resistência (R), dependendo do que o usuário escolher.

Cada cálculo deve ser realizado em uma tela separada.

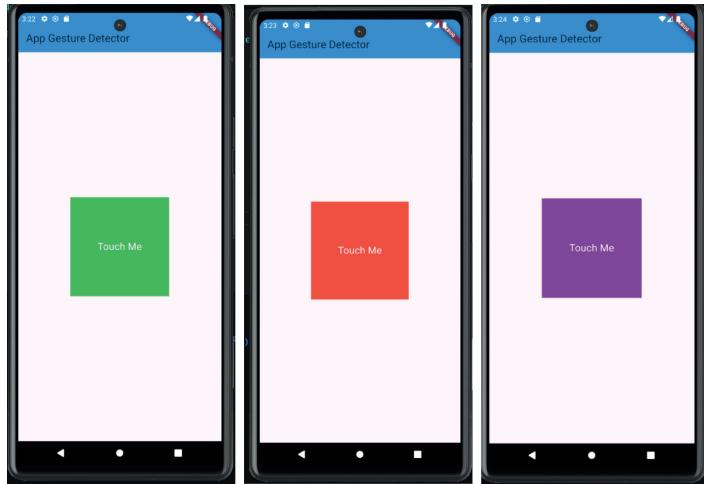
Exemplo de fluxo: o usuário faz o login, escolhe um cálculo na barra de navegação, insere os valores e vê o resultado.

## Controle dos elementos na tela

O controle dos elementos é feito com propriedades e métodos de widgets, como `setState()`, para atualizar elementos de um `StatefulWidget`.

**GestureDetector:** detecta gestos de toque e permite interações customizadas.

Na figura a seguir, é possível visualizar o aplicativo com o Gesture Detector. Neste aplicativo, o container fica verde com um toque, com dois toques fica vermelho e, com um toque longo, fica roxo.



**Figura 2.33 – Aplicativo com Gesture Detector.**

A seguir, é possível visualizar o código com o Gesture Detector.

#### Código exemplo Gesture Detector

```
1. // import: Importa o pacote Flutter necessário para criar  
a interface do usuário.  
2. import 'package:flutter/material.dart';  
3.  
4. // main: Ponto de entrada do aplicativo Flutter, onde o  
widget raiz (MyApp) é chamado.  
5.  
6. void main() {  
7.   runApp(MyApp());  
8. }  
9. /*MyApp:  
10. É um StatelessWidget, ou seja, não possui estado interno.  
11. Retorna um MaterialApp com a página inicial definida como  
GestureDetectorExample.
```

```
12. */
13. class MyApp extends StatelessWidget {
14.   @override
15.   Widget build(BuildContext context) {
16.     return MaterialApp(
17.       home: GestureDetectorExample(),
18.     );
19.   }
20. }
21.
22. /*
23. GestureDetectorExample:
24. Um StatefulWidget, pois o estado (a cor do Container)
25. pode mudar durante a execução.
26. Cria o estado associado à classe através de
27. _GestureDetectorExampleState.
28.
29. */
30. class GestureDetectorExample extends StatefulWidget {
31.   @override
32.   _GestureDetectorExampleState createState() =>
33.   _GestureDetectorExampleState();
34. }
35.
36.
37. class _GestureDetectorExampleState extends
38. State<GestureDetectorExample> {
39.   Color _color = Colors.blue;
40.   /*
41.   Componentes:
42.   _color:
43.
44.   Armazena a cor atual do Container.
45.   Inicialmente é definida como Colors.blue.
46.   _changeColor:
```

```
43.  
44. Método para alterar a cor do Container.  
45. Chama setState para notificar o Flutter que o estado  
    mudou, atualizando a interface com a nova cor.  
46. */  
47.  
48. void _changeColor(Color color) {  
49.     setState(() {  
50.         _color = color;  
51.     });  
52. }  
53.  
54. @override  
55. Widget build(BuildContext context) {  
56. /*  
57.  
58. Componentes:  
59. Scaffold:  
60.  
61. Estrutura básica para a tela, contendo uma AppBar e o  
    corpo (body).  
62. AppBar:  
63.  
64. Exibe um título fixo: “App Gesture Detector”.  
65. Tem um fundo azul.  
66. GestureDetector:  
67.  
68. Envolve o Container e detecta gestos do usuário. Os  
    gestos configurados incluem:  
69. onTap: Detecta um toque único e muda a cor para verde.  
70.  
71. */  
72.     return Scaffold(  
73.         appBar: AppBar(  
74.             title: Text('App Gesture Detector'),
```

```
75.           backgroundColor: Colors.blue,
76.         ),
77.       body: Center(
78.         child: GestureDetector(
79.           onTap: () {
80.
81. /*Container:
82.
83. Um retângulo com largura e altura de 200 pixels.
84. A cor é controlada pela variável _color.
85. Exibe o texto “Touch Me” centralizado.
86.
87. */
88.           _changeColor(Colors.green);
89.         },
90. // onDoubleTap: Detecta dois toques rápidos e muda a cor
91. // para vermelho.
91.
92.           onDoubleTap: () {
93.             _changeColor(Colors.red);
94.           },
95. // onLongPress: Detecta um toque longo e muda a cor para
96. // roxo.
96.
97.           onLongPress: () {
98.             _changeColor(Colors.purple);
99.           },
100.          child: Container(
101.            width: 200,
102.            height: 200,
103.            color: _color,
104.            alignment: Alignment.center,
105.            child: Text(
106.              ‘Touch Me’,
107.              style: TextStyle(
```

```
108.           color: Colors.white,  
109.           fontSize: 20,  
110.           ),  
111.           ),  
112.           ),  
113.           ),  
114.           ),  
115.       );  
116.   }  
117. }  
118.
```

## Tratamento de eventos e exceções

No Flutter, o tratamento de eventos e exceções desempenha um papel crucial no controle do fluxo do aplicativo e no manejo de possíveis erros que possam ocorrer durante a execução.

- Eventos são manipulados com onTap, onPressed e outros controladores de evento.
- O tratamento de exceções usa try-catch para capturar erros em operações assíncronas.

Os eventos são interações do usuário ou mudanças no estado do aplicativo que precisam ser capturadas e tratadas para gerar uma resposta apropriada. Para isso, o Flutter fornece métodos e propriedades específicos, como:

- **onTap** – utilizado para capturar toques em widgets, como o GestureDetector;
- **onPressed** – comumente usado em botões, como ElevatedButton e IconButton;

- **onChanged** – utilizado para capturar mudanças em entradas de texto ou seleções.

A seguir, é possível visualizar o tratamento de eventos on pressed.

Código para tratamento de evento.

```

1. /*
2. onPressed:
3.
4. É uma função callback que define o que acontece quando o
botão é pressionado.
5. Neste exemplo:
6. A função imprime no console a mensagem “Botão
pressionado!”.
7. child:
8.
9. Define o conteúdo exibido dentro do botão.
10. Aqui, o child é um widget Text com o texto “Clique aqui”.
11. */
12. ElevatedButton(
13.   onPressed: () {
14.     print("Botão pressionado!");
15.   },
16.   child: Text("Clique aqui"),
17. );
18.

```

## Tratamento de exceções

O tratamento de exceções é essencial para lidar com erros de forma segura e evitar que o aplicativo trave inesperadamente. O Flutter utiliza o bloco try-catch para capturar e tratar exceções, especialmente em operações assíncronas.

A seguir, é possível visualizar o código para tratamento de exceções com try catch.

## Código Try catch

```

1. /*
2. try
3. O bloco try contém o código que pode gerar uma exceção.
4. Aqui, a operação 10 ~/ 0 tenta realizar uma divisão
   inteira (~/) por zero, o que lança uma exceção de Divisão
   por Zero.
5. catch
6. O bloco catch é executado quando ocorre uma exceção no
   bloco try.
7. O parâmetro e captura a exceção gerada, que é exibida no
   console usando
8. */
9. try {
10.     int resultado = 10 ~/ 0; // Gera uma exceção (divisão
    por zero)
11. } catch (e) {
12.     print("Erro: $e");
13. }
14.
```

A seguir, é possível visualizar o código para tratamento de dados vindos de uma API.

## Código para tratamento de eventos assíncronos

```

1. /*
2. Future<void>:
3. A função retorna um Future que, ao ser concluído, não
   retorna nenhum valor (void).
4. Um Future representa uma operação assíncrona que pode ser
   concluída com sucesso ou erro.
5. async:
6. Marca a função como assíncrona, permitindo o uso de await
   dentro dela.
```

```
7. */
8. Future<void> carregarDados() async {
9.   try {
10.     var dados = await obterDadosDaAPI();
11.     print("Dados carregados: $dados");
12.   } catch (e) {
13.     print("Erro ao carregar dados: $e");
14.   }
15. }
16.
```

O uso combinado de eventos e exceções garante que o aplicativo seja interativo e resiliente, oferecendo uma experiência mais robusta para o usuário.

## Manipulação de listas na interface

- ListView e GridView são amplamente usados para exibir listas de dados.
- Os widgets ListTile e GridTile facilitam o design de itens de lista.

A seguir, é possível visualizar o código para o ListView.

### Código ListView

```
1. /*
2. ListViewScreen:
3.
4. Um StatelessWidget, pois não mantém estado interno.
5. Usa Scaffold para a estrutura básica da tela.
6. AppBar:
7.
8. Exibe o título “Tela com ListView”.
9. ListView.builder:
10.
```

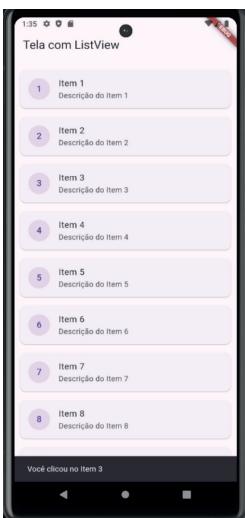
```
11. Cria dinamicamente uma lista de 20 itens, definidos por
    itemCount: 20.
12. O itemBuilder é chamado para renderizar cada item.
13.
14. */
15. // Tela com ListView
16. class ListViewScreen extends StatelessWidget {
17.   @override
18.   Widget build(BuildContext context) {
19.     return Scaffold(
20.       appBar: AppBar(
21.         title: Text('Tela com ListView'),
22.       ),
23.       body: ListView.builder(
24.         padding: const EdgeInsets.all(10),
25.         itemCount: 20, // Número de itens no ListView
26.         itemBuilder: (context, index) {
27.           return Card(
28.             margin: const EdgeInsets.symmetric(vertical: 8),
29.             child: ListTile(
30.               leading: CircleAvatar(
31.                 child: Text('${index + 1}'),
32.               ),
33.               title: Text('Item ${index + 1}'),
34.               subtitle: Text('Descrição do Item ${index +
35.                 1}'),
36.               onTap: () {
37.                 // Exibe uma mensagem ao clicar no item
38.                 ScaffoldMessenger.of(context).showSnackBar(
39.                   SnackBar(content: Text('Você clicou no
40.                     Item ${index + 1}')),
41.                 );
42.               },
43.             );
44.           );
45.         },
46.       );
47.     );
48.   }
49. }
```

```

43.         },
44.         ),
45.     );
46. }
47. }
48.
49.

```

A figura a seguir mostra o aplicativo com o ListView.



**Figura 2.34 – Aplicativo com ListView.**

A seguir, é possível visualizar o código do GridView.

#### Código GridView

1. `/*`
2. `A tela é composta por:`
- 3.
4. `Um AppBar exibindo o título “Tela com GridView”.`
5. `Um GridView.builder no corpo, que exibe os itens em um layout de grade (grid).`
- 6.
7. `Scaffold:`
- 8.

```
9. Fornece a estrutura básica da tela.  
10. Contém:  
11. AppBar: Exibe o título da tela.  
12. GridView.builder: Um layout em grade que constrói itens  
    dinamicamente.  
13. GridView.builder:  
14.  
15. Cria uma grade dinâmica e eficiente para renderizar itens.  
16. padding: Espaçamento ao redor da grade (10 pixels).  
17. gridDelegate:  
18. Define como os itens são organizados no grid.  
19. SliverGridDelegateWithFixedCrossAxisCount:  
20. crossAxisCount: Número de colunas (neste caso, 2).  
21. crossAxisSpacing: Espaçamento horizontal entre as colunas  
    (10 pixels).  
22. mainAxisSpacing: Espaçamento vertical entre as linhas (10  
    pixels).  
23. itemCount: Número total de itens (neste caso, 10).  
24. itemBuilder:  
25. Função que constrói cada item da grade com base no  
    índice.  
26.  
27.  
28. GestureDetector:  
29.  
30. Detecta interações do usuário (toque, gesto) no item.  
31. onTap:  
32. Quando o item é tocado, o usuário é redirecionado para a  
    tela ListViewScreen usando:  
33.  
34. */  
35. class GridViewScreen extends StatelessWidget {  
36.   @override  
37.   Widget build(BuildContext context) {  
38.     return Scaffold(  
39.       appBar: AppBar(
```

```
40.         title: Text('Tela com GridView'),
41.     ),
42.     body: GridView.builder(
43.         padding: const EdgeInsets.all(10),
44.         gridDelegate:
45.             SliverGridDelegateWithFixedCrossAxisCount(
46.                 crossAxisCount: 2, // Número de colunas
47.                 crossAxisSpacing: 10, // Espaçamento horizontal
48.                 mainAxisSpacing: 10, // Espaçamento vertical
49.             ),
50.             itemCount: 10, // Número de itens no GridView
51.             itemBuilder: (context, index) {
52.                 return GestureDetector(
53.                     onTap: () {
54.                         // Navega para a tela de ListView
55.                         Navigator.push(
56.                             context,
57.                             MaterialPageRoute(builder: (context) =>
58.                                 ListViewScreen(),
59.                             );
60.                         },
61.                         child: Container(
62.                             decoration: BoxDecoration(
63.                                 color: Colors.blueAccent,
64.                                 borderRadius: BorderRadius.circular(12),
65.                             ),
66.                             alignment: Alignment.center,
67.                             child: Text(
68.                                 'Item ${index + 1}',
69.                                 style: TextStyle(
70.                                     color: Colors.white,
71.                                     fontSize: 16,
72.                                 ),
73.                             );
74.             );
```

```
74.          },
75.          ),
76.          );
77.      }
78.  }
79.
```

A seguir, é possível visualizar o código completo para o GridView e o ListView.

#### Código completo do aplicativo com GridView e ListView

```
1. import 'package:flutter/material.dart';
2.
3. void main() {
4.   runApp(MyApp());
5. }
6.
7. class MyApp extends StatelessWidget {
8.   @override
9.   Widget build(BuildContext context) {
10.     return MaterialApp(
11.       title: 'GridView e ListView',
12.       theme: ThemeData(
13.         primarySwatch: Colors.blue,
14.       ),
15.       home: GridViewScreen(), // Tela inicial com GridView
16.     );
17.   }
18. }
19.
20. // Tela com GridView
21. class GridViewScreen extends StatelessWidget {
22.   @override
23.   Widget build(BuildContext context) {
24.     return Scaffold(
```

```
25.      appBar: AppBar(
26.        title: Text('Tela com GridView'),
27.      ),
28.      body: GridView.builder(
29.        padding: const EdgeInsets.all(10),
30.        gridDelegate:
31.          SliverGridDelegateWithFixedCrossAxisCount(
32.            crossAxisCount: 2, // Número de colunas
33.            crossAxisSpacing: 10, // Espaçamento horizontal
34.            mainAxisSpacing: 10, // Espaçamento vertical
35.          ),
36.          itemCount: 10, // Número de itens no GridView
37.          itemBuilder: (context, index) {
38.            return GestureDetector(
39.              onTap: () {
40.                // Navega para a tela de ListView
41.                Navigator.push(
42.                  context,
43.                  MaterialPageRoute(builder: (context) =>
44.                    ListViewScreen(),
45.                  );
46.                },
47.                child: Container(
48.                  decoration: BoxDecoration(
49.                    color: Colors.blueAccent,
50.                    borderRadius: BorderRadius.circular(12),
51.                  ),
52.                  alignment: Alignment.center,
53.                  child: Text(
54.                    'Item ${index + 1}',
55.                    style: TextStyle(
56.                      color: Colors.white,
57.                      fontSize: 16,
58.                    ),
59.                  ),
60.                ),
61.              ),
62.            );
63.          },
64.        ),
65.      ),
66.    );
67.  ),
68.);
```

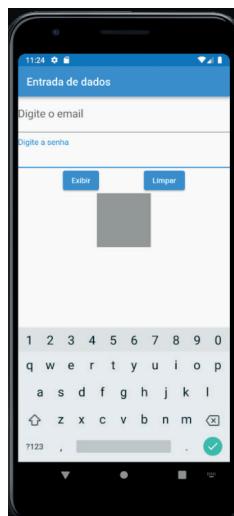
```
59.           );
60.           },
61.           ),
62.           );
63.     }
64.   }
65.
66. // Tela com ListView
67. class ListViewScreen extends StatelessWidget {
68.   @override
69.   Widget build(BuildContext context) {
70.     return Scaffold(
71.       appBar: AppBar(
72.         title: Text('Tela com ListView'),
73.       ),
74.       body: ListView.builder(
75.         padding: const EdgeInsets.all(10),
76.         itemCount: 20, // Número de itens no ListView
77.         itemBuilder: (context, index) {
78.           return Card(
79.             margin: const EdgeInsets.symmetric(vertical: 8),
80.             child: ListTile(
81.               leading: CircleAvatar(
82.                 child: Text('${index + 1}'),
83.               ),
84.               title: Text('Item ${index + 1}'),
85.               subtitle: Text('Descrição do Item ${index +
1}'),
86.               onTap: () {
87.                 // Exibe uma mensagem ao clicar no item
88.                 ScaffoldMessenger.of(context).showSnackBar(
89.                   SnackBar(content: Text('Você clicou no
Item ${index + 1}')),
90.                 );
91.               },
92.             ),
93.           );
94.         }
95.       )
96.     );
97.   }
98. }
```

```
93.           );
94.           },
95.           ),
96.           );
97.       }
98.   }
99.
```

## Entrada, processamento e saída de dados

- **TextField** para entrada de texto, com controller para controle e manipulação.
- **Form** ajuda na validação e no agrupamento de campos de entrada.

Na figura a seguir, é possível visualizar o aplicativo com dois campos “texto” para o usuário inserir informações, e, quando ele clicar no botão, as informações no terminal do VSCode serão exibidas.



**Figura 2.35** – Emulador Android executando aplicativo com campo texto.

A seguir, é possível visualizar o código do aplicativo com o campo texto para receber informações digitadas pelo usuário.

## Código do aplicativo campo texto

```
1. import 'package:flutter/material.dart';
2. /* Função main():
3. A função main() chama o método runApp() para inicializar
o aplicativo. O widget principal (Telaprincipal) é passado
como parâmetro para o runApp().
4. */
5. void main()
6. {
7.   runApp(Telaprincipal());
8. }
9. /*
10. Classe Telaprincipal:
11. Esta classe é o ponto de entrada do aplicativo e é
um StatelessWidget, o que significa que ela não possui
estado. Dentro do método build(), o widget MaterialApp é
retornado, com a tela principal sendo o Campotexto.
12. */
13. class Telaprincipal extends StatelessWidget {
14.   const Telaprincipal({super.key});
15.
16.   @override
17.   Widget build(BuildContext context) {
18.     return MaterialApp(
19.       home: Campotexto() ,
20.     );
21.   }
22. }
23.
24. class Campotexto extends StatefulWidget {
25.   Campotexto({super.key});
26.   @override
27.   State<Campotexto> createState() => CampotextoState();
28. }
```

```
29.  
30. class CampotextoState extends State<Campotexto> {  
31.   // variavel do tipo TextEditing controller para armazenar o  
32.   // que é digitado no campo texto  
33.   TextEditingController ctexto = TextEditingController();  
34.   @override  
35.   Widget build(BuildContext context) {  
36.     appBar: AppBar(  
37.       title: Text("Aplicativo Caixa de texto"),  
38.     ),  
39.     body: Column(  
40.       children: [  
41.         // Campo texto  
42.       /*  
43.      Classe Campotexto:  
44.      Esta classe é um StatefulWidget, pois ela possui um  
        estado mutável (no caso, o valor do texto inserido no  
        campo de texto). A criação do estado é feita pela classe  
        CampotextoState.  
45.      Dentro da classe CampotextoState, temos o controlador  
        de texto (TextEditingController), que é utilizado para  
        manipular o conteúdo do campo de texto.  
46.      */  
47.      /* Uso do TextField:  
48.      O TextField é utilizado para capturar o texto digitado  
        pelo usuário. No seu código, temos dois campos de texto:  
49.      O primeiro campo é utilizado para capturar o nome do  
        usuário e é controlado pela variável ctexto do tipo  
        TextEditingController.  
50.      O segundo campo é um campo para capturar um número  
        (porém, não há controle para ele no momento).  
51.      Botão “Verificar”:  
52.      O botão chama a função print(ctexto.text), que exibe o  
        texto digitado no primeiro campo no terminal.
```

```
53. */
54.         TextField(
55.             keyboardType: TextInputType.name,
56.             decoration: InputDecoration(labelText: "Digite seu
57.                 nome"),
58.             /* onChanged: (String texto){
59.                 print(texto);
60.             },
61.             /*onSubmitted: (String texto){
62.                 print(texto);
63.             },
64.             */
65.             controller: ctexto,
66.         ),
67.             ElevatedButton(onPressed: (){
68.                 print(ctexto.text);
69.             }, child: Text("Verificar")),
70.             TextField(
71.                 keyboardType: TextInputType.url,
72.                 decoration: InputDecoration(labelText: "Digite um
73.                     numero"),
74.                 ),
75.             ],
76.             );
77.         }
78.     }
79.
80.
81.
82.
83.
```

# Navegação entre telas e passagem de parâmetros

O **Navigator** é uma ferramenta fundamental no Flutter, pois gerencia a pilha de navegação das telas ou páginas. Ele permite navegar entre diferentes telas empilhando ou desempilhando páginas na pilha, ou seja, funciona como uma pilha de páginas, conforme indicado a seguir:

- **push** – adiciona (empilha) uma nova página no topo da pilha, exibindo-a.
- **pop** – remove (desempilha) a página do topo da pilha, voltando à página anterior.

A seguir, é possível visualizar o código exemplo para o Navigator.

## Código Navigator

```
1. /*
2. context: O contexto atual do widget.
3. MaterialPageRoute: Define a transição para a nova tela.
4. builder: Especifica a tela de destino (aqui, SegundaTela).
5. */
6. Navigator.push(
7.   context,
8.   MaterialPageRoute(builder: (context) => SegundaTela()),
9. );
10.
11. Navigator.pop(context);
12.
```

As **Named Routes** facilitam a organização de rotas e passagem de parâmetros e são uma abordagem mais organizada para gerenciar a navegação em aplicativos com várias telas. Em vez de definir rotas diretamente no

código com MaterialPageRoute, é possível registrar nomes para as telas em um mapa de rotas, o que simplifica a navegação e a manutenção do código.

Para configurar as Named Routes, defina as rotas nomeadas. Configure as rotas no MaterialApp usando a propriedade routes.

Para navegar usando o nome da rota, use Navigator.pushNamed.

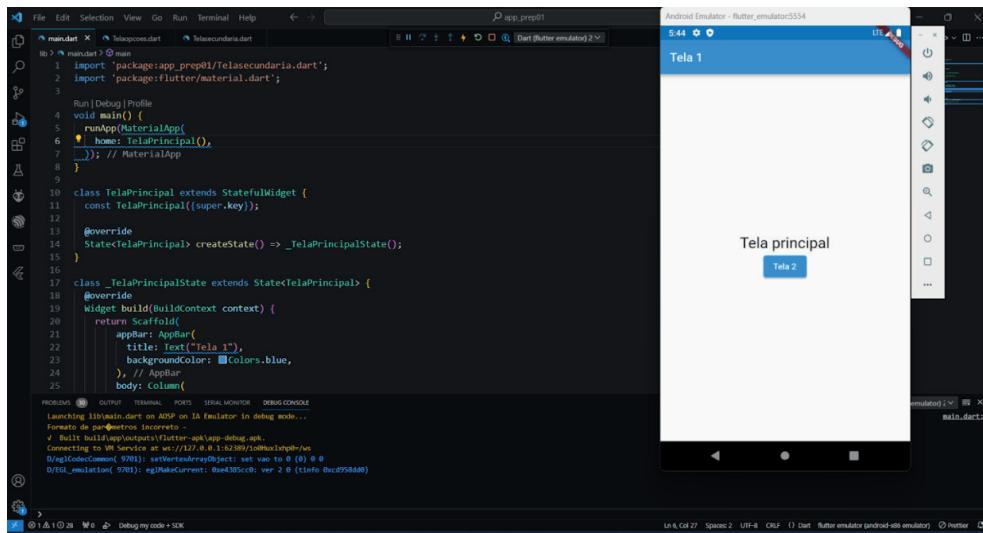
A seguir, é possível visualizar o código para navegar entre as telas utilizando rotas nomeadas.

### Código com rotas nomeadas

```
1. /* runApp:  
2. Inicializa o aplicativo Flutter e configura a árvore de  
   widgets.  
3. MaterialApp:  
4. Define as configurações globais do aplicativo, como tema,  
   rotas e a rota inicial.  
5. initialRoute:  
6. Especifica a rota que será exibida quando o aplicativo  
   iniciar (neste caso, '/', que aponta para PrimeiraTela).  
7. routes:  
8. Um mapa de rotas nomeadas para suas respectivas telas.  
9. '/': Mapeada para PrimeiraTela.  
10. '/segunda': Mapeada para SegundaTela.  
11. */  
12. void main() {  
13.   runApp(MaterialApp(  
14.     initialRoute: '/', // Define a rota inicial  
15.     routes: {  
16.       '/': (context) => PrimeiraTela(),  
17.       '/segunda': (context) => SegundaTela(),  
18.     },  
19.   ));  
20. }  
21.  
22. class PrimeiraTela extends StatelessWidget {
```

```
23. @override
24. Widget build(BuildContext context) {
25.   return Scaffold(
26.     appBar: AppBar(title: Text('Primeira Tela')),
27.     body: Center(
28.       child: ElevatedButton(
29.         child: Text('Ir para a Segunda Tela'),
30.         onPressed: () {
31.           Navigator.pushNamed(context, '/segunda');
32.         },
33.       ),
34.     ),
35.   );
36. }
37. }
38.
39. class SegundaTela extends StatelessWidget {
40.   @override
41.   Widget build(BuildContext context) {
42.     return Scaffold(
43.       appBar: AppBar(title: Text('Segunda Tela')),
44.       body: Center(
45.         child: ElevatedButton(
46.           child: Text('Voltar para a Primeira Tela'),
47.           onPressed: () {
48.             Navigator.pop(context);
49.           },
50.         ),
51.       ),
52.     );
53.   }
54. }
55.
56.
```

Na figura a seguir, é possível visualizar um aplicativo que tem múltiplas telas.



**Figura 2.36 – Emulador Android executando aplicativo com múltiplas telas.**

O código main do aplicativo é o código principal e é executado quando o aplicativo é iniciado. O código apresentado é um exemplo básico de um aplicativo Flutter com múltiplas telas. Ele utiliza o Navigator para navegar entre diferentes páginas e organiza os widgets na tela com o uso de Scaffold, Column e ElevatedButton.

A seguir, é possível visualizar o código main.dart.

Código main.dart do aplicativo

1. */\**
2.     Função Principal **main()**
3.     **runApp:** Inicializa a aplicação Flutter.
4.     **MaterialApp:** Widget raiz para aplicações Android que fornece uma estrutura padrão (temas, navegação e outros recursos).
5.     A tela inicial do app é configurada como **Telaprincipal**
- 6.
- 7.

```
8. Classe Telaprincipal
9. A classe Telaprincipal é Stateless (sem estado), pois não
    possui variáveis mutáveis.
10. Ela constrói a interface principal com um Scaffold, que
    fornece a estrutura básica (app bar e body).
11. Widgets Utilizados:
12. AppBar:
13. A barra superior do aplicativo com um título:
14.
15. Column:
16. Organiza os widgets na vertical:
17.
18. Um Container com a mensagem “Tela 1”.
19. Um ElevatedButton para navegar até a tela 2.
20.
21. Navegação com o Navigator
22. A navegação entre telas é feita usando o Navigator.push.
23.
24. context: O contexto da tela atual.
25. MaterialPageRoute: Uma rota que cria a próxima tela do
    app.
26. */
27.
28. import 'package:app_aula03_telas_multiplas/tela2.dart';
29. import 'package:app_aula03_telas_multiplas/tela4.dart';
30. import 'package:flutter/material.dart'; // pacote de widgets
    para o android
31. // função principal do aplicativo
32. void main() {
33.     // runAPP função que constroi a tela do app
34.     // MaterialAPP função do widget para Android
35.     // home parâmetro inicial para chamar a tela
36.     runApp(MaterialApp(
37.         home: Telaprincipal()
38.
```

```
39.    ));
40. }
41.
42. class Telaprincipal extends StatelessWidget {
43.   const Telaprincipal({super.key});
44.
45.   @override
46.   // Constroi a tela do aplicativo
47.   Widget build(BuildContext context) {
48.     // Scaffold layout semipronto para criar o aplicativo
49.     return Scaffold(
50.       // app bar é a barra do aplicativo
51.       appBar: AppBar(
52.         title: Text("Aplicativo aula 03"),
53.       ),
54.       // body é o corpo do Scaffold
55.       body: // Column coluna do aplicativo
56.       Column(
57.         // Children estabelece relação com os demais
58.         widgets com a coluna
59.         mainAxisAlignment: MainAxisAlignment.center, //
60.         alinhamento principal do widget column
61.         children: [
62.           Container(color: Colors.blue,width: 400,height:
63.             180,
64.             child: Text("Tela 1",style: TextStyle(fontSize:25
65.               ),),),
66.             // child é a relação com 1 widgets
67.             // Navigator.push permite chamar a tela 2 e
68.             MaterialPageRoute chama a tela seguinte
69.             ElevatedButton(onPressed: (){
70.               Navigator.push(context,
71.                 MaterialPageRoute(builder: (context) => Telasecundaria(),));
72.             }, child: Text("Tela 2")),
```

```
68.           ],
69.
70.           ),
71.           );
72.       }
73.   }
74.
75.
76.
```

A seguir, é possível visualizar o código da Tela 2 do aplicativo.

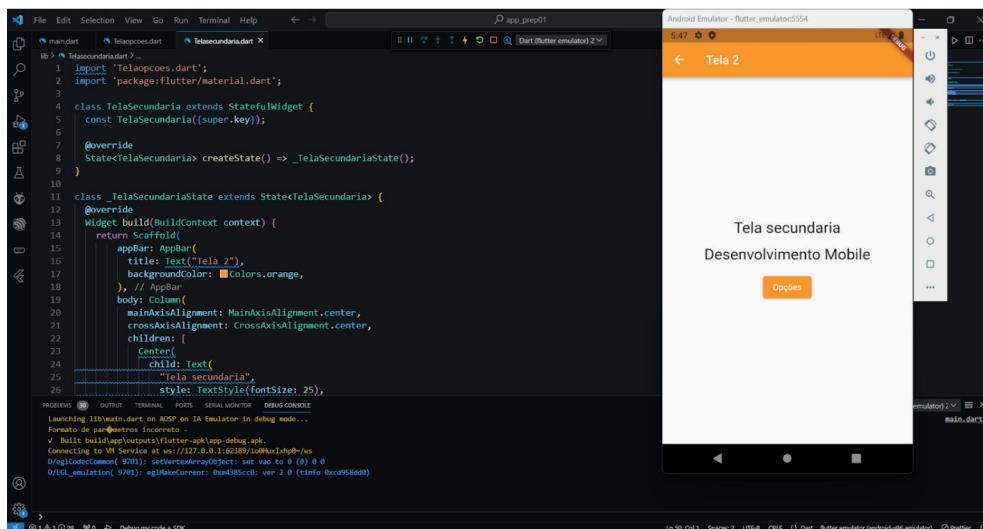
#### Código da Tela 2 do aplicativo

```
1. /*
2. 1. Estrutura do Scaffold
3. A tela utiliza o Scaffold como estrutura principal, que
   fornece um AppBar e um body.
4.
5. AppBar:
6. Mostra o título da tela (“Tela 2”).
7.
8. Column:
9. Organiza os elementos na vertical:
10.
11. Um Container vermelho.
12. Dois botões criados com ElevatedButton.
13.
14. 2. Botão “Tela 1”
15. O primeiro botão utiliza Navigator.pop para retornar à
   tela anterior.
16.
17. 3. Botão “Tela 3”
18. O segundo botão usa Navigator.push para navegar para a
   Tela 3.
19.
```

```
20. Navigator.push: Adiciona uma nova rota (Tela 3) à pilha de navegação.  
21. MaterialPageRoute: Define a rota da nova tela.  
22. Telaop(): A classe responsável por construir a Tela 3.  
23.  
24. */  
25. import 'package:app_aula03_telas_multiplas/tela3.dart';  
26. import 'package:flutter/material.dart';  
27.  
28. class Telasecundaria extends StatelessWidget {  
29.   const Telasecundaria({super.key});  
30.  
31.   @override  
32.   Widget build(BuildContext context) {  
33.     return Scaffold(  
34.       appBar: AppBar(  
35.         title: Text("Tela 2"),  
36.       ),  
37.       body: Column(  
38.         children: [  
39.           Container(color: Colors.red, width: 400, height:  
40.             180,),  
41.           ElevatedButton(onPressed: (){  
42.             Navigator.pop(context);  
43.           }, child: Text("Tela 1")),  
44.           ElevatedButton(onPressed: (){  
45.             // Navigator.push permite chamar a próxima tela e  
46.             // o Page Route  
47.             Navigator.push(context, MaterialPageRoute(builder:  
48.               (context)=>Telaop())));  
49.           }, child: Text("Tela 3"))  
50.         ],  
51.       ),  
52.     );  
53.   }  
54. }  
55. 
```

- 51. );
- 52. }
- 53. }
- 54.
- 55.

A Tela 2 do aplicativo é mostrada na figura a seguir.



**Figura 2.37** – Emulador Android executando aplicativo com a tela 2.

A seguir, é possível visualizar o código da Tela 3.

### Código da Tela 3 do aplicativo

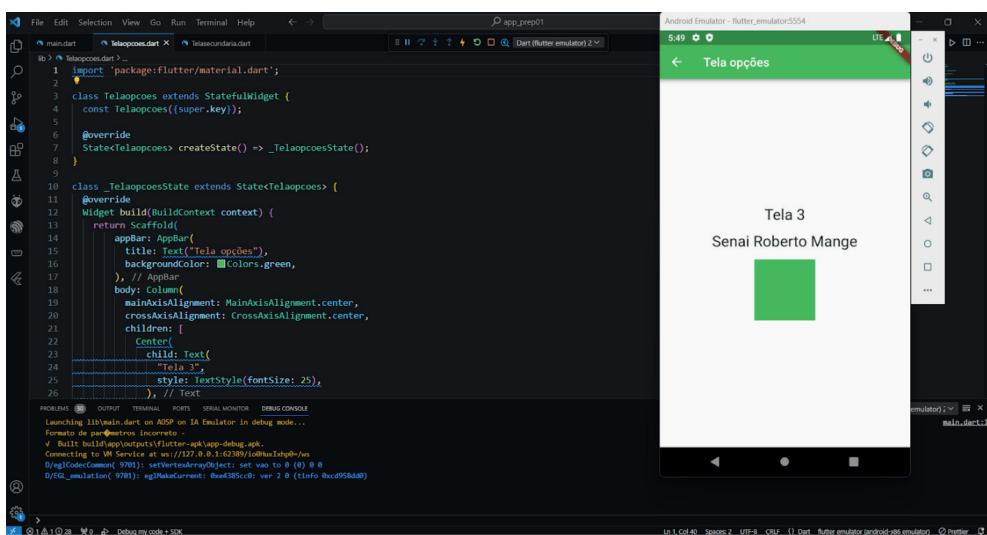
1. /\*
  2. 1. Estrutura do Scaffold
  3. A tela usa Scaffold para fornecer a estrutura padrão da interface com:
  - 4.
  5. AppBar: Título “Tela opções”.
  6. Body: Organizado com um Column contendo widgets verticais.
  - 7.
  8. 2. Botão “Tela 2”

```
9. O primeiro botão usa Navigator.pop para voltar à tela  
anterior (Tela 2):  
10.  
11. 3. Botão “Contador”  
12. O segundo botão utiliza Navigator.push para navegar até a  
Tela 4 (classe telacontador):  
13.  
14. Navigator.push: Adiciona a Tela 4 à pilha de navegação.  
15. telacontador(): A classe responsável por construir a Tela  
4.  
16.  
17. */  
18. import 'package:app_aula03_telas_multiplas/tela4.dart';  
19. import 'package:flutter/material.dart';  
20.  
21. class Telaop extends StatelessWidget {  
22.   const Telaop({super.key});  
23.  
24.   @override  
25.   Widget build(BuildContext context) {  
26.     return Scaffold(  
27.       appBar: AppBar(  
28.         title: Text("Tela opções"),  
29.       ),  
30.       body: Column(  
31.         children: [  
32.           Container(color: Colors.black, width: 400, height:  
180,),  
33.           ElevatedButton(onPressed: (){  
34.             Navigator.pop(context);  
35.  
36.           }, child: Text("Tela 2")),  
37.           ElevatedButton(onPressed: (){  
38.             Navigator.push(context,  
MaterialPageRoute(builder: (context)=>telacontador()));
```

```

39.
40.           }, child: Text("Contador"))
41.         ],
42.       ),
43.
44.     );
45.   }
46. }
47.
48.
49.
50.
51.

```



**Figura 2.38 – Emulador Android executando aplicativo com a Tela 3.**

## Gerenciamento de dependências

O arquivo pubspec.yaml é um dos elementos fundamentais em um projeto Flutter. Ele contém informações e configurações importantes.

A seguir, é possível visualizar o nome e a versão do projeto: define o nome do aplicativo e a versão atual.

#### Partes do arquivo pubspec.yaml

1. name: meu\_app
2. version: 1.0.0

Dependências do projeto: lista as bibliotecas e os pacotes externos utilizados no projeto. Eles podem ser adicionados manualmente ou via comandos do flutter pub.

A seguir, é possível visualizar as dependências utilizadas no projeto.

#### Dependências utilizadas no projeto

1. dependencies:
2. flutter:
3.     sdk: flutter
4.     provider: ^6.0.5
5.     dio: ^5.2.0
- 6.

A seguir, é possível visualizar os Assets e Recursos utilizados no projeto: define imagens, fontes e outros arquivos a serem incluídos no aplicativo.

#### Assets e Recursos utilizados no projeto

1. assets:
2.     - images/logo.png
- 3.

A seguir, é possível visualizar a configuração para pacotes: define pacotes personalizados ou locais.

#### Pacotes personalizados utilizados no aplicativo

1. dependencies:
2.     meu\_pacote\_local:

```
3.     path: ./pacotes/meu_pacote_local  
4.
```

Depois de editar o pubspec.yaml, é necessário executar o comando flutter pub get para atualizar as dependências. Os pacotes populares no Flutter podem ser acessados no repositório pub.dev. Alguns desses pacotes mais populares são: Firebase, Provider e http, entre outros.

A seguir, é possível visualizar um exemplo de arquivo pubspec.yaml.

#### Exemplo de arquivo pubspec.yaml

```
1. name: meu_app  
2. description: Um aplicativo Flutter de exemplo.  
3.  
4. environment:  
5.   sdk: <>=3.0.0 <4.0.0>  
6.  
7. dependencies:  
8.   flutter:  
9.     sdk: flutter  
10.  firebase_core: ^2.5.0  
11.  provider: ^6.0.5  
12.  dio: ^5.2.0  
13.  
14. dev_dependencies:  
15.   flutter_test:  
16.     sdk: flutter  
17.
```

No arquivo pubspec.yaml, são definidas as dependências do projeto. Entre os pacotes mais populares estão o Firebase, o Provider e o Dio (utilizado para requisições HTTP), que são amplamente utilizados na comunidade Flutter.

## Atividades

- 1. Você foi contratado pela empresa SM Mobile para desenvolver um aplicativo que solicite informações pessoais aos usuários em campos de texto (nome, idade, endereço, e-mail, telefone) utilizando o gerenciador de estado GetX para exibir as informações na segunda tela.**
- 2. Crie um aplicativo com uma lista de comidas e com o valor de cada uma. Deve ser exibido o valor total da compra utilizando o gerenciador de estado BloC para que essas informações apareçam na segunda tela.**
- 3. Crie um aplicativo com os nomes dos jogadores do seu esporte favorito e que mostre nome, idade e quantidade de pontos marcados, utilizando o gerenciador de estado BloC para exibir as informações na segunda tela.**
- 4. Crie um aplicativo que receba o nome do aluno e as notas de quatro disciplinas. Para cada disciplina, calcule a média de três avaliações e exiba se o aluno foi aprovado ou reprovado, juntamente com sua média. Utilize o gerenciador de estado GetX para apresentar essas informações em uma segunda tela.**

## Conclusão

Neste capítulo, exploramos os fundamentos essenciais para a criação de interfaces no Flutter, abordando desde os conceitos básicos até aspectos mais avançados. Compreendemos como os componentes visuais e a estrutura hierárquica dos widgets formam a base de uma aplicação móvel. Recursos como leiautes flexíveis, menus, diálogos e barras de ação foram apresentados como elementos fundamentais para garantir interatividade e organização.

Aprendemos a manipular listas, tratar eventos e exceções, além de implementar a entrada, o processamento e a saída de dados de forma eficiente. A navegação entre telas e a passagem de parâmetros também foram abordadas, demonstrando como criar experiências de usuário fluidas e intuitivas.

Além disso, discutimos a importância do tratamento de gestos e do gerenciamento de dependências, elementos que contribuem significativamente para a interatividade e a escalabilidade das aplicações.

Com esse conjunto de habilidades, você está preparado para desenvolver interfaces modernas e profissionais, aproveitando ao máximo o potencial do Flutter no desenvolvimento de aplicativos móveis. O conhecimento adquirido neste capítulo servirá como base sólida para a construção de aplicações mais complexas nos próximos tópicos, permitindo que você avance com segurança no universo do desenvolvimento mobile.



## CAPÍTULO 3

# APIs E CONSUMO DE RESTFUL WEB SERVICE

## Introdução

**N**o desenvolvimento de aplicativos móveis, a integração com serviços externos é uma prática fundamental para criar soluções conectadas, dinâmicas e interativas. Imagine, por exemplo, um aplicativo de delivery que precisa localizar restaurantes próximos, atualizar cardápios em tempo real e processar pagamentos. Tudo isso só é possível graças à integração com serviços externos por meio de APIs (Interfaces de Programação de Aplicações).

As APIs desempenham um papel essencial nesse processo, permitindo que os aplicativos se comuniquem com servidores e serviços para enviar, receber e manipular dados em tempo real. Existem diversas arquiteturas de

APIs, como SOAP, GraphQL e RESTful – sendo esta última a mais amplamente utilizada atualmente e o foco deste capítulo.

Neste capítulo, exploraremos como consumir APIs RESTful, um dos padrões mais adotados na indústria para comunicação entre sistemas. Você aprenderá a utilizar os principais métodos do protocolo HTTP – GET, POST, PUT e DELETE – para realizar operações com dados de forma eficiente e segura.

Também abordaremos os formatos de dados mais comuns, como JSON e XML, que são essenciais para interpretar e estruturar informações trocadas entre o aplicativo e os serviços externos.

Outro ponto importante será o uso de requisições assíncronas, fundamentais para garantir uma experiência fluida ao usuário. Imagine que o aplicativo precise buscar dados em um servidor e, durante esse processo, a interface fique congelada até que a resposta seja recebida. Esse tipo de comportamento pode gerar frustração. As requisições assíncronas resolvem esse problema ao permitir que a interface continue responsiva enquanto os dados são processados em segundo plano.

Ao final deste capítulo, você estará apto a desenvolver aplicativos móveis capazes de se conectar de forma eficiente a APIs RESTful, ampliando significativamente o potencial das suas aplicações ao integrá-las com serviços externos.

## Conceitos

Para compreender como aplicativos móveis interagem com serviços externos, é fundamental entender o conceito de requisições HTTP. Nesse processo, um dispositivo, como um navegador ou aplicativo móvel (o cliente), envia uma solicitação a outro dispositivo (o servidor). O servidor, por sua vez, processa a solicitação e retorna uma resposta ao cliente.

Essa troca de informações entre cliente e servidor é baseada no protocolo HTTP. Vamos explorar os componentes envolvidos nessa comunicação para aprofundar nosso entendimento.

## O que é uma API?

A sigla **API** significa *Application Programming Interface*, ou seja, uma interface que permite a comunicação entre dois sistemas ou programas diferentes. Pense em uma API como uma “ponte” que conecta seu aplicativo a um serviço externo, como informações meteorológicas, mapas ou processamento de pagamentos. Para facilitar o entendimento, imagine que você está em um restaurante. A API é como o garçom: ele recebe o pedido (requisição), leva-o à cozinha (servidor) para ser preparado e depois retorna com o prato solicitado (resposta). Assim, a API desempenha o papel de intermediária, garantindo que a comunicação entre seu aplicativo e o serviço externo ocorra de forma eficiente e organizada.

Por exemplo, ao criar um aplicativo de previsão do tempo, em vez de coletar você mesmo os dados sobre o clima, basta configurá-lo para fazer uma requisição à API de um serviço meteorológico, que retornará as informações solicitadas.

## O que é um Web Service?

Um Web Service é um tipo específico de API projetado para funcionar exclusivamente através da rede. Ele usa protocolos como HTTP e formatos como SOAP, REST ou XML-RPC para transmitir dados. Em resumo, todo Web Service é uma API, mas nem toda API é um Web Service.

### CURIOSIDADE

#### Qual é a diferença entre API e Web Service?

A dúvida sobre a diferença entre API e Web Service é comum. Vamos simplificar.

**API:** é mais ampla. Pode ou não usar a internet para comunicação. Por exemplo, um aplicativo local no seu computador pode usar APIs internas para executar tarefas.

**Web Service:** é uma API que sempre usa a internet para comunicação.

#### Exemplos práticos:

- Uma API de um sistema operacional, como a API de câmera de um celular, não depende da internet;
- Um Web Service, como a API de um serviço de mapas, depende da internet para buscar informações sobre rotas.

#### Conclusão:

- Todas as Web Services são APIs, mas nem todas as APIs são Web Services;
- Web Services precisam de uma rede; APIs podem funcionar localmente;
- APIs podem usar diversos métodos de comunicação, enquanto Web Services geralmente usam SOAP, REST ou XML-RPC.

## Como funciona essa comunicação?

Um servidor é um computador ou sistema responsável por armazenar dados ou serviços que podem ser acessados por outros dispositivos, por exemplo, o servidor do Google ou da Amazon.

Na Figura 3.1, é possível visualizar uma página web realizando uma requisição para o servidor do Google.

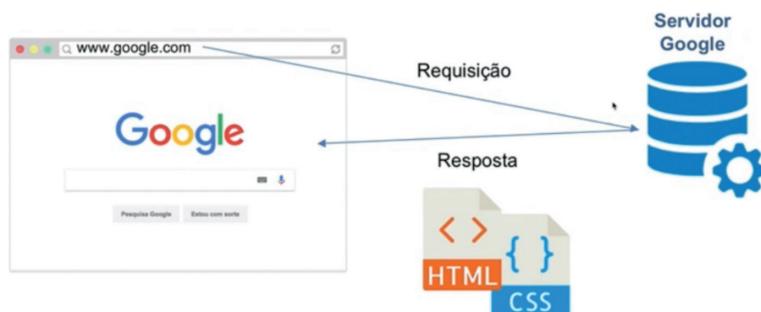


Figura 3.1 – Requisição para o servidor Google.

Esse servidor responde a essa requisição com um código de resposta que pode ser visualizado na Tabela 3.1.

**Tabela 3.1 – Código de status de uma requisição HTTP**

Código de status	Tipo de resposta
200	Ok
201	Created
400	Bad Request
403	Forbidden
500	Internal Server Error
501	Bad Gateway

O formato de resposta dessa requisição pode ser JSON (Java Script Object Notation) ou XML.

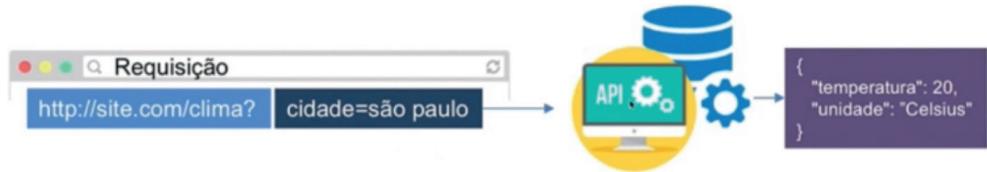
#### SAIBA MAIS

Você pode conferir todos os códigos de status de resposta HTTP no QR Code ou no link a seguir.



Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Reference>Status>.  
Acesso em: 4 ago.2025.

Vamos imaginar que uma API recebe uma requisição sobre o clima na cidade de São Paulo. Na Figura 3.2, é possível visualizar a resposta a essa solicitação em formato JSON, contendo os campos “temperatura” e sua unidade.



**Figura 3.2** – Figura ilustrativa com a resposta no formato JSON.

Já na Figura 3.3, visualizamos a resposta à mesma requisição em formato XML.

#### XML (eXtensible Markup Language)

```
<resultado>
    <temperatura>20</temperatura>
    <unidade>Celsius</unidade>
</resultado>
```

**Figura 3.3** – Representação da resposta no formato XML.

Existem dois tipos de requisição: HTTP e HTTPS. Suas características são apresentadas a seguir.

- HTTP: é um protocolo não seguro. Os dados transmitidos entre cliente e servidor não são criptografados, o que significa que podem ser interceptados e lidos por terceiros. É mais comum em páginas que não lidam com informações sensíveis, como *blogs* ou *sites* informativos.
- HTTPS: é a versão segura do HTTP. Ele utiliza criptografia para proteger os dados transmitidos, impedindo que sejam interceptados ou alterados. Isso é feito por meio de certificados SSL/TLS, que garantem a autenticidade do servidor e a segurança da comunicação.

## Por que o HTTPS é importante para APIs que processam dados sensíveis?

APIs que lidam com informações sensíveis, como dados de login, detalhes de pagamento ou informações pessoais, precisam garantir a segurança das

informações durante a comunicação. O uso do HTTPS é essencial nesses casos porque:

- protege a confidencialidade – os dados são criptografados, tornando-os ilegíveis para interceptadores;
- garante a integridade – evita que os dados sejam alterados durante a transmissão;
- assegura a autenticidade – o cliente pode verificar se está realmente se comunicando com o servidor legítimo.

Portanto, ao desenvolver ou consumir APIs que processam informações críticas, o uso de HTTPS não é apenas uma boa prática, mas um requisito essencial para proteger os dados dos usuários e garantir a confiança na aplicação.

Os principais métodos de requisição HTTP desempenham papéis específicos na comunicação entre cliente e servidor. Eles podem ser descritos da seguinte forma:

- **GET** – utilizado para buscar informações no servidor;
- **POST** – usado para enviar dados ao servidor, como no caso de formulários ou criação de novos registros;
- **PUT** – empregado para atualizar dados existentes no servidor;
- **DELETE** – responsável por excluir dados no servidor.

Além dos métodos de requisição, temos a **resposta HTTP**, que é o retorno do servidor ao cliente após o processamento de uma solicitação. Essa resposta inclui dois elementos principais:

1. **código de status** – indica o resultado da operação, como 200 para “OK” ou 404 para “Não encontrado”;
2. **dados requisitados** – geralmente fornecidos no formato JSON ou XML, contendo as informações solicitadas ou o resultado da operação.

# Requisições HTTP

Veremos a seguir os principais métodos utilizados na requisição HTTP.

## GET

O método GET é usado para solicitar informações de um servidor, principalmente em consultas que não alterem os dados armazenados. Suas características principais são:

- finalidade – solicitar dados de um servidor;
- sem efeito colateral – apenas lê dados sem modificá-los; por isso é considerado um método seguro;
- envio de parâmetros – os dados podem ser enviados como query strings na URL (exemplo: <https://api.exemplo.com/usuarios?id=123>);
- cache – as requisições GET são frequentemente armazenadas em cache pelo navegador, tornando o carregamento mais rápido em requisições repetidas;
- tamanho limitado – como os parâmetros são enviados na URL, há uma limitação no tamanho do que pode ser enviado.

**Exemplo de uso:** obter uma lista de produtos de uma API de e-commerce.

## POST

O método POST no protocolo HTTP é utilizado para enviar dados ao servidor, geralmente para criação ou atualização de recursos. Em Dart, com a ajuda do pacote http, é possível realizar requisições POST de maneira simples e eficiente.

## Características do método POST

- utilizado para envio de dados;
- permite enviar dados no corpo da requisição;
- seguro.

Comumente usado para enviar:

- dados de formulários;
- JSON ou XML;
- arquivos ou blobs binários;
- os dados enviados não aparecem na URL, diferentemente do método GET, o que oferece maior privacidade.

Finalidade:

- criar novos recursos no servidor (exemplo: criar um novo usuário);
- submeter formulários com informações detalhadas.

Resposta:

- geralmente, o servidor retorna um código de status indicando o sucesso ou a falha da operação (ex.: 200 OK, 201 Created, 400 Bad Request).

**Exemplo de uso:** enviar dados de cadastro de um usuário.

Características:

- » altera os dados no servidor;
- » os dados são enviados no corpo da requisição.

Principais erros que ocorrem no POST e precauções a serem tomadas:

- erros de URL – necessário certificar-se de que a URL está correta;
- utilizar Uri.parse para evitar erros de formatação;
- cabeçalhos ausentes – alguns servidores exigem cabeçalhos como Content-Type ou Authorization;

- formato dos dados – necessário verificar se o corpo da requisição está no formato esperado pelo servidor;
- erros de conexão – necessário garantir que o dispositivo tenha acesso à internet.

### Situações de uso do POST:

- autenticação;
- enviar credenciais de login para obter um token de acesso;
- criação de recursos;
- registrar novos usuários, produtos ou pedidos;
- envio de formulários;
- submeter informações coletadas do usuário para processamento.

## PUT

O método PUT no protocolo HTTP é utilizado para atualizar ou substituir completamente um recurso existente em um servidor. Suas características principais são:

- atualização – substitui completamente os dados de um recurso no servidor;
- criação opcional – se o recurso não existir, o servidor pode criar um novo (dependendo da implementação);
- formato de envio – envia dados no corpo da requisição, geralmente no formato JSON;
- comportamento – múltiplas chamadas PUT com os mesmos dados resultam no mesmo estado no servidor;
- resposta – o servidor retorna um código de status para indicar sucesso ou falha.

Casos de uso:

- atualização de perfil (ex.: nome, e-mail ou senha de usuário).

Exemplo:

```
{  
  "nome": "João Atualizado",  
  "email": "joao@example.com"  
}
```

## PATCH

O método PATCH também é usado para atualizar recursos, mas com uma diferença importante em relação ao PUT: ele realiza uma atualização parcial. Apenas os campos fornecidos são alterados. Suas características são:

- atualização parcial – modifica apenas os campos fornecidos;
- eficiência – ideal para operações em que apenas alguns campos precisam ser modificados.

**Exemplo de uso:** atualizar apenas o endereço de um usuário em um cadastro.

## DELETE

O método HTTP DELETE é utilizado para remover recursos de um servidor. Em Dart, com o pacote http, você pode implementar facilmente uma requisição DELETE para APIs RESTful. Suas características principais são:

- finalidade – excluir um recurso específico no servidor;
- corpo de requisição – geralmente não tem body (algumas APIs permitem enviar informações no corpo, dependendo do caso);

- resposta – código de status é retornado como:
  - » 200 OK – recurso deletado com sucesso;
  - » 204 No Content – deleção bem-sucedida sem conteúdo de resposta;
  - » 404 Not Found – recurso não encontrado;
  - » 401 Unauthorized – falta de autorização para realizar a ação.

### Casos de uso:

- excluir usuários – remover contas de um sistema (ex.: DELETE <https://api.exemplo.com/usuarios/1>);
- deletar arquivos – excluir arquivos armazenados em um servidor;
- gerenciamento de recursos – remover registros, produtos ou qualquer dado não mais necessário.

**Exemplo de uso:** deletar usuário pelo ID.

## Requisição do tipo GET

Como mencionado, a requisição GET é um dos métodos HTTP mais comuns usados para obter dados de um servidor. É amplamente utilizada quando você precisa buscar informações sem enviar nenhum dado sensível ou alterar recursos no servidor.

Como exemplo da requisição GET será desenvolvido um aplicativo em Flutter para consultar um CEP e exibir as informações no terminal do VS CODE. Para verificar como criar um projeto Flutter no VS CODE, consulte o Capítulo 2.

Para realizar requisições HTTP utilizando Dart, é necessário adicionar o package `http` como dependência do projeto no arquivo `pubspec.yaml`.

Este arquivo `pubspec.yaml` é usado em projetos Flutter para gerenciar informações, dependências, configurações de ambiente e ativos. Vamos detalhar o conteúdo.

O primeiro passo é adicionar a dependência HTTP. Para isso é necessário ir em pubspec.yaml, conforme vemos na Figura 3.4.

```
# The following adds the Cupertino Icons font to your application.  
# Use with the CupertinoIcons class for iOS style icons.  
cupertino_icons: ^1.0.2  
http: ^0.13.3
```

**Figura 3.4** – Arquivo pubspec.yaml.

Após adicionar a dependência HTTP no pubspec.yaml, deve-se executar no terminal o comando flutter pub get para atualizar todos os pacotes e dependências do projeto.

A seguir, é possível visualizar o código do arquivo pubspec.yaml.

#### Arquivo pubspec.yaml

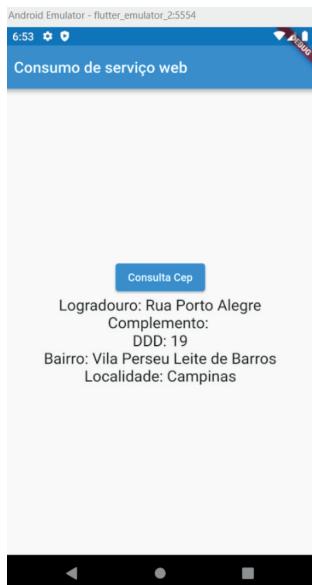
1. name: aula06\_api\_http\_tb
2. description: A new Flutter project.
3. # The following line prevents the package from being accidentally published to
4. # pub.dev using `flutter pub publish`. This is preferred for private packages.
5. publish\_to: ‘none’ # Remove this line if you wish to publish to pub.dev
- 6.
7. # The following defines the version and build number for your application.
8. # A version number is three numbers separated by dots, like 1.2.43
9. # followed by an optional build number separated by a +.
10. # Both the version and the builder number may be overridden in flutter
11. # build by specifying --build-name and --build-number, respectively.
12. # In Android, build-name is used as versionName while build-number used as versionCode.

```
13. # Read more about Android versioning at https://developer.android.com/studio/publish/versioning
14. # In iOS, build-name is used as CFBundleShortVersionString while build-number is used as CFBundleVersion.
15. # Read more about iOS versioning at
16. # https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html
17. # In Windows, build-name is used as the major, minor, and patch parts
18. # of the product and file versions while build-number is used as the build suffix.
19. version: 1.0.0+1
20.
21. environment:
22.   sdk: '>=3.0.6 <4.0.0'
23.
24. # Dependencies specify other packages that your package needs in order to work.
25. # To automatically upgrade your package dependencies to the latest versions
26. # consider running `flutter pub upgrade --major-versions`.
  Alternatively,
27. # dependencies can be manually updated by changing the version numbers below to
28. # the latest version available on pub.dev. To see which dependencies have newer
29. # versions available, run `flutter pub outdated`.
30. dependencies:
31.   flutter:
32.     sdk: flutter
33.
34.
35. # The following adds the Cupertino Icons font to your application.
```

```
36. # Use with the CupertinoIcons class for iOS style icons.
37. cupertino_icons: ^1.0.2
38. http: ^0.13.3      # instala o pacote http
39. dev_dependencies:
40.   flutter_test:
41.     sdk: flutter
42.
43.   # The “flutter_lints” package below contains a set of
44.   # recommended lints to
45.   # encourage good coding practices. The lint set
46.   # provided by the package is
47.   # activated in the `analysis_options.yaml` file located
48.   # at the root of your
49.   # package. See that file for information about
50.   # deactivating specific lint
51.   # rules and activating additional ones.
52.   flutter_lints: ^2.0.0
53.
54. # For information on the generic Dart part of this file,
55. # see the
56. # following page: https://dart.dev/tools/pub/pubspec
57.
58. # The following section is specific to Flutter packages.
59. flutter:
60.
61.   # The following line ensures that the Material Icons
62.   # font is
63.   # included with your application, so that you can use
64.   # the icons in
65.   # the material Icons class.
66.   uses-material-design: true
67.
68.   # To add assets to your application, add an assets
69.   # section, like this:
70.
71.   # assets:
```

```
63. # - images/a_dot_burr.jpeg
64. # - images/a_dot_ham.jpeg
65.
66. # An image asset can refer to one or more resolution-
   specific "variants", see
67. # https://flutter.dev/assets-and-images/#resolution-aware
68.
69. # For details regarding adding assets from package
   dependencies, see
70. # https://flutter.dev/assets-and-images/#from-packages
71.
72. # To add custom fonts to your application, add a fonts
   section here,
73. # in this "flutter" section. Each entry in this list
   should have a
74. # "family" key with the font family name, and a "fonts"
   key with a
75. # list giving the asset and other descriptors for the
   font. For
76. # example:
77. # fonts:
78. # - family: Schyler
79. #   fonts:
80. #     - asset: fonts/Schyler-Regular.ttf
81. #     - asset: fonts/Schyler-Italic.ttf
82. #       style: italic
83. # - family: Trajan Pro
84. #   fonts:
85. #     - asset: fonts/TrajanPro.ttf
86. #     - asset: fonts/TrajanPro_Bold.ttf
87. #       weight: 700
88. #
89. # For details regarding fonts from package
   dependencies,
90. # see https://flutter.dev/custom-fonts/#from-packages
91.
```

Na Figura 3.5, conseguimos visualizar a tela do aplicativo exibindo as informações de uma consulta de CEP.



**Figura 3.5** – Tela do aplicativo para consultar CEP.

A seguir, é possível observar o código da tela inicial (home) de um aplicativo para consulta de CEP.

#### Código da tela inicial

```
1. //Importação das bibliotecas
2. // Importa os widgets e ferramentas do Flutter
   necessários para a interface.
3.
4. import 'package:flutter/material.dart';
5. // Importa a biblioteca HTTP para realizar requisições à
   API.
6.
7. //package:http/http.dart
8.
9. // permite os métodos para consumo da api
10. //Importa ferramentas para converter dados JSON em
    estruturas utilizáveis, como //Map.
```

```
11.
12. import 'dart:convert'; // pacote que permite a conversao
   dos dados
13.
14. class Home extends StatefulWidget {
15.   const Home({super.key});
16.
17.   @override
18.   State<Home> createState() => _HomeState();
19. }
20.
21. class _HomeState extends State<Home> {
22.   TextEditingController ncep = TextEditingController();
23.   String ? logradouro;
24.   String? bairro;
25.   String ? cidade;
26.   String ? estado;
27.   String? ddd;
28.   // async é porque a comunicação é assincrona
29.   _consultaCep () async {
30.     String url = "https://cep.awesomeapi.com.br/${ncep.
   text}";
31.     // api para consultar o endereço através do cep
32.     http.Response response; // variavel para armazenar a
   resposta da api
33.     response = await http.get(Uri.parse(url)); // response
   armazena a resposta da api
34.     print("Codigo de status da API: ${response.statusCode.
   toString()}"); // codigo de resposta da api
35.     //print("Resposta da api ${response.body}"); //
   resposta da api
36.
37. // Map<String, dynamic> dados = json.decode(response.body);
38. //Converte o corpo da resposta (JSON) em um mapa.
39. //Atualização do Estado
40.
41. Map<String,dynamic>dados = json.decode(response.body);
```

```
42.     setState(() {
43.         logradouro = dados[“address”];
44.         bairro = dados[“district”];
45.         cidade = dados[“city”];
46.         estado = dados[“state”];
47.         ddd = dados[“ddd”];
48.
49.     }); // variavel para armazenar os dados da api
50.
51.     print(“Logradouro: $logradouro”);
52.     print(“Bairro: $bairro”);
53.     print(“cidade: $cidade - $estado”);
54.     print(“ddd: $ddd”);
55. }
56. @override
57. Widget build(BuildContext context) {
58.     return Scaffold(
59.         appBar: AppBar(
60.             title: Text(“App consulta CEP”),  

61.
62.         ),
63.         body: Column(
64.             mainAxisAlignment: MainAxisAlignment.center,
65.             children: [
66.                 Padding(
67.                     padding: const EdgeInsets.all(8.0),
68.                     child: TextField(
69.                         keyboardType: TextInputType.number,
70.                         decoration: InputDecoration(labelText:
“Digite o cep”),  

71.                         controller: ncep ,
72.
73.
74.                     ),
75.                 ),
76.                 Center(child: ElevatedButton(onPressed: _  

consultaCep, child: Text(“Verificar”))),
```

```

77.           Text("Logradouro: $logradouro ",style:
    TextStyle(fontSize: 18),),
78.           Text("Bairro: $bairro ",style:
    TextStyle(fontSize: 18),),
79.           Text("Cidade: $cidade - $estado ",style:
    TextStyle(fontSize: 18),),
80.           Text("DDD: $ddd ",style: TextStyle(fontSize:
    18),),
81.           ],
82.           ),
83.           );
84.       }
85.   }
86.

```

Temos, a seguir, o código principal do aplicativo main.dart.

#### Código principal do aplicativo main.dart

```

1. import 'package:aula06_api_http_tb/home.dart';
2. import 'package:flutter/material.dart';
3.
4. void main() {
5.   runApp(MaterialApp(
6.     home: Home(),
7.   ));
8. }

```

O fluxo de execução desse aplicativo pode ser resumido como:

1. o usuário digita um CEP no campo de texto;
2. pressiona o botão “Verificar”;
3. o aplicativo faz uma requisição para a API com o CEP fornecido;
4. os dados retornados pela API são convertidos em um mapa e armazenados nas variáveis;

5. o estado da interface é atualizado com as informações retornadas;
6. os dados (logradouro, bairro, cidade, estado, DDD) são exibidos na tela.

#### SAIBA MAIS

É possível obter o código completo do aplicativo no QR Code ou no link a seguir.



Disponível em: [https://github.com/danielvieira95/DESM-2/tree/master/Apps-TB/aula06\\_api\\_http\\_tb](https://github.com/danielvieira95/DESM-2/tree/master/Apps-TB/aula06_api_http_tb). Acesso em: 19 ago. 2025.

Para exemplificar o uso do método GET, utilizaremos, como exemplo, o desenvolvimento de um aplicativo para consultar o preço de bitcoins e realizar a conversão para moedas como euro, dólar e real. Na Figura 3.6, é possível visualizar a interface do aplicativo.



**Figura 3.6** – Tela do aplicativo para consultar o preço de bitcoins.

A seguir, é apresentado o código da tela inicial (home) do aplicativo.

### Código da home do aplicativo

```
1. // Importações
2. //dart:convert: Para decodificar os dados JSON retornados
   pela API.
3. //flutter/material.dart: Para criar a interface do
   aplicativo.
4. // http/http.dart: Para realizar requisições HTTP à API.
5.
6. import 'dart:convert';
7.
8. import 'package:flutter/material.dart';
9. import 'package:flutter/rendering.dart';
10. import 'package:http/http.dart' as http;
11. class Home extends StatefulWidget {
12.   const Home({super.key});
13.
14.   @override
15.   State<Home> createState() => _HomeState();
16. }
17.
18. // Classe Home
19.
20. //Um widget com estado que implementa as duas
   funcionalidades principais do // aplicativo.
21. //Classe _HomeState
22.
23.
24. class _HomeState extends State<Home> {
25.   // Gerencia o estado e contém a lógica do aplicativo.
26.   //Atributos
27.   // TextEditingController valor
28.
29.   Controlador para capturar o valor inserido no campo de
      texto pelo usuário.
```

```
30. int? selectop, selectop1
31.
32. Armazena as opções selecionadas (origem e destino) para
    a conversão de moeda.
33. double vdolar, veuro, vreais
34.
35. TextEditingController valor = TextEditingController();
36. int? selectop , selectop1;
37. double vdolar =0;
38. double veuro =0;
39. double vreais =0;
40. String preco = “0”;
41. _consultaBitCoin() async{
42.     String url = “https://blockchain.info/ticker”; // url
        com a api
43.     http.Response response = await http.get(Uri.
            parse(url)) ; // espera o retorno da api
44.     Map<String, dynamic> dados = json.decode(response.
            body); // decodifica o dado da api
45.     setState(() {
46.         preco = dados[“BRL”][“buy”].toString(); // faz o
            parse da informação
47.     });
48.
49.     print(“Resultado api ${response.statusCode.
            toString()}”);
50.     print(“Valores BitCoin ${dados}”);
51.     print(“Valor do bitcoin em R\$ ${preco}”);
52. }
53. _converter(){
54.     if(selectop ==0 && selectop1 ==1)
55.         { setState(() {
56.             vdolar = double.parse(valor.text);
57.         });
58.
59.         }
60.         else if(selectop ==0 &&
```

```
61. selector1 ==3)
62. {
63.     setState(() {
64.         vdolar = double.parse(valor.text) /5.01;
65.     });
66.
67. }
68. else if(selector ==0 &&
69. selector1 ==5)
70. {
71.     setState(() {
72.         vdolar = double.parse(valor.text)/5.45;
73.     });
74.
75. }
76. else if(selector ==2 &&
77. selector1 ==1)
78. {
79.     setState(() {
80.         vdolar = double.parse(valor.text)*5.01;
81.     });
82.
83. }
84. else if(selector ==2 &&
85. selector1 ==3)
86. {
87.     setState(() {
88.         vdolar = double.parse(valor.text);
89.     });
90.
91. }
92. else if(selector ==2 &&
93. selector1 ==5)
94. {
95.     setState(() {
96.         vdolar = double.parse(valor.text)*0.92;
97.     });
}
```

```
98.  
99.        }  
100.       else if(selectop ==3 &&  
101. selectop1 ==1)  
102.       {  
103.           setState(() {  
104.               vdolar = double.parse(valor.text)*0.18;  
105.           });  
106.  
107.       }  
108.       else if(selectop ==4 &&  
109. selectop1 ==3)  
110.       {  
111.           setState(() {  
112.               vdolar = double.parse(valor.text)*0.91;  
113.           });  
114.  
115.       }  
116.       else if(selectop ==4 &&  
117. selectop1 ==5)  
118.       {  
119.           setState(() {  
120.               vdolar = double.parse(valor.text);  
121.           });  
122.  
123.       }  
124.       else if(selectop ==4 &&  
125. selectop1 ==1)  
126.       {  
127.           setState(() {  
128.               vdolar = double.parse(valor.text)*5.45;  
129.           });  
130.  
131.       }  
132.  
133.  
134.
```

```
135.        }
136.        _limpar(){
137.            setState(() {
138.                vdolar = 0;
139.                valor.text = “”;
140.
141.            });
142.        }
143.
144.
145.
146.
147.    @override
148.    Widget build(BuildContext context) {
149.        return Scaffold(
150.            appBar: AppBar(
151.                title: Text(“App consulta preço BitCoin”),
152.
153.            ),
154.            body:
155.                ListView(
156.
157.                    scrollDirection: Axis.vertical,
158.                    children: [
159.                        Container(width: 250,height: 150,color:
160.                            Colors.amber,child: Image.asset(‘imagens/bitcoin.png’),),
161.                        Padding(
162.                            padding: const EdgeInsets.all(50.0),
163.                            child: Text(“Valor BitCoin R\$: $preco
164.                            ”,style: TextStyle(fontSize: 20),),
165.                        ),
166.
167.                        TextField(
168.                            keyboardType: TextInputType.number,
169.                            decoration: InputDecoration(labelText:
“Digite o valor a ser convertido”),
```

```
169.           ),
170.
171.
172.           Row(
173.             mainAxisAlignment: MainAxisAlignment.
174.             spaceEvenly,
175.             children: [
176.               Text("Origem",style: TextStyle(fontSize:
177.                 15)),
178.
179.               Text("Destino",style: TextStyle(fontSize:
180.                 15)),
181.               Container(color: Colors.white,width:
182.                 450,height: 150,
183.                 child: Column(
184.                   children: [
185.                     Row(
186.                       mainAxisAlignment: MainAxisAlignment.
187.                         spaceEvenly,
188.                         children: [
189.                           Radio(
190.                             value: 0,
191.                             groupValue: selectop,
192.                             onChanged: (value){
193.                               setState(() {
194.                                 selectop = value;
195.                               });
196.                             },
197.                             Text("R\$"),
198.                             Radio(
199.                               value: 1,
200.                               groupValue: selectop1,
```

```
201.                               selectop1 = value;
202.                           });
203.                         },
204.                         Text("R\$"),
205.                       ],
206.                     ),
207.                     Row(
208.                       mainAxisAlignment: MainAxisAlignment.
209.                         spaceEvenly,
210.                         children: [
211.                           Radio(
212.                             value: 2,
213.                             groupValue: selectop,
214.                             onChanged: (value){
215.                               setState(() {
216.                                 selectop = value;
217.                               });
218.                               Text("Dólar"),
219.                               Radio(
220.                                 value: 3,
221.                                 groupValue: selectop1,
222.                                 onChanged: (value){
223.                                   setState(() {
224.                                     selectop1 = value;
225.                                   });
226.                                 });
227.                               Text("Dólar"),
228.                             ],
229.                           ),
230.                           Row(
231.                             mainAxisAlignment: MainAxisAlignment.
232.                               spaceEvenly,
233.                               children: [
234.                                 Radio(
235.                                   value: 4,
```



```

269.           mainAxisAlignment: MainAxisAlignment.
270.             spaceEvenly,
271.             children: [
272.               ElevatedButton(onPressed: _consultaBitCoin,
273.                 child: Text("Verificar")),
274.               ElevatedButton(onPressed: _converter, child:
275.                 Text("Calcular")),
276.               ElevatedButton(onPressed: _limpar, child:
277.                 Text("limpar")),
278.
279.
280.
281.
282.
283.
284.
285.               //Text("Opção: $selectop"),
286.               // Text("Opção: $selectop1"),
287.             ],
288.           );
289.
290.
291.
292.     }
293. }
```

A seguir, é possível visualizar o código principal main.dart.

Código principal main.dart do aplicativo de consulta de bitcoins

1. import 'package:aula07\_app\_bitcoin/home.dart';
2. import 'package:flutter/material.dart';
- 3.

```
4. void main() {  
5.   runApp(MaterialApp(  
6.     home: Home(),  
7.   ));  
8. }
```

A seguir, o código do arquivo pubspec.yaml é apresentado. O arquivo pubspec.yaml é uma parte essencial do projeto Flutter, configurando informações básicas do aplicativo, dependências e ativos.

Arquivo pubspec.yaml com as dependências do projeto

```
1. name: aula07_app_bitcoin  
2. description: A new Flutter project.  
3. # The following line prevents the package from being  
# accidentally published to  
4. # pub.dev using `flutter pub publish`. This is preferred  
# for private packages.  
5. publish_to: 'none' # Remove this line if you wish to  
# publish to pub.dev  
6.  
7. # The following defines the version and build number for  
# your application.  
8. # A version number is three numbers separated by dots,  
# like 1.2.43  
9. # followed by an optional build number separated by a +.  
10. # Both the version and the builder number may be  
# overridden in flutter  
11. # build by specifying --build-name and --build-number,  
# respectively.  
12. # In Android, build-name is used as versionName while  
# build-number used as versionCode.  
13. # Read more about Android versioning at https://developer.  
# android.com/studio/publish/versioning  
14. # In iOS, build-name is used as CFBundleShortVersionString  
# while build-number is used as CFBundleVersion.
```

```
15. # Read more about iOS versioning at
16. # https://developer.apple.com/library/archive/documentation/
   General/Reference/InfoPlistKeyReference/Articles/
   CoreFoundationKeys.html
17. # In Windows, build-name is used as the major, minor, and
   patch parts
18. # of the product and file versions while build-number is
   used as the build suffix.
19. version: 1.0.0+1
20.
21. environment:
22.   sdk: '>=3.0.6 <4.0.0'
23.
24. # Dependencies specify other packages that your package
   needs in order to work.
25. # To automatically upgrade your package dependencies to
   the latest versions
26. # consider running `flutter pub upgrade --major-versions`.
   Alternatively,
27. # dependencies can be manually updated by changing the
   version numbers below to
28. # the latest version available on pub.dev. To see which
   dependencies have newer
29. # versions available, run `flutter pub outdated`.
30. dependencies:
31.   flutter:
32.     sdk: flutter
33.
34.
35.   # The following adds the Cupertino Icons font to your
   application.
36.   # Use with the CupertinoIcons class for iOS style icons.
37.   cupertino_icons: ^1.0.2
38.   http: ^0.13.3 # método para permitir o consumo da api
39.
```

```
40. dev_dependencies:  
41.   flutter_test:  
42.     sdk: flutter  
43.  
44.   # The “flutter_lints” package below contains a set of  
      recommended lints to  
45.   # encourage good coding practices. The lint set  
      provided by the package is  
46.   # activated in the `analysis_options.yaml` file located  
      at the root of your  
47.   # package. See that file for information about  
      deactivating specific lint  
48.   # rules and activating additional ones.  
49.   flutter_lints: ^2.0.0  
50.  
51. # For information on the generic Dart part of this file,  
  see the  
52. # following page: https://dart.dev/tools/pub/pubspec  
53.  
54. # The following section is specific to Flutter packages.  
55. flutter:  
56.  
57.   # The following line ensures that the Material Icons  
      font is  
58.   # included with your application, so that you can use  
      the icons in  
59.   # the material Icons class.  
60.   uses-material-design: true  
61.  
62.   # To add assets to your application, add an assets  
      section, like this:  
63.   assets:  
64.     - imagens/bitcoin.png  
65.     #     - images/a_dot_ham.jpeg  
66.
```

```
67. # An image asset can refer to one or more resolution-
    specific “variants”, see
68. # https://flutter.dev/assets-and-images/#resolution-aware
69.
70. # For details regarding adding assets from package
    dependencies, see
71. # https://flutter.dev/assets-and-images/#from-packages
72.
73. # To add custom fonts to your application, add a fonts
    section here,
74. # in this “flutter” section. Each entry in this list
    should have a
75. # “family” key with the font family name, and a “fonts”
    key with a
76. # list giving the asset and other descriptors for the
    font. For
77. # example:
78. # fonts:
79. #   - family: Schyler
80. #     fonts:
81. #       - asset: fonts/Schyler-Regular.ttf
82. #       - asset: fonts/Schyler-Italic.ttf
83. #         style: italic
84. #   - family: Trajan Pro
85. #     fonts:
86. #       - asset: fonts/TrajanPro.ttf
87. #       - asset: fonts/TrajanPro_Bold.ttf
88. #         weight: 700
89. #
90. # For details regarding fonts from package
    dependencies,
91. # see
92.
93. https://flutter.dev/custom-fonts/#from-packages
```

**SAIBA MAIS**

É possível obter o código completo do aplicativo de consulta de bitcoin no QR Code ou no link a seguir.



Disponível em: [https://github.com/danielvieira95/DESM-2/blob/master/Apps-TA/app\\_aula07\\_bitcoin/lib/home.dart](https://github.com/danielvieira95/DESM-2/blob/master/Apps-TA/app_aula07_bitcoin/lib/home.dart). Acesso em: 19 ago. 2025.

Para que o aplicativo acesse a internet, é necessário adicionar permissão, conforme mostra a Figura 3.7.

Para permitir que um aplicativo Flutter acesse a internet, você precisa definir permissões no arquivo **‘AndroidManifest.xml’** para dispositivos Android e no arquivo **‘Info.plist’** para dispositivos iOS.

#### **Para dispositivos Android:**

Você precisa adicionar a permissão **‘INTERNET’** no arquivo **‘android/app/src/main/AndroidManifest.xml’**. Certifique-se de que o seguinte código esteja presente dentro da tag **‘<manifest>’**:

```
xml
Copy code

<uses-permission android:name="android.permission.INTERNET"/>
```

**Figura 3.7** – Adicionando permissões no Android Manifest.

## Requisição do tipo POST

O método POST envia dados no corpo da requisição, garantindo maior segurança e capacidade de envio de informações complexas, enquanto o

GET transmite os dados diretamente na URL, tornando-os visíveis e mais adequados para solicitações simples.

Para realizar um POST em linguagem Dart, é necessário realizar os passos indicados no código a seguir.

### Estrutura do POST em linguagem Dart

```
1. // 1. Importação do Pacote
2. // É necessário importar o pacote http:
3.
4. import 'package:http/http.dart' as http; import
   'dart:convert'; // Para codificação/decodificação de JSON.
5. // 2. URL do Endpoint
6. //A URL do endpoint é o destino para //onde os dados serão
   enviados:
7.
8.
9. String url = "https://api.exemplo.com/endpoint";
10.
11. // 3. Dados no Corpo da Requisição
12. //Os dados enviados geralmente estão //no formato JSON:
13.
14. Map<String, dynamic> body = {
15.   "nome": "João",
16.   "email": "joao@example.com",
17.   "senha": "123456"
18. };
19. // 4. Requisição POST
20. http.Response response = await http.post(
21.   Uri.parse(url),
22.   headers: {"Content-Type": "application/json"}, //
   Cabeçalhos
23.   body: json.encode(body), // Codificar os dados como JSON
24. );
25.
```

A seguir, vemos o exemplo completo do POST em Dart.

### Exemplo completo do POST em Dart

```
1. import 'package:http/http.dart' as http;
2. import 'dart:convert';
3.
4. /*
5. Cabeçalhos Importantes
6. Os cabeçalhos são usados para informar ao servidor como
os dados devem ser processados:
7. */
8.
9. void realizarPost() async {
10.   // URL do servidor
11.   String url = "https://api.exemplo.com/usuarios";
12.
13.   // Corpo da requisição
14.   Map<String, dynamic> dados = {
15.     "nome": "Maria",
16.     "email": "maria@example.com",
17.     "senha": "senha123"
18.   };
19.
20.   try {
21.
22.     /*
23.     Content-Type
24.
25.     Define o formato dos dados enviados no corpo. Exemplo:
26.     application/json: Indica que os dados estão no formato
JSON.
27.     application/x-www-form-urlencoded: Dados no formato de
formulário HTML.
28.     Authorization
29.
30.
```

```
31. */
32.
33. /*
34. Decodificação
35.
36. Caso a resposta seja JSON, pode-se decodificar para um Map:
37. dart
38.
39.
40. */
41.     // Enviar a requisição POST
42.     http.Response response = await http.post(
43.         Uri.parse(url),
44.         headers: {"Content-Type": "application/json"},
45.         body: json.encode(dados),
46.     );
47.
48.     // Verificar o código de status
49.
50. /*
51. Status Codes
52.
53. 200 OK: Sucesso.
54. 201 Created: Recurso criado com sucesso.
55. 400 Bad Request: Erro nos dados enviados.
56. 401 Unauthorized: Falta de autenticação.
57. 500 Internal Server Error: Erro no servidor.
58.
59.
60. */
61.     if (response.statusCode == 201) {
62.         print("Usuário criado com sucesso!");
63.         print("Resposta: ${response.body}");
64.     } else {
65.
66. /*
67.
```

```
68. Map<String, dynamic> resposta = json.decode(response.body);
69. print("ID do usuário criado: ${resposta['id']}");
70.
71. http.ClientException: Captura erros relacionados à conexão
   com o servidor.
72. FormatException: Trata erros no formato da resposta (por
   exemplo, JSON malformado).
73. Exception: Captura quaisquer outros erros inesperados.
74. Mensagens mais específicas:
75.
76.
77. */
78.     print("Erro ao criar o usuário. Código: ${response.
   statusCode}");
79.     print("Resposta: ${response.body}");
80. }
81. } on http.ClientException catch (e) {
82.     print("Erro na conexão com o servidor: $e");
83. } on FormatException catch (e) {
84.     print("Erro ao processar a resposta: $e");
85. } on Exception catch (e) {
86.     print("Erro inesperado: $e");
87. }
88. }
89.
```

## Requisição do tipo PUT

A seguir, é possível visualizar a estrutura do método PUT em Dart.

### Estrutura método PUT

1. /\*
2. Uma requisição PUT em Dart segue a mesma estrutura do POST, com algumas adaptações:

```
3. como o id do valor que se deseja realizar alteração
4. */
5. http.Response response = await http.put(
6.   Uri.parse("https://api.exemplo.com/recurso/1"),
7.   headers: {"Content-Type": "application/json"},
8.   body: json.encode({
9.     "nome": "João Atualizado",
10.    "email": "joao_atualizado@example.com"
11.  }),
12. );
13.
```

A seguir, é possível visualizar o exemplo completo da requisição PUT em Dart.

#### Código completo do método PUT

```
1. import 'dart:convert'; // Para codificar e decodificar JSON
   import 'package:http/http.dart' as http; // Para requisições
   HTTP
2.
3. void atualizarRecurso() async {
4.   // URL do recurso no servidor
5.   String url = "https://api.exemplo.com/usuarios/1";
6.
7.   // Dados atualizados que serão enviados
8.   Map<String, dynamic> dadosAtualizados = {
9.     "nome": "Maria Atualizada",
10.    "email": "maria_atualizada@example.com"
11.  };
12.
13. /*
14.
15. Tratamento de Erros
16.
17. O try-catch captura possíveis erros, como problemas de
   conexão.
```

```
18.  
19. */  
20.  
21.     try {  
22.         /*  
23.          * Content-Type: application/json: Indica que os dados no  
24.            corpo estão no formato JSON.  
25.          * Corpo da Requisição  
26.          * json.encode(dadosAtualizados): Transforma o mapa  
27.            dadosAtualizados em uma string JSON.  
28.          * Tratamento de Respostas  
29.          *  
30.          */  
31.         // Realizando a requisição PUT  
32.         http.Response response = await http.put(  
33.             Uri.parse(url),  
34.             headers: {"Content-Type": "application/json"},  
35.             body: json.encode(dadosAtualizados),  
36.             );  
37.         /*  
38.          * O código verifica o código de status da resposta para  
39.            determinar se a operação foi bem-sucedida.  
40.          * Códigos Comuns:  
41.          * 200 OK: Recurso atualizado com sucesso.  
42.          * 201 Created: Recurso criado (em alguns casos).  
43.          * 404 Not Found: O recurso não existe.  
44.          * 400 Bad Request: Erro nos dados enviados.  
45.          *  
46.          */  
47.         // Verificar o status da resposta
```

```

50.     if (response.statusCode == 200) {
51.         print("Recurso atualizado com sucesso!");
52.         print("Resposta: ${response.body}");
53.     } else {
54.         print("Erro ao atualizar recurso. Código:
55.             ${response.statusCode}");
56.     }
57. } catch (e) {
58.     print("Erro na requisição: $e");
59. }
60. }
61.

```

## Diferenças entre PUT e PATCH

Embora ambos sejam usados para atualizar recursos, existem diferenças importantes:

- PUT – substitui completamente o recurso, e com ele todos os campos precisam ser enviados, mesmo os que não mudaram;
- PATCH – atualiza apenas os campos especificados; é mais eficiente se apenas alguns campos precisam ser alterados.

A seguir, um exemplo com PATCH é apresentado.

### Exemplo com PATCH

```

1. /*
2. Requisição PATCH
3.
4.
5. http.Response response = await http.patch(...)
6. Realiza a requisição PATCH de forma assíncrona e aguarda
   a resposta.

```

```
7. Resposta do Servidor
8. O objeto response contém a resposta do servidor. É
   importante analisar os seguintes aspectos:
9.
10. Status Code
11.
12. Indica o resultado da requisição.
13. Exemplos:
14. 200 OK: Atualização bem-sucedida.
15. 204 No Content: Atualização bem-sucedida sem corpo de
   resposta.
16. 400 Bad Request: Erro nos dados enviados.
17. 404 Not Found: O recurso não foi encontrado.
18. Corpo da Resposta
19.
20. Pode conter informações adicionais, como o estado
   atualizado do recurso.
21.
22. */
23. http.Response response = await http.patch
24. (
25. Uri.parse(
26. "https://api.exemplo.com/usuarios/1"),
27. headers: {
28. "Content-Type": "application/json"},
29. body: json.encode(
30. { "email": "novo_email@example.com" }
31. ),
32. );
```

O exemplo completo da requisição PUT é apresentado a seguir.

#### Código completo da requisição PUT

```
1. import 'dart:convert'; // Para codificar e decodificar JSON
2. import 'package:http/http.dart' as http;
```

```
3.  
4. void atualizarEmailUsuario() async {  
5.   String url = "https://api.exemplo.com/usuarios/1";  
6.  
7.   // Dados a serem atualizados  
8.   Map<String, dynamic> dadosAtualizados = {  
9.     "email": "novo_email@example.com"  
10.    };  
11.  
12.  try {  
13.    // Requisição PATCH  
14.    http.Response response = await http.patch(  
15.      Uri.parse(url),  
16.      headers: {"Content-Type": "application/json"},  
17.      body: json.encode(dadosAtualizados),  
18.    );  
19.  
20.    // Verifica o status da resposta  
21.    if (response.statusCode == 200 || response.statusCode  
== 204) {  
22.      print("Atualização bem-sucedida!");  
23.      if (response.body.isNotEmpty) {  
24.        Map<String, dynamic> resposta = json.  
decode(response.body);  
25.        print("Dados atualizados: $resposta");  
26.      }  
27.    } else {  
28.      print("Erro ao atualizar. Código: ${response.  
statusCode}");  
29.      print("Resposta do servidor: ${response.body}");  
30.    }  
31.  } catch (e) {  
32.    print("Erro na requisição: $e");  
33.  }  
34.
```

As diferenças entre os métodos PATCH e PUT estão resumidas na tabela a seguir.

**Tabela 3.2 – Diferença entre PATCH e PUT**

PATCH	PUT
Atualiza <b>parcialmente</b> um recurso.	Substitui o recurso por completo.
Envia apenas os campos que serão alterados.	Envia todos os campos do recurso.
Mais eficiente para pequenas alterações.	Útil para substituir completamente o recurso.

## Requisição do tipo DELETE

A requisição http.delete é utilizada para deletar alguma informação da API e pode ser visualizada a seguir.

Exemplo de requisição DELETE

```

1. import 'package:http/http.dart' as http;
2.
3. void deletarUsuario() async {
4.   // URL do recurso a ser deletado
5.   String url = "https://api.exemplo.com/usuarios/1";
6.
7.   try {
8.     // Requisição DELETE
9.     http.Response response = await http.delete(Uri.
    parse(url));
10.
11.    // Verificar o status da resposta
12.    if (response.statusCode == 200 || response.statusCode
    == 204) {
13.      print("Usuário deletado com sucesso!");
14.    } else {
15.      print("Erro ao deletar usuário. Código: ${response.
        statusCode}");
}

```

```

16.     print("Resposta do servidor: ${response.body}");
17.   }
18. } catch (e) {
19.   print("Erro na requisição: $e");
20. }
21. }
22.

```

Caso a API exija autenticação, o token de autorização pode ser inserido conforme o exemplo apresentado a seguir.

Exemplo de requisição DELETE quando a API exige autenticação

```

1. /*
2.
3. Em Authorization o token de autorização deve ser
   inserido para permitir deletar uma informação da API.
4. */
5.
6. http.Response response = await http.delete(
7.   Uri.parse(url),
8.   headers: {
9.     "Authorization": "Bearer seu_token_aqui",
10.    "Content-Type": "application/json",
11.  },
12. );

```

A seguir, é possível visualizar outro exemplo, mais completo, da requisição DELETE quando necessita realizar autenticação.

Exemplo de requisição DELETE com autenticação

```

1. // package:http: Biblioteca usada para realizar requisições
   HTTP como GET, POST, PUT, PATCH e DELETE.
2.
3. import 'package:http/http.dart' as http;

```

```
4.
5. void deletarUsuarioComAutenticacao() async {
6.     String url = "https://api.exemplo.com/usuarios/1";
7.
8.     try {
9.         http.Response response = await http.delete(
10.             Uri.parse(url),
11.             headers: {
12.                 "Authorization": "Bearer seu_token_aqui",
13.                 "Content-Type": "application/json",
14.             },
15.         );
16.     /*
17.     response.statusCode:
18.     Retorna o código de status da resposta do servidor.
19.     200 OK: Indica que o recurso foi deletado com sucesso e
20.     uma resposta detalhada pode ser enviada.
21.     204 No Content: Indica que o recurso foi deletado com
22.     sucesso, mas sem resposta no corpo.
23.     7. Mensagem de Erro
24.     */
25.     if (response.statusCode == 200 || response.statusCode
26.         == 204) {
27.         print("Usuário deletado com sucesso!");
28.     } else {
29.         print("Erro ao deletar usuário. Código: ${response.
30.             statusCode}");
31.     }
32. } catch (e) {
33.     print("Erro na requisição: $e");
34. }
35.
```

# Manipulação de dados

A manipulação de dados é fundamental para realizar o tratamento dos dados provenientes de APIs. Ela pode ser necessária, por exemplo, após a realização de requisições http.get. Portanto, é importante conhecer o tipo de dado que uma requisição GET pode retornar como JSON ou XML. É o que veremos a seguir.

## JSON

JSON (JavaScript Object Notation) é um formato leve de troca de dados, amplamente utilizado para comunicação entre sistemas e APIs. Em Dart, é possível manipular JSON de maneira eficiente com a biblioteca padrão `dart:convert`.

O JSON, apresentado a seguir, descreve uma lista de produtos, cada um com propriedades como `id`, `nome` e `valor`.

### Formato JSON

```
1. /*
2. produtos: Uma lista (array) de objetos.
3. Cada objeto dentro da lista contém:
4. id: Identificador do produto.
5. nome: Nome do produto.
6. valor: Preço do produto.
7. */
8.
9. {
10.   "produtos": [
11.     {
12.       "id": "0",
13.       "nome": "Ipad",
14.       "valor": "5000"
15.     },
16.   ],
17. }
```

```

16.      {
17.          "id": "0",
18.          "nome": "Iphone",
19.          "valor": "8000"
20.      },
21.      {
22.          "id": "0",
23.          "nome": "Notebook",
24.          "valor": "5500"
25.      },
26.      {
27.          "id": "0",
28.          "nome": "TV",
29.          "valor": "4500"
30.      }
31.  ]
32. }
```

Para codificar e decodificar o JSON em Dart, utiliza-se a biblioteca `dart:convert`, que oferece os métodos:

- `json.decode`: converte uma string JSON em um mapa (`Map<String, dynamic>`);
- `json.encode`: converte um mapa ou lista de Dart para uma string JSON.

A seguir, temos um exemplo de uma decodificação JSON em Dart.

### Decodificando JSON

```

1. /*
2. A biblioteca dart:convert fornece funções para trabalhar
   com JSON, como:
3. json.decode: Converte uma string JSON em um mapa
   (Map<String, dynamic>) ou lista (List).
4. json.encode: Converte mapas ou listas em uma string JSON.
5. */
6. import 'dart:convert';
```

```

7.
8. void main() {
9.   // String JSON
10.  String jsonString = """
11.  {
12.    "produtos": [
13.      {"id": "0", "nome": "Ipad", "valor": "5000"},
14.      {"id": "0", "nome": "Iphone", "valor": "8000"},
15.      {"id": "0", "nome": "Notebook", "valor": "5500"},
16.      {"id": "0", "nome": "TV", "valor": "4500"}
17.    ]
18.  }
19.  """;
20.
21.  // Decodificando JSON para Map
22.  Map<String, dynamic> dados = json.decode(jsonString);
23.
24.  // Acessando a lista de produtos
25.  List produtos = dados['produtos'];
26.
27.  // Iterando sobre os produtos
28.  for (var produto in produtos) {
29.    print('Produto: ${produto['nome']}, Valor: R\$$
30.      ${produto['valor']}');
31.  }

```

A seguir, é possível visualizar uma codificação do formato JSON em Dart.

### Codificando formato JSON

```

1. void main() {
2.   // Lista de produtos em Dart
3.   List<Map<String, String>> produtos = [
4.     {"id": "0", "nome": "Ipad", "valor": "5000"},
5.     {"id": "0", "nome": "Iphone", "valor": "8000"},
6.     {"id": "0", "nome": "Notebook", "valor": "5500"},
```

```

7.      {"id": "0", "nome": "TV", "valor": "4500"}
8.    ];
9.
10.   // Estrutura de dados
11.   Map<String, dynamic> dados = {"produtos": produtos};
12.
13.   // Codificando para JSON
14.   String jsonString = json.encode(dados);
15.
16.   print(jsonString);
17. }
```

Em Dart, o tratamento dos dados JSON pode ser realizado por meio da transformação do objeto JSON em uma classe, o que facilita sua manipulação e modulação.

A seguir, é possível visualizar o código para tratar o objeto JSON e transformá-lo em classe.

#### Transformando o objeto JSON em classe

```

1. class Produto {
2.   final String id;
3.   final String nome;
4.   final String valor;
5.
6.   Produto({required this.id, required this.nome, required
  this.valor});
7.
8.   // Método para criar um objeto Produto a partir de um
  mapa
9.   factory Produto.fromJson(Map<String, dynamic> json) {
10.     return Produto(
11.       id: json['id'],
12.       nome: json['nome'],
13.       valor: json['valor'],
14.     );
}
```

```
15.    }
16.
17.    // Método para converter o objeto Produto em um mapa
18.    Map<String, dynamic> toJson() {
19.        return {
20.            'id': id,
21.            'nome': nome,
22.            'valor': valor,
23.        };
24.    }
25. }
26. void main() {
27.     String jsonString = """
28.     {
29.         "produtos": [
30.             {"id": "0", "nome": "Ipad", "valor": "5000"},
31.             {"id": "0", "nome": "Iphone", "valor": "8000"},
32.             {"id": "0", "nome": "Notebook", "valor": "5500"},
33.             {"id": "0", "nome": "TV", "valor": "4500"}
34.         ]
35.     }
36.     """;
37.
38.     // Decodificar JSON
39.     Map<String, dynamic> dados = json.decode(jsonString);
40.
41.     // Converter lista JSON para objetos Produto
42.     List<Produto> produtos = (dados['produtos'] as List)
43.         .map((produtoJson) => Produto.fromJson(produtoJson))
44.         .toList();
45.
46.     // Imprimir os produtos
47.     for (var produto in produtos) {
48.         print('Produto: ${produto.nome}, Valor: R\$ ${produto.
49.             valor}');
50.     }
51.
```

```

50.
51. // Codificar de volta para JSON
52. String novoJson = json.encode({'produtos': produtos.
53.   map((p) => p.toJson()).toList()});
54. print(novoJson);
55.

```

## XML

XML (Extensible Markup Language) é um formato estruturado e hierárquico amplamente usado para armazenar e trocar dados. Semelhante ao JSON, mas com estilo baseado em tags (como o HTML), tem como característica, além da estrutura hierárquica, a utilização de elementos aninhados para representar dados.

A seguir, é possível visualizar um exemplo de formato de dado em XML.

### Formato de dado XML

```

1. // Formato similar ao HTML
2. <produtos> <produto> <id>1</id>
3. <nome>Ipad</nome>
4. <valor>5000</valor> </produto>
5. <produto> <id>2</id>
6. <nome>Iphone</nome>
7. <valor>8000</valor> </produto>
8. </produtos>
9.

```

Podemos citar algumas vantagens em relação ao XML:

- flexível;
- extensível, permitindo criar tags personalizadas;
- legível;

- fácil de entender devido à estrutura baseada em tags;
- amplamente usado.

É possível utilizá-lo em sistemas legados, comunicação entre serviços (SOAP) e em documentos como RSS, SVG etc.

A linguagem Dart não possui uma biblioteca nativa para manipulação de XML, mas permite o uso de pacotes como o xml. Para isso, é necessário adicionar ao arquivo pubspec.yaml do projeto a seguinte dependência:

- dependencies: xml: ^6.1.0

A seguir, temos um código exemplo para manipulação de dados no formato XML.

Código exemplo para tratar dados no formato XML

```
1. /*
2. XmlDocument.parse(xmlData)
3. Analisa a string XML e retorna uma estrutura navegável.
4. findAllElements('produto')
5. Encontra todos os elementos <produto> no documento.
6. findElements('nome').single.text
7. Acessa o texto do elemento <nome> dentro de <produto>.
8.
9. */
10. import 'package:xml/xml.dart';
11.
12. void main() {
13.   // Exemplo de string XML
14.   String xmlData = """
15.   <produtos>
16.     <produto>
17.       <id>1</id>
18.       <nome>Ipad</nome>
19.       <valor>5000</valor>
20.     </produto>
```

```
21.      <produto>
22.          <id>2</id>
23.          <nome>Iphone</nome>
24.          <valor>8000</valor>
25.      </produto>
26.  </produtos>
27.  ''';
28.
29. // Parse do XML
30. final document = XmlDocument.parse(xmlData);
31.
32. // Acessar os elementos <produto>
33. final produtos = document.findAllElements('produto');
34.
35. for (var produto in produtos) {
36.     final id = produto.findElements('id').single.text;
37.     final nome = produto.findElements('nome').single.text;
38.     final valor = produto.findElements('valor').single.text;
39.
40.     print('Produto ID: $id, Nome: $nome, Valor: $valor');
41. }
42. }
43.
44.
45.
```

### Casos de uso:

- comunicação entre sistemas (incluindo sistemas legados e APIs SOAP);
- armazenamento de configurações (ex.: AndroidManifest.xml);
- formatos de documento (como RSS, SVG e outros padrões baseados em XML).

**Validação:** utiliza esquemas como XSD ou DTD para validar a estrutura dos dados.

A comparação entre o formato de dados JSON e XML é apresentada na tabela a seguir.

**Tabela 3.3 – Comparação entre o formato JSON e XML**

Aspecto	JSON	XML
Formato	Baseado em chaves e valores.	Baseado em tags hierárquicas.
Legibilidade	Mais legível e compacto.	Mais verboso, mas estruturado.
Validação	Não possui esquema padrão.	Usa XSD/DTD para validação.
Popularidade	Usado em APIs modernas (REST).	Usado em sistemas legados e SOAP.
Flexibilidade	Menos extensível que XML.	Altamente extensível.

## Requisições assíncronas

Requisições assíncronas são utilizadas para realizar tarefas que podem levar algum tempo para ser concluídas, como acessar uma API, ler um arquivo ou aguardar uma resposta do servidor. Essa abordagem evita que o programa fique “travado” enquanto a operação é processada, permitindo que outras partes do código continuem sendo executadas.

Em Dart, o código assíncrono pode ser programado de maneira simples por meio de três palavras-chave:

- `async` – indica que a função é assíncrona (e retorna um `Future`);
- `await` – pausa a execução da função até que a tarefa assíncrona seja concluída;
- `Future` – representa o resultado de operações assíncronas que podem ser completadas no futuro.

Para simplificar operações como requisições HTTP, o pacote `http` oferece alguns métodos prontos, como:

- `GET` – para recuperar dados;

- POST – para enviar informações ao servidor;
- PUT/DELETE – para atualizar ou remover recursos.

A seguir, é possível visualizar o código para realizar requisições assíncronas http.

#### Exemplo de código para requisição assíncrona

```
1. /*
2. Faz uma requisição GET para o URL especificado.
3. await é usado para aguardar a conclusão da requisição.
4. response.statusCode
5. Verifica o status da resposta HTTP.
6. 200 indica sucesso.
7. json.decode(response.body)
8. Converte o corpo da resposta (JSON) para um mapa
(Map<String, dynamic>).
9.
10.*/
11. import 'dart:convert';
12. import 'package:http/http.dart' as http;
13.
14. void main() async {
15.   String url = 'https://jsonplaceholder.typicode.com/
posts/1';
16.
17.   try {
18.     // Fazendo a requisição GET
19.     http.Response response = await http.get(Uri.
parse(url));
20.
21.     if (response.statusCode == 200) {
22.       // Decodificando o corpo da resposta JSON
23.       Map<String, dynamic> dados = json.decode(response.
body);
24.       print('Título: ${dados['title']}');
```

```
25.     } else {
26.         print('Erro na requisição. Código: ${response.
27.             statusCode}');
28.     }
29. } catch (e) {
30.     print('Erro: $e');
31. }
32.
```

A seguir, é possível visualizar a estrutura da requisição POST.

### Requisição POST

```
1. /*
2. Requisição POST
3. Envia dados no corpo da requisição (formato JSON).
4. Utiliza o método http.post.
5. Cabeçalhos
6. Content-Type: application/json: Informa que o corpo da
    requisição está no formato JSON.
7.
8.
9. */
10.
11. void fazerPost() async {
12.     String url = 'https://jsonplaceholder.typicode.com/posts';
13.
14.     Map<String, dynamic> body = {
15.         "title": "Novo Post",
16.         "body": "Conteúdo do post",
17.         "userId": 1
18.     };
19.
20.     try {
21.         http.Response response = await http.post(
```

```

22.     Uri.parse(url),
23.     headers: {"Content-Type": "application/json"},
24.     body: json.encode(body),
25. );
26.
27.     if (response.statusCode == 201) {
28.         print('Post criado com sucesso!');
29.         print('Resposta: ${response.body}');
30.     } else {
31.         print('Erro: ${response.statusCode}');
32.     }
33. } catch (e) {
34.     print('Erro na requisição: $e');
35. }
36. }
```

Além das ações já expostas, é essencial adotar um tratamento de erro. Para tanto, podemos usar blocos try-catch, garantindo que falhas – como conexões interrompidas – sejam devidamente capturadas e gerenciadas.

### Tratamento de erro

```

1. /*
2. Use um bloco try-catch para tratar erros de conexão,
   tempo limite ou outros problemas:
3. */
4.
5. void fazerRequisicaoSegura() async {
6.     String url = 'https://jsonplaceholder.typicode.com/
   posts/1';
7.
8.     try {
9.         http.Response response = await http.get(Uri.
   parse(url));
10.
11.        if (response.statusCode == 200) {
```

```

12.     print('Dados: ${response.body}');
13. } else {
14.     print('Erro: Código ${response.statusCode}');
15. }
16. } catch (e) {
17.     print('Erro de conexão: $e');
18. }
19. }
20.

```

### Simulando tarefas assíncronas

```

1. /*
2. Future.delayed:
3. Simula uma tarefa assíncrona que leva 2 segundos para ser
concluída.
4. await:
5. Aguarda a conclusão do Future antes de continuar.
6.
7. */
8. Future<String> tarefaAssincrona() async {
9.   await Future.delayed(Duration(seconds: 2));
10.  return "Tarefa concluída!";
11. }
12.
13. void main() async {
14.   print("Iniciando...");
15.   String resultado = await tarefaAssincrona();
16.   print(resultado);
17. }

```

### Exemplos práticos:

- GET: Recuperar dados;
- POST: Enviar informações para o servidor;

- Esses conceitos são amplamente utilizados em aplicações Dart e Flutter para consumir APIs e realizar tarefas demoradas sem bloquear a interface do usuário.

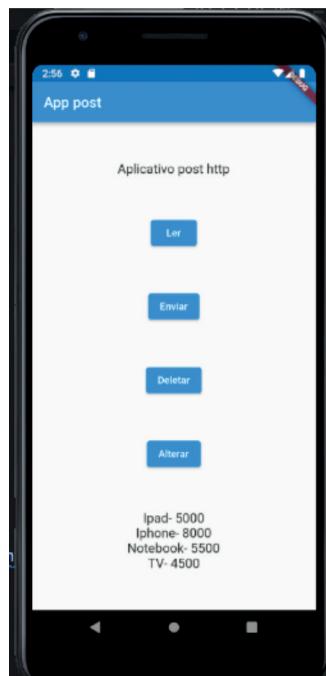
## Criando aplicativo para realizar GET, POST, PUT, DELETE

Nesta seção, desenvolveremos um aplicativo para realizar os quatro tipos de requisições HTTP: GET, POST, PUT e DELETE.

Utilizaremos o JSON Server, que simula um backend por meio de um framework Node JS.

O passo a passo do processo será apresentado nas próximas subseções.

Na Figura 3.8, é possível visualizar a tela inicial do aplicativo a ser desenvolvido.



**Figura 3.8 – Tela do aplicativo.**

**SAIBA MAIS**

O código completo da tela apresentada na Figura 3.8 pode ser acessado pelo QR Code ou pelo link a seguir.



Disponível em: [https://github.com/danielvieira95/DESM-2/tree/master/Apps-TB/app\\_aula11](https://github.com/danielvieira95/DESM-2/tree/master/Apps-TB/app_aula11).

Acesso em: 19 ago. 2025.

A seguir, temos a implementação completa para envio das requisições HTTP GET, POST, PUT e DELETE.

#### Aplicativo de demonstração HTTP

```
1. import 'package:flutter/material.dart';
2. import 'package:http/http.dart' as http;
3. import 'dart:convert'; // método para realizar a
   conversão do dado consumido da api
4.
5.
6. class _HomeState extends State<Home> {
7.   String url = "http://10.109.83.10:3000/produtos"; // url
   que será consumida para realizar os métodos
8.   // async função assíncrona
9.
10.  List dado =[];
11.  var produtos = <Produto>[]; // variável produtos está
   criando uma lista do tipo produto vazia
12.  //List<int> valor =[]; exemplo lista com numero
   inteiro
13.
14.  _getdado(){async{
15.
```

```
16.          // await é porque a função é assincrona
17.      // método para fazer requisição get http.get
18.
19.      // Uri.parse é necessário para o http.get
20.
21.      http.Response resposta = await http.get(Uri.
22.          parse(url));
23.      // var dado =json.decode(resposta.body); // json decode
24.      // faz a conversão do dado para json
25.      dado = json.decode(resposta.body);
26.      // List cria um dado do tipo lista para receber a
27.      // informação json
27.
28.      //Map<String,dynamic> dado = json.decode(resposta.body)
29.      as List;
30.
31.      // print a lista de produtos
32.
33.      for(int i =0; i<dado.length; i++)
34.      {
35.          print(dado[i]);
36.      }
37.      setState(() {
38.          // dado.map vai converter nosso json em uma lista
39.          produtos = dado.map((json) => Produto.
40.              fromJson(json)).toList();
41.      });
42.
43.      _post(){
44.          // estrutura do dado a ser publicado
45.          Map<String,dynamic>mensagem={
46.              //“id”: “6”,
```

```
45.         "nome":"PS5",
46.         "valor": "3700"
47.     };
48.     Map<String,dynamic>msg2={
49.         //"id": "6",
50.         "nome":"Mac Pro",
51.         "valor": "15000"
52.     };
53.
54.     // Método http post para realizar uma requisição
55.     // http post Uri parse, headers. body
56.     http.post(
57.         Uri.parse(url),
58.         headers: <String,String>{
59.             'Content-type':'application/json; charset=UTF-8',
60.         },
61.
62.         //body:jsonEncode(mensagem) , // transforma a
63.         mensagem para o formato json para fazer o post
64.
65.         body: jsonEncode(msg2),
66.     );
67.
68.     _deletepost(){
69.         // http delete é o metodo utilizado para deletar um
70.         // produto da api
71.         http.delete(Uri.parse('http://10.109.83.10:3000/
72.         produtos/5f01'));
73.     }
74.
75.     // put
76.     // Url, header, body
77.     // Json Encode formata a mensagem
```

```
76.      _put(){
77.        http.put(Uri.parse('http://10.109.83.10:3000/
    produtos/736a'),
78.        headers: <String, String>{
79.          'Content-type': 'application/json; charset=UTF-8',
80.        },
81.
82.        body: jsonEncode({'id':'8', 'nome':'Xbox Series
    X','valor':'2500'}),
83.      );
84.    }
85.
86.    @override
87.    Widget build(BuildContext context) {
88.      return Scaffold(
89.        appBar: AppBar(
90.          title: Text("App http métodos"),
91.        ),
92.        body: Center(
93.          child: Column(
94.            mainAxisAlignment: MainAxisAlignment.center,
95.            children: [
96.              Text("Get, Post, Put, delete http", style:
    TextStyle(fontSize: 18),),
97.              ElevatedButton(onPressed: _getdados, child:
    Text("Ler")),
98.              ElevatedButton(onPressed: _post, child:
    Text("Publicar")),
99.              ElevatedButton(onPressed: _deletepost, child:
    Text("Deletar")),
100.             ElevatedButton(onPressed: _put, child:
    Text("Alterar")),
101.
102.             Column(
```

```
103.           // exibe os dados no text
104.           // map faz o mapeamento dos dados na lista
105.           children:produtos.
106.           map((produto)=>Text("${produto.nome} - R\$ ${produto.
107.             valor}"),
108.             style: TextStyle(fontSize: 18),)).toList(),
109.           ],
110.           ),
111.           ),
112.         );
113.       }
114.     }
115.
116. class Produto{
117.   String id;
118.   String nome;
119.   String valor;
120.   Produto(this.id, this.nome, this.valor);
121.   // Função factory é a função responsável por decodificar
122.   // o dado json consumido através da api
123.   factory Produto.fromJson(Map<String,dynamic>json){
124.     return Produto(json[“id”],json[“nome”],json[“valor”]);
125.   }
126. }
127. // Classe produto_n para armazenar a lista total de
128. // produtos
129. class Produto_n{
130.   List prod =[];
131.   Produto_n(this.prod);
```

## Node.js

Node.js é um runtime JavaScript construído sobre o motor V8 do Google Chrome. Ele permite executar JavaScript no lado do servidor, possibilitando a criação de aplicações backend escaláveis e eficientes.

Suas principais características são:

- Event-driven – usa um modelo assíncrono orientado a eventos, ideal para aplicações que exigem alta escalabilidade;
- Single-threaded – um único thread gerencia múltiplas conexões por meio de operações assíncronas;
- Rápido – baseado no motor V8, executa código JavaScript com alta performance;
- Ecossistema vasto – possui o gerenciador de pacotes npm, que facilita a instalação e uso de bibliotecas.

## JSON Server

O JSON Server é uma ferramenta que permite criar rapidamente uma API REST completa e funcional a partir de um simples arquivo JSON. É ideal para prototipagem, desenvolvimento frontend e simulação de APIs. A seguir, explicaremos o processo de instalação do Node.js e JSON Server.

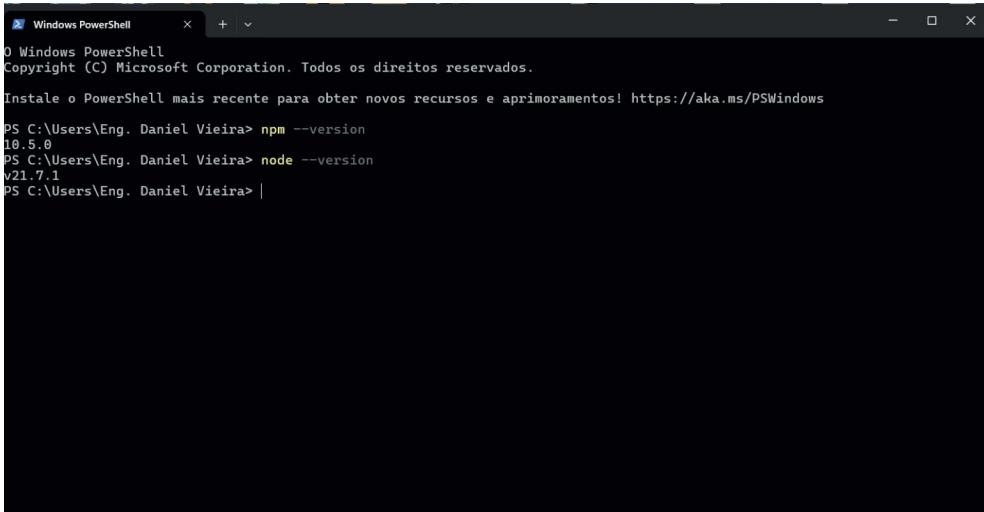
### SAIBA MAIS

O download do Node.js pode ser feito pelo QR Code ou pelo link a seguir.



Disponível em: <https://nodejs.org/en/download/current>. Acesso em: 19 ago. 2025.

Após ter realizado o download e de ter feito a instalação, entre no PowerShell e execute os comandos indicados na Figura 3.9.



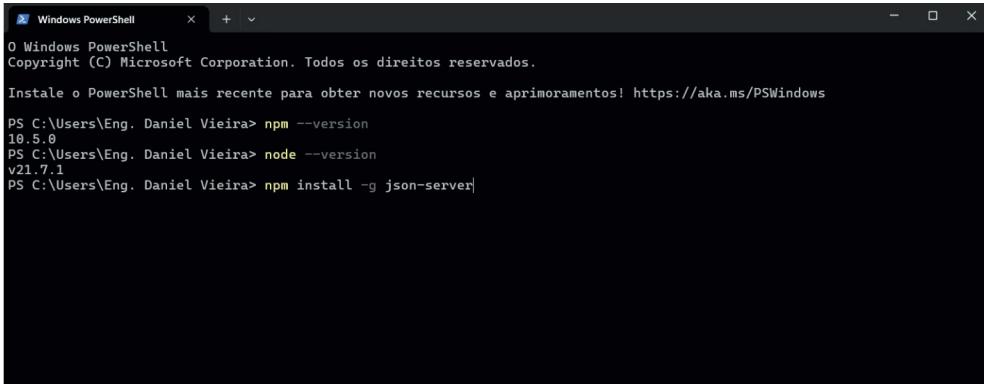
```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Instale o PowerShell mais recente para obter novos recursos e aprimoramentos! https://aka.ms/PSWindows

PS C:\Users\Eng. Daniel Vieira> npm --version
10.5.0
PS C:\Users\Eng. Daniel Vieira> node --version
v21.7.1
PS C:\Users\Eng. Daniel Vieira> |
```

**Figura 3.9** – PowerShell com os comandos para verificar a versão do Node.

Com o Node.js instalado, o próximo passo é instalar o JSON Server (ver Figura 3.10), que será responsável por emular uma API RestFull a partir do arquivo JSON.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Instale o PowerShell mais recente para obter novos recursos e aprimoramentos! https://aka.ms/PSWindows

PS C:\Users\Eng. Daniel Vieira> npm --version
10.5.0
PS C:\Users\Eng. Daniel Vieira> node --version
v21.7.1
PS C:\Users\Eng. Daniel Vieira> npm install -g json-server|
```

**Figura 3.10** – PowerShell com os comandos para instalar o JSON Server.

Após realizar a instalação do JSON Server, criaremos uma pasta chamada serve no diretório lib do projeto Flutter. Nela, geraremos um arquivo db.json contendo os dados para simular uma API.

Para iniciar o servidor, precisamos saber o ip local da máquina, o que faremos via prompt de comando. Usando o comando ipconfig, as informações sobre as configurações de rede da máquina local são mostradas (ver Figura 3.11).

```

Prompt de Comando
Estado da mídia. . . . . : mídia desconectada
Sufixo DNS específico de conexão. . . . . :

Adaptador de Rede sem Fio Conexão Local* 2:
Estado da mídia. . . . . : mídia desconectada
Sufixo DNS específico de conexão. . . . . :

Adaptador de Rede sem Fio Conexão Local* 3:
Estado da mídia. . . . . : mídia desconectada
Sufixo DNS específico de conexão. . . . . :

Adaptador de Rede sem Fio Wi-Fi:
Sufixo DNS específico de conexão. . . . . :
Endereço IPv6 . . . . . : 2804:431:cfcc:e09:8568:3017:6009:8bc3
Endereço IPv6 Temporário. . . . . : 2804:431:cfcc:e09:419a:93ff:3862:ca87
Endereço IPv6 de link local . . . . . : fe80::c67a:994:4def:7ff0%22
Endereço IPv4. . . . . : 192.168.15.10
Máscara de Sub-rede . . . . . : 255.255.255.0
Gateway Padrão. . . . . : fe80::9a7e:caff:fe50%22
                                                192.168.15.1

Adaptador Ethernet Conexão de Rede Bluetooth:
Estado da mídia. . . . . : mídia desconectada
Sufixo DNS específico de conexão. . . . . :

C:\Users\Eng. Daniel Vieira>

```

**Figura 3.11** – Prompt de comando com as informações de rede da máquina.

O endereço ip inserido para iniciar o servidor é ipv4. Para executar o servidor, é necessário entrar na pasta server com o comando cd server. Ali digitaremos o seguinte comando: json-server --watch --numerodoseuip db.json (ver Figura 3.12).

```

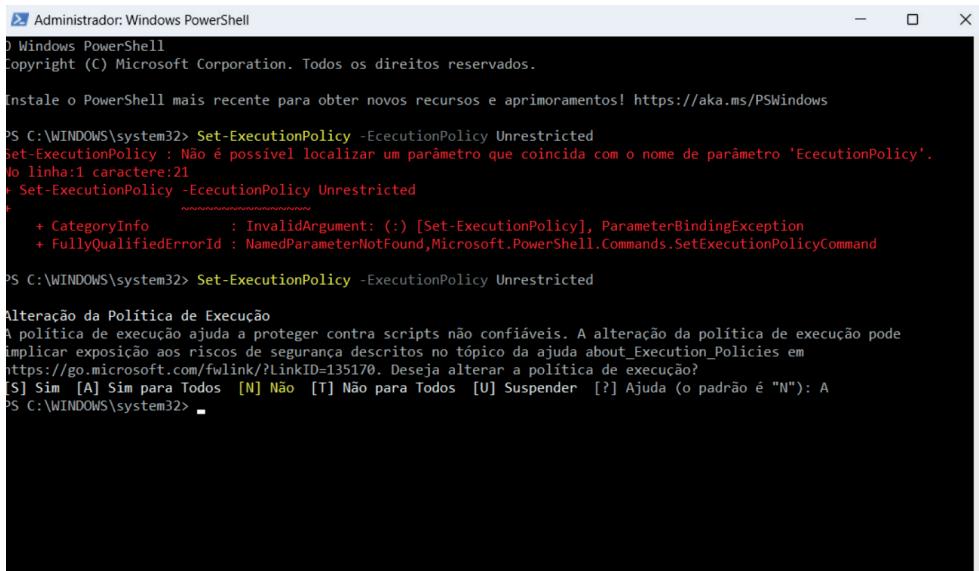
d----- 02/04/2024 16:01      server
d----- 02/04/2024 15:56      test
-a---- 02/04/2024 15:56      745 .gitignore
-a---- 02/04/2024 15:56      930 .metadata
-a---- 02/04/2024 15:56      1882 analysis.options.yaml
-a---- 02/04/2024 15:56      1899 db.json.post_prep.iml
-a---- 02/04/2024 16:06      5951 pubspec.lock
-a---- 02/04/2024 16:05      3998 pubspec.yaml
-a---- 02/04/2024 15:56      579 README.md

PS D:\SENAI\2024-1\Desenvolvimento Mobile 2\DESM-2\Appl prep\app.aula09\post_prep> cd server
PS D:\SENAI\2024-1\Desenvolvimento Mobile 2\DESM-2\Appl prep\app.aula09\post_prep> jsonserver --watch --host 192.168.15.10 db.json
jsonserver: o termo 'jsonserver' não é reconhecido como nome de cmdlet, função, arquivo de script ou programa operável. Verifique a grafia do nome ou, se um caminho tiver sido incluído, veja se o caminho está correto e tente novamente.
No linha:1 caractere:1
+ jsonserver --watch --host 192.168.15.10 db.json
+ ~~~~~
    CategoryInfo          : ObjectNotFound: (jsonserver:String) [], CommandNotFoundException
    FullyQualifiedErrorId : CommandNotFoundException

```

**Figura 3.12** – Erro ao executar o servidor.

Podemos notar que o comando executado gerou um erro, que resulta de restrições do Windows. Precisaremos, portanto, retirá-las com o uso do comando Set-ExecutionPolicy -ExecutionPolicy Unrestricted, conforme mostrado na Figura 3.13.



```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Instale o PowerShell mais recente para obter novos recursos e aprimoramentos! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> Set-ExecutionPolicy -ExecutionPolicy Unrestricted
Set-ExecutionPolicy : Não é possível localizar um parâmetro que coincida com o nome de parâmetro 'EexecutionPolicy'.
No linha:1 caractere:21
+ Set-ExecutionPolicy -EexecutionPolicy Unrestricted
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : InvalidArgument: () [Set-ExecutionPolicy], ParameterBindingException
+ FullyQualifiedErrorId : NamedParameterNotFound,Microsoft.PowerShell.Commands.SetExecutionPolicyCommand

PS C:\WINDOWS\system32> Set-ExecutionPolicy -ExecutionPolicy Unrestricted

Alteração da Política de Execução
A política de execução ajuda a proteger contra scripts não confiáveis. A alteração da política de execução pode implicar exposição aos riscos de segurança descritos no tópico da ajuda about_Execution_Policies em https://go.microsoft.com/fwlink/?linkID=135170. Deseja alterar a política de execução?
[S] Sim [A] Sim para Todos [N] Não [T] Não para Todos [U] Suspender [?] Ajuda (o padrão é "N"): A
PS C:\WINDOWS\system32>

```

**Figura 3.13 – Retirando as políticas de restrição do Windows.**

Com as políticas de restrição removidas, executa-se o servidor JSON Server novamente.



```

PROBLEMS OUTPUT TERMINAL PORTS SERIAL MONITOR DEBUG CONSOLE

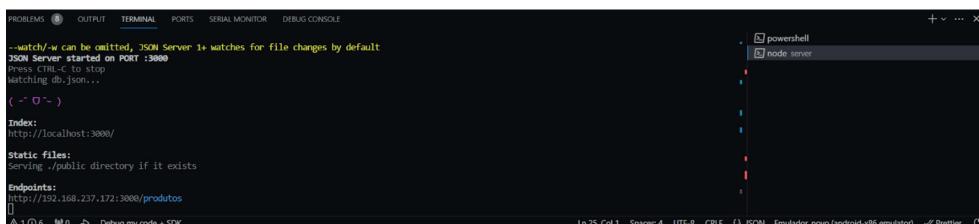
options:
  -p, --port <port> Port (default: 3000)
  -h, --host <host> Host (default: localhost)
  -s, --static <dir> Static directory (multiple allowed)
  -h[elp]
  -v[ersion]
  -watch/-w can be omitted, JSON Server 1+ watches for file changes by default
  JSON Server started on PORT :3000
Press CTRL-C to stop
Watching db.json...
( ⌂ ⌂ ⌂ )

Index:
1 △ 1 ○ 6 ⌂ 0 ⌂ Debug my code + SDK

```

**Figura 3.14 – Executando o servidor JSON Server.**

Na Figura 3.15, é possível visualizar o servidor JSON Server executando e o end point com o endereço da API criado.



```

PROBLEMS OUTPUT TERMINAL PORTS SERIAL MONITOR DEBUG CONSOLE

-watch/-w can be omitted, JSON Server 1+ watches for file changes by default
JSON Server started on PORT :3000
Press CTRL-C to stop
Watching db.json...
( ⌂ ⌂ ⌂ )

Index:
http://localhost:3000/
Static files:
Serving ./public directory if it exists
Endpoints:
[ http://192.168.237.172:3000/produtos
[

△ 1 ○ 6 ⌂ 0 ⌂ Debug my code + SDK

```

**Figura 3.15 – JSON Server e o end point criado.**

Esse end point será utilizado para fazer as requisições HTTP GET, POST, PUT e DELETE. Outra etapa precisa ser realizada para que as requisições sejam feitas: adicionar o pacote HTTP no arquivo pubspec.yaml do projeto e executar o flutter pub get para atualizar todos os pacotes e dependências do projeto. Vejamos na Figura 3.16, o pacote http adicionado no pubspec.yaml.

```
# The following adds the Cupertino Icons font to your application.  
# Use with the CupertinoIcons class for iOS style icons.  
cupertino_icons: ^1.0.2  
http: ^0.13.3
```

**Figura 3.16** – Pacote http adicionado ao arquivo pubspec.yaml.

Vantagens do JSON Server:

- rápida configuração – ideal para criar APIs temporárias;
- flexível – fácil de personalizar e expandir;
- simulação realista – imita operações CRUD completas com dados JSON.

Limitações:

- não recomendado para produção – é ideal para desenvolvimento e prototipagem, mas não oferece segurança ou performance para produção;
- dados não persistentes – alterações são gravadas no arquivo JSON e podem ser sobreescritas facilmente.

O JSON Server é uma ferramenta poderosa e prática para simular APIs REST, especialmente em aplicações nas quais o foco inicial está no desenvolvimento frontend. É uma ótima escolha para estudantes e profissionais que desejam prototipar rapidamente sem se preocupar com um backend complexo.

## Exemplo de aplicativo completo com requisições GET, POST, PUT e DELETE

Desenvolveremos um aplicativo no qual o usuário digitará um TextField cujos dados serão postados na API quando confirmados. Esses dados aparecerão em uma tela separada.

O aplicativo terá cinco telas:

- duas telas de login;
- uma tela para digitar as informações;
- duas telas que mostrarão esses dados após o envio.

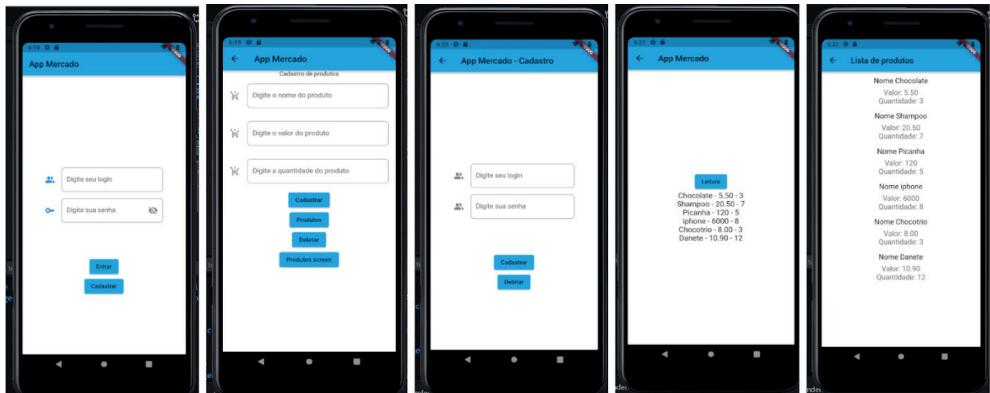
Para que tudo funcione, precisaremos criar uma API local usando Node.js, que permitirá que o aplicativo realize todas as operações necessárias com os dados:

- enviar novas informações (POST);
- recuperar dados para exibição (GET);
- atualizar dados (PUT);
- remover informações (DELETE).

Em suma, o aplicativo será projetado para:

- coletar dados por meio de um TextField;
- enviar esses dados para a API após confirmação;
- exibir os dados enviados em uma tela separada;
- trabalhar com todas as operações básicas de dados (GET, POST, PUT e DELETE).

As telas a serem criadas são apresentadas na figura a seguir.



**Figura 3.17 – Telas do aplicativo.**

O arquivo db.json, mostrado na Figura 3.18, contém a estrutura de dados (produto e usuários) utilizada para simular a API.

```

1  {
2    "produtos": [
3      {
4        "id": "0",
5        "nome": "Chocolate",
6        "valor": "5.50",
7        "qtde": "3"
8      },
9      {
10        "id": "1",
11        "nome": "Shampoo",
12        "valor": "20.50",
13        "qtde": "7"
14      },
15      {
16        "id": "2",
17        "nome": "Picanha",
18        "valor": "120",
19        "qtde": "5"
20      },
21      {
22        "id": "b352",
23        "nome": "maca",
24        "valor": "2.50",
25        "qtde": "5"
26      },
27    ],
28    {
29      "id": "af44",
30      "nome": "iphone",
31      "valor": "6000",
32      "qtde": "8"
33    },
34    {
35      "id": "0e5e",
36      "nome": "Chocotrio",
37      "valor": "8.00",
38      "qtde": "3"
39    },
40    {
41      "id": "04a0",
42      "nome": "Danete",
43      "valor": "10.90",
44      "qtde": "12"
45    }
46  ],
47  {
48    "usuarios": [
49      {
50        "login": "admin",
51        "senha": "1234",
52        "id": "fb45"
53      },
54      {
55        "login": "filipe",
56        "senha": "1004",
57        "id": "4e0b"
58      },
59      {
60        "id": "4a9a",
61        "login": "daniel",
62        "senha": "100455"
63      }
64    ]
65  }

```

**Figura 3.18 – Arquivo db.json.**

Para executar o servidor, precisaremos entrar na pasta server pelo terminal e digitar: json-server --watch --numerodoseuip db.json.

Precisaremos, agora, criar os arquivos necessários para o funcionamento do aplicativo. São eles: cadastrocliente.dart, cadastraproduto.dart, login.dart e prodscreen.dart. Cada um deles tem seu próprio código de tela, como veremos a seguir.

### Código da tela de cadastro de cliente

```
1. import 'package:flutter/material.dart'; // pacotes do widget
2. import 'package:http/http.dart' as http; // pacote http
   que permite fazer as requisições http
3. import 'dart:convert'; // pacote para converter json
4. class Cadastrousuario extends StatefulWidget {
5.
6.
7.   @override
8.   State<Cadastrousuario> createState() =>
9.   _CadastrousuarioState();
10.
11. class _CadastrousuarioState extends
12.   State<Cadastrousuario> {
13.   TextEditingController user_n = TextEditingController();
14.   TextEditingController senha_n = TextEditingController();
15.   bool exibir = false; // variável para exibir ou não a
   senha
16.   _cadastrausuario()async{
17.     bool encuser = false; // variável para setar quando o
   usuário for encontrado
18.     String url = "http://10.109.83.10:3000/usuarios";
19.     Map<String,dynamic> mensagem = {
20.       "id": user_n.text,
21.       "login":user_n.text,
22.       "senha": senha_n.text,
23.     };
24.     // http post método para realizar um post na api
25.     http.post(Uri.parse(url),
26.     headers: <String,String> {
27.       'Content-type': 'application/json; charset=UTF-8',
28.     },
29.     body: jsonEncode(mensagem) );
```

```
30.  
31.      // url da api com os usuarios  
32.      print("Usuario cadastrado");  
33.      user_n.text =“ ”; // limpa os campos  
34.      senha_n.text =“ ”; // limpa os campos  
35.  
36.  
37.  
38.    }  
39.  
40.  
41.  @override  
42.  Widget build(BuildContext context) {  
43.    // Scaffold faz parte do layout do app  
44.    return Scaffold(  
45.      appBar: AppBar(title: Text("Cadastro de usuário"),),  
46.      // body corpo do scaffold  
47.      //backgroundColor: Colors.white,  
48.      body: Center(  
49.        child: Column(  
50.          mainAxisAlignment: MainAxisAlignment.center,  
51.          children: [  
52.            // elemento decorativo  
53.            SizedBox(  
54.              height:300 ,  
55.              width: 300,  
56.              child: Column(  
57.                children: [  
58.                  // textformfield para permitir que o  
usuário digite seu nome  
59.                  Padding(  
60.                    padding: const EdgeInsets.all(8.0),  
61.                    child: TextFormField(  
62.                      keyboardType: TextInputType.name,  
63.                      decoration:  
InputDecoration(border: OutlineInputBorder(borderRadius:  
BorderRadius.circular(8)),
```

```
64.                      icon: Icon(Icons.people_alt_  
65.                         outlined,color: Colors.red,),  
66.                         hintText: "Digite seu nome"),//  
67. Mensagem no text form field  
68.  
69. ),  
70. ),  
71. Padding(  
72.           padding: const EdgeInsets.all(8.0),  
73.           child: TextFormField(  
74.             keyboardType: TextInputType.name,  
75.             decoration:  
76.               InputDecoration(border: OutlineInputBorder(borderRadius:  
77.                 BorderRadius.circular(8)),  
78.               icon: Icon(Icons.key_  
79.                 outlined,),iconColor: Colors.red,  
80.               // suffix icon responsável por criar  
81.               um icone no TextFormField  
82.               // Icon button icone para o botão  
83.               // exibir ? operador ternário  
84.               // if( exibir == true icons.  
85.               visibility else visibility off)  
86.               suffixIcon: IconButton(icon:  
87.                 Icon(exibir ? Icons.visibility :Icons.visibility_  
88.                   off),onPressed: (){  
89.                   setState(() {  
90.                     exibir = !exibir; // inverte o  
91.                     estado da variavel exibir  
92.                   });  
93.                 }),  
94.                 hintText: "Digite sua senha"),  
95.                 obscureText: exibir,  
96.                 obscuringCharacter: "*",// Mensagem  
97.               no text form field
```

```
89.                     // ícone do TextFormField
90.
91.                     controller: senha_n,
92.
93.                     ),
94.                     ),
95.                     ],
96.                     ),
97.
98.                     ),
99.                     ElevatedButton(onPressed: _cadastrausuario,
100.                         child: Text("Entrar")),
101.                         ElevatedButton(onPressed: (){}, child:
102.                             Text("Cadastrar")),
103.                             ],
104.                             );
105. }
106. }
107.
108. // Classe para realizar o mapeamento json através da api
109. class Users{
110.     String id;
111.     String login;
112.     String senha;
113.     Users(this.id,this.login, this.senha);
114.     // Função para mapear nossos dados após a leitura da
115.     // api e
116.     // retorna id, login e senha
117.     factory Users.fromJson(Map<String,dynamic>json){
118.         return Users(json["id"],json["login"],json["senha"]);
119.     }
120.
```

## Tela de cadastro do produto

```
1. import 'package:aula13_app_correcaoformativa/prodscreen.  
dart';  
2. import 'package:aula13_app_correcaoformativa/produto.  
dart';  
3. import 'package:flutter/material.dart';  
4. import 'package:http/http.dart' as http;  
5. import 'dart:convert';  
6.  
7. class Cadastroproduto extends StatefulWidget {  
8.   const Cadastroproduto({super.key});  
9.  
10.  @override  
11.  State<Cadastroproduto> createState() =>  
    _CadastroprodutoState();  
12. }  
13.  
14. class _CadastroprodutoState extends State<Cadastroproduto>  
{  
15.   TextEditingController nomeprod =  
    TextEditingController(); // nome do produto  
16.   TextEditingController valorprod =  
    TextEditingController();  
17.   TextEditingController qtde = TextEditingController();  
18.   // função para cadastro de produto  
19.   _cadastrarprod()async{  
20.     String url = "http://10.109.83.10:3000/produtos"; // url  
    produtos  
21.     Map<String,dynamic> prod={  
22.       "id": nomeprod.text,  
23.       "nome": nomeprod.text,  
24.       "valor": valorprod.text,  
25.       "qtde": qtde.text  
26.     };  
27.
```

```
28.     http.post(Uri.parse(url),
29.     headers: <String, String>{
30.       'Content-type': 'application/json; charset=UTF-8',
31.     },
32.     body: jsonEncode(prod));
33.
34.     nomeprod.text=" ";
35.     valorprod.text=" ";
36.     qtde.text=" ";
37.   }
38.   _deleteprod(){
39.     http.delete(Uri.parse("http://10.109.83.10:3000/
        produtos/${nomeprod.text}")); // metodo para deletar o
        produto
40.     nomeprod.text =" "; // limpa o campo com nome do
        produto
41.   }
42.   @override
43.   Widget build(BuildContext context) {
44.     return Scaffold(
45.       appBar: AppBar(
46.         title: Text("App Mercado"),
47.       ),
48.       body: Column(
49.         mainAxisAlignment: MainAxisAlignment.center,
50.         children: [
51.           TextField(
52.             keyboardType: TextInputType.name,
53.             decoration: InputDecoration(border:
                OutlineInputBorder(borderRadius: BorderRadius.circular(8)),
54.               hintText: "Digite o nome do produto",),
55.               controller: nomeprod,
56.
57.           ),
58.           TextField(
59.             keyboardType: TextInputType.name,
```

```

60.           decoration: InputDecoration(border:
    OutlineInputBorder(borderRadius: BorderRadius.circular(8)),
61.           hintText: "Digite o valor do produto",),
62.           controller: valorprod,
63.       ),
64.           TextField(
65.               keyboardType: TextInputType.name,
66.               decoration: InputDecoration(border:
    OutlineInputBorder(borderRadius: BorderRadius.circular(8)),
67.                   hintText: "Digite a quantidade do produto",),
68.                   controller: qtde,
69.
70.           ),
71.           ElevatedButton(onPressed: _cadastrarprod,
    child: Text("Cadastrar")),
72.           ElevatedButton(onPressed: _deleteprod, child:
    Text("Deletar")),
73.           ElevatedButton(onPressed: (){
74.               Navigator.push(context,
    MaterialPageRoute(builder: (context)=>Produto())));
75.           }, child: Text("Produto")),
76.           ElevatedButton(onPressed: (){Navigator.
    push(context, MaterialPageRoute(builder:
    (context)=>Produtoscreen())));
77.           }, child: Text("Prod Screen"))
78.       ],
79.   ),
80. );
81. }
82. }
```

Código da tela prodscreen (exibição dos produtos)

```

1. import 'package:flutter/material.dart';
2. import 'package:http/http.dart' as http;
3. import 'dart:convert';
```

```
4. // Classe produto tela
5. class Produtoscreen extends StatefulWidget {
6.   const Produtoscreen({super.key});
7.
8.   @override
9.   State<Produtoscreen> createState() =>
10.    _ProdutoscreenState();
11.
12. class _ProdutoscreenState extends State<Produtoscreen> {
13.   // função que permite os dados serem exibidos de forma
14.   // automatica na tela
15.   void initState(){
16.     super.initState();
17.     exibeprod();
18.   }
19.
20.   List dado = [];
21.   Future<void> exibeprod()async{
22.     String url = "http://10.109.83.10:3000/produtos";
23.     http.Response resposta = await http.get(Uri.
24.       parse(url));
25.     // lista de produtos prod itens
26.     if(resposta.statusCode ==200){
27.       setState(() {
28.         dado = json.decode(resposta.body) as List<dynamic>; // 
29.         cria a variavel dado como lista para receber o json
30.         print(dado);// transforma os dados como uma lista para
31.         // poder exibir
32.       });
33.     }
34.   }
```

```
35.     @override
36.     Widget build(BuildContext context) {
37.       return Scaffold(
38.         appBar: AppBar(
39.           title: Text("App Mercado produtos"),
40.         ),
41.         body: Center(
42.           child: ListView.builder(
43.             itemCount: dado.length, // conta o tamanho
44.               da lista de dados
45.               itemBuilder: (context,index){
46.                 final item = dado[index]; // variavel item
47.                   que irá armazenar os elementos da lista
48.                   return ListTile(
49.                     title: Text("Nome:
50. ${item["nome"]}",style: TextStyle(fontSize: 16),textAlign:
51. TextAlign.center),
52.                     subtitle: Column(
53.                       children: [
54.                         Text("Valor: ${item["valor"]}")
55.                         ,style: TextStyle(fontSize: 16),),
56.                         Text("Qtde: ${item["qtde"]}",style:
57. TextStyle(fontSize: 16)),
58.                         ],
59.                     ),
60.                   );
61.                 },
62.               //ElevatedButton(onPressed: _exibeprod, child:
63.               Text("Exibir")),
64.             );
```

```
63.           ),
64.       );
65.
66.   }
67. }
68. // classe produto itens
69. class ProdItens{
70.     String id;
71.     String nome;
72.     String valor;
73.     String qtde;
74.     ProdItens(this.id, this.nome, this.valor, this.qtde); //  
    construtor da classe produto
75.     factory ProdItens.fromJson(Map<String,dynamic>json){
76.         return
77.             ProdItens(json[“id”],json[“nome”],json[“valor”],json[“qtde”]);
78.     }
79. }
```

### Código da tela de login

```
1. import ‘package:aula13_app_correcaoformativa/  
    cadastrocliente.dart’;
2. import ‘package:aula13_app_correcaoformativa/  
    cadastropduto.dart’;
3. import ‘package:aula13_app_correcaoformativa/produto.  
    dart’;
4. import ‘package:flutter/material.dart’; // pacotes do  
    widget
5. import ‘package:http/http.dart’ as http; // pacote http  
    que permite fazer as requisições http
6. import ‘dart:convert’; // pacote para converter json
7. class Login extends StatefulWidget {
8.     const Login({super.key});
9.
```

```
10.    @override
11.    State<Login> createState() => _LoginState();
12. }
13.
14. class _LoginState extends State<Login> {
15.   TextEditingController user = TextEditingController();
16.   TextEditingController senha = TextEditingController();
17.   bool exibir = false; // variavel para exibir ou nao a
   senha
18.   _verificaLogin()async{
19.     bool encuser = false; // variavel para setar quando o
   usuario for encontrado
20.     String url = "http://10.109.83.10:3000/usuarios"; //
   url da api com os usuarios
21.     http.Response resposta = await http.get(Uri.
   parse(url)); // resposta irá guardar o retorno da api
22.     List clientes =<Users>[]; // cria uma lista chamada
   clientes que recebe minha classe users.
23.     print(resposta.statusCode);
24.     var dados = json.decode(resposta.body) as List; //
   armazena em dados a minha resposta da api
25.     clientes = json.decode(resposta.body) as List; //
   Lista clientes armazena os dados da api
26.     print("${dados[0][“login”]} ${dados[0][“senha”]}");
27.     for(int i=0; i<clientes.length;i++){
28.       if(user.text == clientes[i][“login”] && senha.text
   == clientes[i][“senha”]){
29.         encuser = true;
30.         break;
31.       }
32.     }
33.     if(encuser ==true){
34.       print("Usuario ${user.text} encontrado");
35.       encuser = false;
36.       Navigator.push(context, MaterialPageRoute(builder:
   (context)=>Cadastroproduto())));

```

```
37.         user.text =“ ”;
38.         senha.text =“ ”;
39.     }
40.     else{
41.         print(“Usuario ${user.text} nao encontrado”);
42.     }
43. }
44.
45.
46. @override
47. Widget build(BuildContext context) {
48.     // Scaffold faz parte do layout do app
49.     return Scaffold(
50.         // body corpo do scaffold
51.         //backgroundColor: Colors.white,
52.         body: Center(
53.             child: Column(
54.                 mainAxisAlignment: MainAxisAlignment.center,
55.                 children: [
56.                     // elemento decorativo
57.                     SizedBox(
58.                         height:300 ,
59.                         width: 300,
60.                         child: Column(
61.                             children: [
62.                                 // textformfield para permitir que o
63.                                 // usuário digite seu nome
64.                                 Padding(
65.                                     padding: const EdgeInsets.all(8.0),
66.                                     child: TextFormField(
67.                                         keyboardType: TextInputType.name,
68.                                         decoration: InputDecoration(border:
OutlineInputBorder(borderRadius: BorderRadius.circular(8)),
icon: Icon(Icons.people_alt_outlined,color: Colors.red,),
```

```

69.                               hintText: "Digite seu nome"),//  

    Mensagem no text form field  

70.                               // icone do textformfield  

71.                               controller: user,  

72.  

73.                               ),  

74.                               ),  

75.                               Padding(  

76.                               padding: const EdgeInsets.all(8.0),  

77.                               child: TextFormField(  

78.                               keyboardType: TextInputType.name,  

79.                               decoration:  

    InputDecoration(border: OutlineInputBorder(borderRadius:  

    BorderRadius.circular(8)),  

80.                               icon: Icon(Icons.key_  

    outlined),iconColor: Colors.red,  

81.                               // suffix icon responsável por criar  

    um icone no textformfield  

82.                               // Icon button icone para o botão  

83.                               // exibir ? operador ternário  

84.                               // if( exibir == true icons.  

    visibility else visibility off)  

85.                               suffixIcon: IconButton(icon:  

    Icon(exibir ? Icons.visibility_off :Icons.  

    visibility),onPressed: (){  

86.                               setState(() {  

87.                               exibir = !exibir; // inverte o  

    estado da variavel exibir  

88.                               });  

89.                               },  

90.                               hintText: "Digite sua senha"),  

91.                               obscureText: exibir,  

92.                               obscuringCharacter: "*",// Mensagem  

    no text form field

```

```
93.                                     // ícone do TextFormField
94.
95.                                     controller: senha,
96.
97.                                     ),
98.                                     ),
99.                                     ],
100.                                    ),
101.
102.                                    ),
103.                                     ElevatedButton(onPressed: _verificaLogin,
104.                                         child: Text("Entrar"),),
105.                                         ElevatedButton(onPressed: (){
106.                                             // chama a tela de cadastro
107.                                             Navigator.push(context,
108.                                                 MaterialPageRoute(builder: (context)=>Cadastrousuario())));
109.                                             ],),
110.                                         ),
111.                                         );
112.                                     }
113.                                 }
114.
115. // Classe para realizar o mapeamento json através da api
116. class Users{
117.     String id;
118.     String login;
119.     String senha;
120.     Users(this.id,this.login, this.senha);
121.     // Função para mapear nossos dados após a leitura da
122.     // api e
123.     factory Users.fromJson(Map<String,dynamic>json){
```

```
124.     return Users(json["id"],json["login"],json["senha"]);  
125.   }  
126. }  
127.
```

A seguir, é possível visualizar o main.dart, código principal do aplicativo.

#### Código main.dart

```
1. import 'package:aula13_app_correcaoformativa/login.dart';  
2. import 'package:flutter/material.dart';  
3.  
4. void main() {  
5.   runApp(MaterialApp(  
6.  
7.   // tema do app  
8.   theme: ThemeData(  
9.   scaffoldBackgroundColor: Colors.white,  
10.  primaryColor: Colors.white, // cor primária  
11.  primarySwatch: Colors.red,),// cor de botões  
12.  home: Login(),  
13.  ));  
14. }  
15.
```

#### SAIBA MAIS

O código completo desse aplicativo pode ser encontrado no QR Code ou no link a seguir.

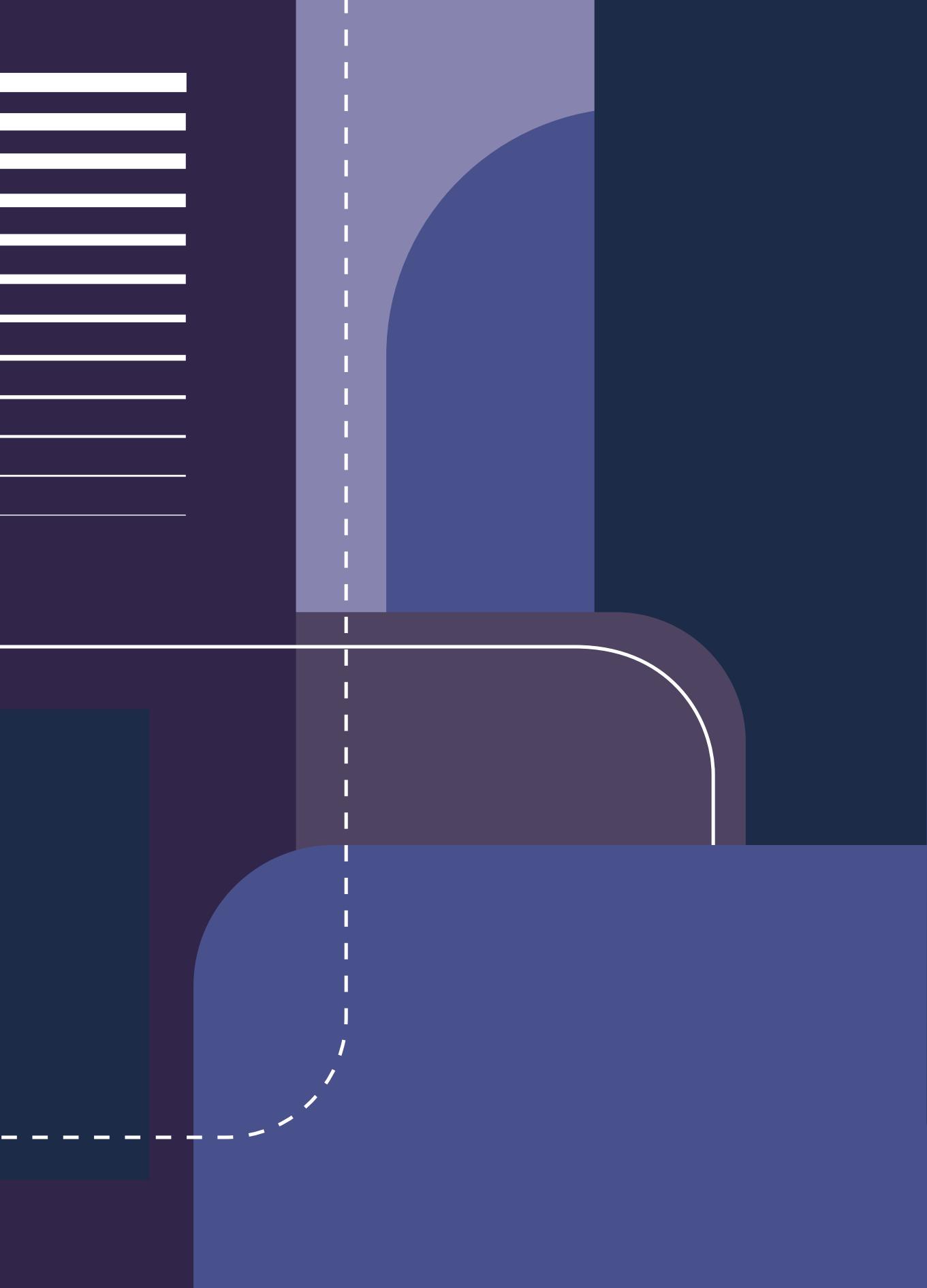


Disponível em: <https://api.flutter.dev/>. Acesso em: 19 ago. 2025.

## Conclusão

Neste capítulo, exploramos os conceitos fundamentais de APIs RESTful e os principais métodos de requisição HTTP: GET, POST, PUT e DELETE. Aprendemos a desenvolver uma API simples utilizando Node.js em conjunto com o JSON Server, permitindo o recebimento de dados enviados pelo aplicativo – como o cadastro de usuários e produtos, além da inserção de informações de login –, estabelecendo assim a integração entre o aplicativo e o backend.

No próximo capítulo, abordaremos a persistência de dados e os diferentes tipos de bancos de dados. Também aprenderemos a utilizar o Firebase como plataforma para armazenar essas informações de forma eficiente e segura.





## CAPÍTULO 4

# PERSISTÊNCIA DE DADOS EM APLICATIVOS MÓVEIS

### Introdução

No desenvolvimento de aplicativos móveis, a persistência de dados é tão essencial quanto a integração com serviços externos. Afinal, enquanto APIs permitem que aplicativos se conectem a serviços e servidores em tempo real, a persistência de dados garante que informações importantes sejam armazenadas localmente e estejam disponíveis mesmo quando o dispositivo estiver off-line.

Imagine um aplicativo que exibe informações de um catálogo de produtos, como uma loja virtual. Quando o usuário está conectado à internet, o aplicativo carrega a lista mais atualizada de produtos, exibindo informações como nome, preço e imagem. Agora, suponha que o usuário perca a conexão

com a internet enquanto navega no aplicativo. Sem a persistência de dados, a lista de produtos simplesmente desapareceria, e o aplicativo exibiria uma mensagem de erro, tornando-se inutilizável.

Para evitar esse problema, a persistência de dados entra em cena. O aplicativo pode armazenar localmente as informações obtidas anteriormente, como configurações, cache de dados das APIs ou preferências do usuário, utilizando tecnologias como SQLite, Hive ou SharedPreferences. Assim, mesmo sem conexão, o aplicativo acessa os dados armazenados localmente, garantindo que a lista de produtos continue sendo exibida. Por exemplo, se o usuário já navegou anteriormente e os dados foram armazenados, ele ainda poderá visualizar o catálogo, adicionar produtos ao carrinho ou consultar os detalhes de um item. Além disso, ao detectar o retorno da conexão, o aplicativo pode sincronizar automaticamente os dados locais com a versão mais recente, garantindo que o usuário tenha acesso às informações atualizadas. Esse fluxo não só melhora a confiabilidade do aplicativo, mas também proporciona uma experiência mais fluida e satisfatória para o usuário.

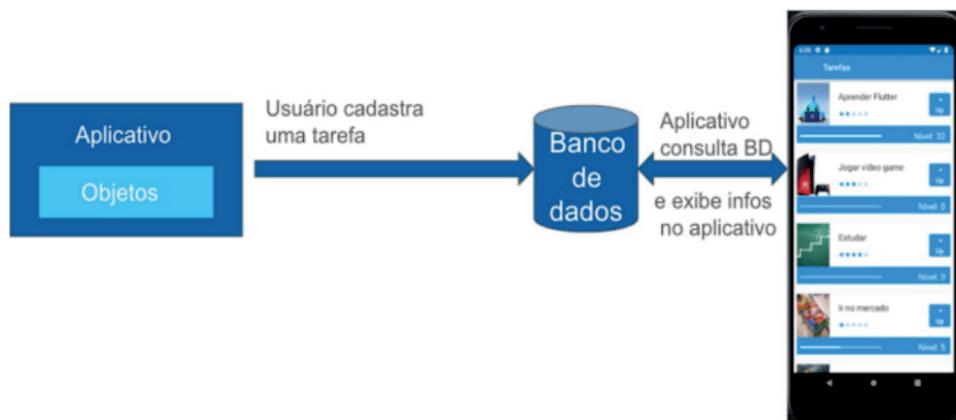
Neste capítulo, os recursos que serão explorados para armazenar os dados são: SharedPreferences, SqFlite, Hive, Drift e Firebase. A proposta é que, ao final, você seja capaz de desenvolver aplicativos móveis que tornam os dados persistentes de forma segura e eficiente.

## Armazenamento de dados

Quando falamos em persistência de dados no desenvolvimento de aplicativos móveis, estamos nos referindo à capacidade do aplicativo de armazenar informações de forma que possam ser recuperadas e utilizadas mesmo após ser fechado. Esse armazenamento é fundamental para garantir uma experiência contínua e confiável ao usuário, independentemente das condições de uso, como perda de conexão com a internet.

Imagine, por exemplo, um aplicativo de lista de tarefas. O usuário insere informações importantes, como atividades e prazos, ao longo do dia. Sem persistência de dados, todas essas informações seriam perdidas assim que o aplicativo fosse fechado ou reiniciado, resultando em uma experiência frustrante.

Na figura a seguir, vemos como a persistência de dados resolve esse problema. O fluxo começa quando o usuário cadastra uma nova tarefa no aplicativo. Assim que a tarefa é criada, o aplicativo armazena essas informações no banco de dados, garantindo que elas fiquem salvas de forma segura e duradoura.



**Figura 4.1** – Fluxo de funcionamento do aplicativo com persistência de dados.

Posteriormente, o usuário fecha o aplicativo e, mais tarde, o reabre. Ao ser iniciado novamente, o aplicativo realiza uma consulta ao banco de dados para recuperar as tarefas cadastradas anteriormente. Essas tarefas são, então, exibidas na tela do aplicativo, permitindo que o usuário continue de onde parou, sem perder nenhuma informação.

Esse processo não só ilustra a importância da persistência de dados, mas também destaca como ela contribui para uma experiência de uso contínua e confiável.

Na tabela a seguir, é possível visualizar os principais recursos utilizados em conjunto com o framework Flutter para armazenar dados.

**Tabela 4.1 – Principais bibliotecas utilizadas para salvar dados com o framework Flutter**

Biblioteca	Descrição	Relacional/ Não relacional	Linguagem base	Vantagem principal	Desvantagem principal	Aplicações
SharedPreferences	Armazenamento simples de pares chave-valor.	Não relacional.	Dart	Simplicidade.	Limitação de dados.	Dados de usuários como login e senha.
Sqflite	Interface SQLite direta.	Relacional.	SQL	Consultas avançadas.	Complexidade.	Características de um produto como cor, quantidade, preço.
Hive	Banco de dados NoSQL rápido.	Não relacional.	Dart	Performance.	Escalabilidade.	Lista de contatos, lista de tarefas.
Drift	ORM para SQLite com integração Dart.	Relacional.	Dart/SQL	Integração com Dart.	Curva de aprendizado.	Gerenciamento de estoque ou finanças que precisa realizar consultas SQL complexas com joins, agregações e subconsulta.
Firebase	Banco de dados Cloud.	Não relacional.	JSON (estrutura de dados)	Fácil integração com aplicativos móveis, sincronização em tempo real e escalabilidade automática.	Não é ideal para dados altamente relacionados e complexos devido à falta de suporte a joins.	Aplicativos que precisam de dados em tempo real, sincronização rápida entre dispositivos ou integração com serviços na nuvem, como autenticação e armazenamento.

## SharedPreferences

**SharedPreferences** é uma biblioteca do Flutter amplamente utilizada para armazenar dados simples e persistentes diretamente no dispositivo do usuário. Por sua simplicidade e abordagem intuitiva, essa ferramenta é, muitas vezes, o primeiro contato dos desenvolvedores com a persistência de dados no Flutter.

Os dados são armazenados em um formato de **chave-valor**, que funciona de maneira semelhante aos **Maps no Dart**, ao **LocalStorage no JavaScript** ou ao **UserDefaults no iOS**. Essa estrutura organiza informações de forma clara e acessível, permitindo que cada dado seja identificado por uma chave única.

- **Maps no Dart:** os **Maps** são coleções associativas que armazenam pares de chave e valor. Por exemplo, um Map pode ser usado para associar o nome de um usuário (chave) ao seu endereço de e-mail (valor).
- **LocalStorage no JavaScript:** o **LocalStorage** permite que dados sejam salvos localmente no navegador do usuário, também no formato chave-valor, tornando as informações acessíveis mesmo após recarregar ou fechar a página.
- **UserDefaults no iOS:** o **UserDefaults** é uma API utilizada para armazenar pequenas quantidades de dados, como preferências ou configurações, no dispositivo iOS. Ele utiliza a mesma lógica de chave-valor para recuperar dados de forma rápida.

O **SharedPreferences** é ideal para salvar pequenas quantidades de dados que precisam ser acessadas rapidamente pelo aplicativo, como preferências do usuário, configurações, tokens de autenticação ou estados simples da interface. Por não exigir uma estrutura complexa, como um banco de dados, ele é uma solução prática e eficiente para armazenamentos leves e temporários.

## Vantagens do SharedPreferences

- **Simplicidade:** o SharedPreferences é fácil de usar e não requer configuração complicada, sendo ideal para desenvolvedores iniciantes.
- **Desempenho:** oferece acesso rápido aos dados, já que são armazenados localmente no dispositivo.
- **Persistência:** os dados permanecem disponíveis mesmo após o aplicativo ser fechado e reaberto, garantindo uma boa experiência do usuário.
- **Pequeno armazenamento:** é ideal para salvar configurações, preferências do usuário, tokens de autenticação e pequenos pedaços de informação, mas com a limitação de não suportar grandes volumes de dados.

## Desvantagens do SharedPreferences

- **Limitação de dados:** não é adequado para armazenar grandes volumes de dados ou informações complexas, sendo ideal apenas para dados simples e de pequeno porte.
- **Falta de segurança:** os dados armazenados no SharedPreferences não são criptografados por padrão, tornando-os vulneráveis ao acesso indevido, especialmente para informações sensíveis, como senhas ou dados pessoais.
- **Sem suporte para dados estruturados:** apenas permite armazenar tipos de dados primitivos (strings, números, booleanos) e listas simples. Não suporta armazenamento de objetos complexos ou relações entre dados.

## Quando usar o SharedPreferences?

O SharedPreferences é ideal para situações em que você precisa armazenar informações básicas que devem persistir entre sessões do aplicativo. Ele é amplamente utilizado em cenários como:

- configurações do usuário, como temas do aplicativo (modo claro/escuro), preferências de idioma ou outras opções personalizáveis;
- tokens de autenticação ou informações simples de sessão que não são altamente sensíveis;
- estados simples do aplicativo, como a última aba acessada, o progresso em um tutorial ou opções escolhidas pelo usuário.

O SharedPreferences é uma ferramenta prática e eficiente para gerenciar dados simples e persistentes no Flutter, oferecendo uma solução rápida e fácil de implementar para necessidades comuns de armazenamento local. No entanto, é importante lembrar que ele **não é a opção mais segura para dados sensíveis**, pois os dados são armazenados sem criptografia. Para cenários que exigem um nível maior de segurança, existe o flutter\_secure\_storage, considerado o “irmão mais seguro” do SharedPreferences. Ele oferece as mesmas funcionalidades, mas com criptografia integrada, sendo ideal para armazenar dados confidenciais, como tokens de acesso e informações sensíveis do usuário.

Para utilizar o pacote shared\_preferences, é preciso adicioná-lo como dependência do projeto, conforme indicado a seguir.

Dependência shared\_preferences adicionada ao projeto

1. [Dependencies](#):
2. [Shared\\_preferences](#): ^2.0.8
- 3.

Outra ação necessária para utilizar o shared\_preferences é adicionar sua biblioteca, conforme indicado a seguir.

Importando a biblioteca do shared\_preference

1. [Import package](#): shared\_preferences/shared\_preferences
- 2.

A seguir, é possível visualizar o código para salvar, recuperar e remover uma informação utilizando o shared\_preference.

Código para realizar operações com os dados utilizando o shared\_preference

```
1. /*
2. O fluxo de dados com o SharedPreferences:
3.
4. 1. Cria a instancia no Shared preferences para realizar
   as operações com os dados
5.
6. 2. Recupera a informação
7.
8. 3. Remove uma informação
9.
10. */
11. // Código para salvar uma informação
12. // Obtém a instância do SharedPreferences, que é
   responsável por gerenciar o armazenamento local
13. final prefs = await SharedPreferences.getInstance();
14. // Salva o valor ‘John’ associado à chave ‘username’ no
   armazenamento local
15. prefs.setString('username', 'John');
16.
17. // Código para recuperar uma informação
18. // Obtém novamente a instância do SharedPreferences
19. final prefs = await SharedPreferences.getInstance();
20. // Recupera o valor associado à chave ‘username’. Se a
   chave não existir, retorna uma string vazia (‘’)
21. final username = prefs.getString('username') ?? '';
22. // Exibe o nome de usuário recuperado no console
23. print('Username: $username');
24.
```

```

25. // Código para remover uma informação
26. // Obtém a instância do SharedPreferences
27. final prefs = await SharedPreferences.getInstance();
28. // Remove o dado associado à chave ‘username’ do
    armazenamento local
29. prefs.remove('username');
30.
31. // Código para verificar se uma informação existe
32. // Obtém a instância do SharedPreferences
33. final prefs = await SharedPreferences.getInstance();
34. // Verifica se a chave ‘username’ existe no armazenamento
    local
35. if (prefs.containsKey('username')) {
36.     // Caso a chave exista, imprime que o nome de usuário
        foi salvo anteriormente
37.     print('O nome de usuário foi salvo anteriormente');
38. } else {
39.     // Caso a chave não exista, imprime que o nome de
        usuário não foi salvo anteriormente
40.     print('O nome de usuário não foi salvo anteriormente');
41. }

```

Os principais comandos da biblioteca shared\_preferences são:

- **salvar** – usa setString, setInt, setBool etc., dependendo do tipo de dado;
- **recuperar** – usa getString, getInt, getBool para ler o valor associado a uma chave;
- **remover** – usa remove para apagar uma informação específica;
- **verificar** – usa containsKey para checar se uma chave existe.

Essas operações tornam o SharedPreferences uma ferramenta poderosa para armazenar informações simples e persistentes no Flutter.

## Sqflite

Sqflite é uma biblioteca Flutter que fornece uma interface para o SQLite, um sistema de banco de dados relacional leve embutido no dispositivo.

Com o Sqflite, é possível realizar operações CRUD (criar, ler, atualizar e excluir) em tabelas, oferecendo um armazenamento local robusto e estruturado em comparação com soluções mais simples, como o Shared Preferences.

Por utilizar SQL (linguagem popular de gerenciamento de banco de dados) e ser relativamente fácil de implementar, a biblioteca Sqflite se torna uma das primeiras opções de estudo para quem deseja aprender a trabalhar com persistência de dados no Flutter.

### Vantagens da Sqflite

- **Tem estrutura de dados complexa:** suporta a criação de bancos de dados relacionais com múltiplas tabelas e relações complexas entre dados. Para aplicativos que precisam armazenar dados estruturados em múltiplas tabelas, como um sistema de gerenciamento de estoque, finanças ou um aplicativo de agenda com tarefas, categorias e usuários, a Sqflite permite criar essas relações complexas entre dados. Por exemplo, um aplicativo de vendas pode usar tabelas para produtos, pedidos e clientes, vinculando-as por meio de chaves primárias e estrangeiras.
- **Armazena grande volume de dados:** adequada para armazenar e gerenciar grandes quantidades de dados de forma eficiente. Aplicativos que precisam lidar com grandes volumes de dados, como bibliotecas digitais, catálogos de produtos ou aplicativos de streaming off-line, beneficiam-se da Sqflite, que gerencia e organiza eficientemente grandes quantidades de registros.
- **Efetua consultas poderosas:** permite a execução de consultas SQL avançadas para filtrar, ordenar e agrupar dados conforme necessário. Em aplicativos que requerem filtros complexos, como um app de busca avançada de hotéis com critérios como localização, preço e

comodidades, as consultas SQL da Sqflite tornam possível implementar essas funcionalidades.

- **Tem boa persistência e performance:** oferece um armazenamento persistente e de alto desempenho para dados estruturados. Aplicativos que necessitam de acesso rápido a dados persistentes, como um aplicativo de notas ou diário pessoal, podem confiar no desempenho da Sqflite para carregar e salvar informações estruturadas sem comprometer a experiência do usuário.

## Desvantagens da Sqflite

- **Complexidade:** requer entendimento sólido de bancos de dados relacionais e SQL, o que pode aumentar a complexidade do desenvolvimento. Para aplicativos simples, como um contador de cliques ou um gerenciador de tarefas básico, usar a Sqflite pode ser excessivamente complexo. Isso ocorre porque o desenvolvedor precisa lidar com SQL e gerenciar a estrutura do banco, o que pode não ser necessário para projetos menores.
- **Configuração:** a configuração inicial e a manutenção de um banco de dados relacional podem ser mais trabalhosas comparadas a soluções mais simples. Em projetos pequenos ou com cronogramas apertados, a configuração inicial de tabelas e relações na Sqflite pode atrasar o desenvolvimento. Por exemplo, para um aplicativo que apenas salva preferências ou tokens, soluções como o SharedPreferences ou Hive seriam mais práticas.
- **Sobrecarga:** pode ser excessiva para armazenamento de dados simples ou pequenas quantidades de informações. Se o aplicativo armazena apenas pequenos pedaços de informação, como configurações ou estado de temas, usar Sqflite pode ser uma solução exagerada. Nesse caso, a simplicidade de alternativas mais leves, como o SharedPreferences, seria suficiente.

## Quando usar a biblioteca Sqflite

A Sqflite é ideal para aplicações que exigem:

- **armazenamento de dados estruturados** – quando os dados são complexos e requerem uma organização em tabelas com relações entre elas, como em um aplicativo de gestão de clientes ou inventário;
- **consultas e filtragem de dados** – quando há necessidade de realizar consultas avançadas ou filtragem de dados baseadas em múltiplos critérios;
- **grandes volumes de dados** – para aplicações que precisam armazenar e acessar uma grande quantidade de informações de maneira eficiente e rápida;
- **persistência longa e segura** – quando é necessário garantir que os dados serão persistidos de forma segura e disponível por longos períodos, mesmo após várias reinicializações do aplicativo.

A Sqflite é uma excelente ferramenta para desenvolvedores Flutter que precisam de um banco de dados local robusto. Ela é ideal para aplicativos que lidam com dados estruturados, grandes volumes de informação ou precisam de consultas SQL avançadas. Seus benefícios incluem a capacidade de criar e gerenciar tabelas de forma eficiente, atender a requisitos de armazenamento complexos e oferecer alta performance na manipulação de dados.

Para utilizar a Sqflite com o aplicativo será necessário adicioná-lo como uma dependência ao projeto, usando Sqflite: ^2.0.3+1, conforme a figura a seguir.

```

35 |
36   # The following adds the Cupertino Icons font to your application.
37   # Use with the CupertinoIcons class for iOS style icons.
38   cupertino_icons: ^1.0.2
39   http: ^0.13.3
40   sqflite: ^2.0.3+1
41   dev_dependencies:
42     flutter_test:
43       sdk: flutter

```

**Figura 4.2** – Adição da dependência Sqflite ^2.0.3+1.

Para demonstrar a utilização da Sqflite, desenvolveremos um aplicativo que exibe o nome e a idade de cães, conforme a figura a seguir.



**Figura 4.3** – Aplicativo com a informação do nome e idade de cães.

A seguir, é possível visualizar o código do aplicativo.

#### Aplicativo com banco de dados Sqflite

```
1. import 'package:flutter/material.dart';
2. import 'package:sqflite/sqflite.dart';
3.
4. // Pacote que permite a manipulação de banco de dados
   SQLite
5. import 'package:path/path.dart';
6.
7. // Pacote para manipulação de diretórios e caminhos de
   arquivos
```

```
8.  
9. void main() async {  
10.   runApp(MaterialApp(  
11.     home: Home(), // Define o widget inicial do aplicativo  
12.   ));  
13.  
14.   WidgetsFlutterBinding.ensureInitialized();  
15. // Garante que o Flutter esteja totalmente inicializado  
    antes de acessar o banco de dados  
16.  
17.   await _insertInitialDog(); // Insere dados iniciais no  
    banco de dados  
18.  
19.   var Rocky = Dog(id: 5, nome: "Rocky", idade: 5); // Cria  
    um objeto Dog para ser atualizado  
20.  
21.   await updateDog(Rocky); // Atualiza o objeto no banco  
    de dados  
22. }  
23.  
24. // Função para inserir dados iniciais no banco de dados  
25. Future<void> _insertInitialDog() async {  
26.  
27.   var database = await _initializeDatabase();  
28.  
29. // Inicializa o banco de dados  
30.   var Rocky = Dog(id: 5, nome: "Rocky", idade: 2); // Cria  
    um objeto Dog  
31.   var Caju = Dog(id: 6, nome: "Caju", idade: 6); // Cria  
    outro objeto Dog  
32.  
33.   // Exemplos comentados de inserção:  
34.   // await _insertDog(database, Caju);  
35.   // await _insertDog(database, Rocky);
```

```
36.     await deleteDog(6); // Remove o cachorro com ID 6 do
      banco de dados
37. }
38.
39. // Função para inicializar o banco de dados
40. Future<Database> _initializeDatabase() async {
41.
42.     return openDatabase(
43.         join(await getDatabasesPath(), 'dogs_a.db'),
44. // Define o caminho do banco de dados
45.         onCreate: (db, version) {
46.
47.             // Executa quando o banco de dados é criado pela
      primeira vez
48.             db.execute(
49.
50.                 'CREATE TABLE dogsa(
51. id INTEGER PRIMARY KEY, nome TEXT, idade INTEGER)', //
      Cria a tabela 'dogs'
52.             );
53.         },
54.         version: 1, // Define a versão do banco de dados
55.     );
56. }
57.
58. // Função para inserir um Dog no banco de dados
59. Future<void> _insertDog(
60. Database database, Dog dog) async {
61. // Converte o objeto Dog para um Map
62.
63.     await database.insert(
64.         'dogs', // Nome da tabela
65.         dog.toMap(),
66.         conflictAlgorithm: ConflictAlgorithm.replace, //
      Substitui caso já exista um registro com o mesmo ID
```

```
67.    );
68. }
69.
70. // Função para remover um Dog do banco de dados pelo ID
71. Future<void> deleteDog(int id) async {
72.
73.     final db = await _initializeDatabase(); // Inicializa o
        banco de dados
74.     await db.delete(
75.         'dogs', // Nome da tabela
76.         where: 'id = ?', // Condição para deletar (ID
            correspondente)
77.         whereArgs: [id], // Argumento para a condição
78.     );
79.     print("Deletando dado"); // Log de exclusão
80. }
81.
82. // Função para atualizar um Dog no banco de dados
83. Future<void> updateDog(Dog dog) async {
84.
85.     final db = await _initializeDatabase(); // Inicializa o
        banco de dados
86.     await db.update(
87.         'dogs', // Nome da tabela
88.         dog.toMap(), // Converte o objeto Dog para um Map
89.         where: 'id = ?', // Condição para atualizar (ID
            correspondente)
90.         whereArgs: [dog.id], // Argumento para a condição
91.     );
92. }
93.
94. // Widget Home - Interface do aplicativo
95. class Home extends StatefulWidget {
96.     const Home({super.key});
97.
```

```
98.    @override
99.    State<Home> createState() => _HomeState();
100.   }
101.
102.  class _HomeState extends State<Home> {
103.
104.    late Future<List<Dog>> _dogs; // Variável para
     armazenar a lista de cães
105.
106.    @override
107.    void initState() {
108.      super.initState();
109.      _dogs = _fetchDogs(); // Recupera a lista de cães ao
     inicializar o widget
110.    }
111.
112.    // Função para buscar os dados da tabela 'dogsa'
113.    Future<List<Dog>> _fetchDogs() async {
114.      var database = await _initializeDatabase(); //
     Inicializa o banco de dados
115.      final List<Map<String, dynamic>> maps = await
     database.query('dogsa'); // Consulta todos os dados da
     tabela
116.
117.      // Converte os dados do Map para uma lista de
     objetos Dog
118.      return List.generate(maps.length, (i) {
119.        return Dog(
120.          id: maps[i]['id'],
121.          nome: maps[i]['nome'],
122.          idade: maps[i]['idade'],
123.        );
124.      });
125.    }
126.
```

```
127. @override
128. Widget build(BuildContext context) {
129.
130.     return Scaffold(
131.         appBar: AppBar(
132.             title: Text("APP BD"), // Título da aplicação
133.         ),
134.         body: FutureBuilder<List<Dog>>(
135.             future: _dogs, // Futuro que contém os dados dos
136.                 cães
137.                 builder: (context, snapshot) {
138.                     if (snapshot.connectionState ==
139.                         ConnectionState.waiting) {
140.                         // Exibe um indicador de carregamento
141.                         enquanto os dados estão sendo buscados
142.                         return Center(child:
143.                             CircularProgressIndicator()));
144.                     } else if (snapshot.hasError) {
145.                         // Exibe uma mensagem de erro caso ocorra
146.                         algum problema
147.                         return Center(child: Text('Error: ${snapshot.
148.                             error}'));
149.                     } else {
150.                         // Exibe a lista de cães
151.                         final dogs = snapshot.data!;
152.                         return ListView.builder(
153.                             itemCount: dogs.length, // Número de itens
na lista
154.                             itemBuilder: (context, index) {
155.                                 final dog = dogs[index];
156.                                 return ListTile(
157.                                     title: Text(dog.nome), // Nome do
cachorro
158.                                     subtitle: Text('Idade: ${dog.idade}'),
// Idade do cachorro
159.                                 );
160.                             );
161.                         );
162.                     );
163.                 );
164.             );
165.         );
166.     );
167. }
```

```
154.           },
155.           );
156.       }
157.       },
158.       ),
159.   );
160. }
161. }
162.
163. // Classe Dog - Representa os dados de um cachorro
164. class Dog {
165.     final int id; // ID do cachorro
166.     String nome; // Nome do cachorro
167.     final int idade; // Idade do cachorro
168.
169.     // Construtor da classe Dog
170.     Dog({
171.         required this.id,
172.         required this.nome,
173.         required this.idade,
174.     });
175.
176.     // Método para converter um objeto Dog em um Map para
177.     // salvar no banco de dados
178.     Map<String, dynamic> toMap() {
179.         return {
180.             'id': id, // Chave 'id' com o valor correspondente
181.             'nome': nome, // Chave 'nome' com o valor
182.             correspondente
183.             'idade': idade, // Chave 'idade' com o valor
184.             correspondente
185.         };
186.     }
187. }
```

## Hive

Hive é uma biblioteca leve de banco de dados NoSQL para Flutter, projetada para ser rápida, segura e fácil de usar. Ao contrário de soluções baseadas em SQL, a Hive armazena dados em um formato binário, permitindo um acesso mais eficiente. É especialmente adequada para dispositivos móveis devido ao seu baixo consumo de recursos e sua alta performance.

### Vantagens da Hive

- **Desempenho:** é extremamente rápida para leitura e gravação, sendo ideal para aplicativos que exigem alta performance, como gerenciadores de tarefas ou aplicativos de diário, melhorando a experiência em dispositivos com recursos limitados.
- **Facilidade de uso:** oferece uma API intuitiva, simplificando o armazenamento e recuperação de dados. Por exemplo, salvar e recuperar preferências do usuário em um aplicativo de configuração é fácil com a Hive.
- **Sem dependência de SQL:** por ser NoSQL, elimina a necessidade de criar tabelas ou escrever consultas SQL, agilizando o desenvolvimento, especialmente em aplicativos simples, como rastreadores de hábitos ou leitores de QR Code.
- **Off-line-first:** projetada para funcionar off-line, garantindo acesso aos dados mesmo sem internet. Ideal para aplicativos como listas de compras, notas ou rastreadores de atividades, especialmente em regiões com conectividade instável.
- **Suporte a tipos complexos:** permite armazenar dados como listas e mapas sem conversão para tipos primitivos, facilitando o desenvolvimento de apps como organizadores pessoais ou gerenciadores de tarefas.

## Desvantagens da Hive

- **Escalabilidade:** não é adequada para grandes volumes de dados ou aplicações que exigem consultas complexas, como lojas virtuais com milhares de produtos.
- **Falta de consultas avançadas:** não suporta filtros ou relacionamentos complexos, o que pode limitar a funcionalidade em aplicativos como catálogos de produtos ou sistemas de recomendação.
- **Crescimento da base de dados:** o desempenho pode ser impactado à medida que o volume de dados cresce, tornando-a menos eficiente para aplicativos que acumulam grandes quantidades de informações, como diários ou sistemas de log.

## Exemplos de aplicações da Hive

- **Gerenciamento de tarefas:** aplicativos de lista de tarefas que armazenam localmente as atividades do usuário, incluindo prazos, categorias e estados de conclusão. A Hive permite salvar e recuperar dados de forma rápida, mesmo off-line.
- **Diários e notas:** aplicativos como diários pessoais ou gerenciadores de notas, que precisam armazenar informações textuais, listas ou imagens. A Hive facilita o armazenamento de dados complexos, como mapas e listas.
- **Jogos móveis:** jogos que precisam salvar dados como progresso, pontuações, configurações do jogador ou itens adquiridos. A Hive é ideal para gerenciar esses dados de forma eficiente e com alta performance.
- **Aplicativos de produtividade:** aplicativos como rastreadores de hábitos, gerenciadores de tempo ou planejadores de metas que precisam armazenar informações simples a moderadamente complexas, como listas de tarefas ou progresso.

- **Armazenamento de preferências:** aplicativos que salvam configurações do usuário, como temas, idiomas ou preferências de notificações. A Hive permite gerenciar essas informações de forma rápida e prática.
- **E-commerce off-line:** lojas virtuais que precisam salvar dados de produtos localmente para funcionar off-line, como catálogos de produtos ou histórico de compras.
- **Cache de dados:** aplicativos que armazenam dados de APIs localmente para melhorar a performance e reduzir o consumo de internet, como aplicativos de notícias, previsão do tempo ou streaming.
- **Aplicativos educacionais:** aplicações que salvam progresso de aprendizado, resultados de quizzes ou anotações do usuário. Por exemplo, um app de estudo de idiomas pode armazenar o histórico de palavras aprendidas localmente.
- **Gerenciadores de contatos:** aplicativos que salvam informações de contatos do usuário, como nomes, números de telefone e e-mails, permitindo acessá-los mesmo sem conexão.
- **Aplicativos de finanças pessoais:** aplicativos que registram transações financeiras, categorias de gastos e saldos de contas. A Hive permite armazenar essas informações de forma segura e acessível.
- **Rastreadores de saúde e bem-estar:** aplicativos que monitoram exercícios, calorias consumidas ou ciclos de sono. A Hive permite salvar e organizar os dados do usuário de forma eficiente.
- **Aplicativos de eventos:** aplicações que gerenciam agendas e lembretes de eventos, como calendários ou rastreadores de compromissos.

Como exemplo de uso da Hive, será desenvolvido um aplicativo para armazenar os dados de alunos, como nome, RA e notas.

Para utilizar Hive para armazenar os dados, devemos adicioná-la como dependência do projeto, conforme a figura a seguir.

```

dependencies:
  flutter:
    sdk: flutter
  hive: ^2.0.5
  hive_flutter: ^1.1.0
  path_provider: ^2.0.11 # Para ajudar a encontrar o caminho correto do diretório

```

**Figura 4.4** – Adicionar Hive como dependência do projeto.

A seguir, é possível visualizar o código de um aplicativo para salvar nome, RA e notas de cada aluno.

#### Código com Hive do projeto

```

1. import 'package:flutter/material.dart';
2. import 'package:hive/hive.dart';
3. import 'package:hive_flutter/hive_flutter.dart';
4. import 'package:path_provider/path_provider.dart';
5.
6. void main() async {
7.   // Inicia o Hive e carrega o banco de dados
8.   WidgetsFlutterBinding.ensureInitialized();
9.   final appDocumentDir = await
10.      getApplicationDocumentsDirectory();
11.
12.   // Registra o adaptador se necessário (caso você esteja
13.     usando objetos customizados)
13.   // Hive.registerAdapter(AlunoAdapter());
14.
15.   runApp(MyApp());
16. }
17.
18. class MyApp extends StatelessWidget {
19.   @override
20.   Widget build(BuildContext context) {
21.     return MaterialApp(
22.       title: 'App de Alunos',

```

```
23.     theme: ThemeData(  
24.         primarySwatch: Colors.blue,  
25.     ),  
26.     home: AlunosPage(),  
27. );  
28. }  
29.  
30.
```

A seguir, é possível visualizar o modelo que será utilizado para salvar os dados de cada aluno.

#### Modelo para salvar os dados de cada aluno

```
1. /*  
2. import 'package:hive/hive.dart';:  
3.  
4. Esta linha importa o pacote hive.dart, que contém as  
    funcionalidades necessárias para utilizar o Hive no  
    Flutter. Ele oferece a API para manipulação de caixas de  
    dados, criação de adaptadores e outros recursos.  
5.  
6. part 'aluno.g.dart';:  
7. O Hive utiliza um mecanismo de geração de código para  
    criar os adaptadores que convertem os objetos em formatos  
    que podem ser armazenados e recuperados do banco de dados.  
    O comando part indica que o arquivo aluno.dart faz parte  
    do arquivo aluno.g.dart, que será gerado automaticamente  
    quando você rodar o comando do build runner.  
8. O arquivo aluno.g.dart será responsável por gerar o  
    adaptador que vai converter o objeto Aluno para um formato  
    legível para o Hive e vice-versa.  
9.  
10. HiveType(typeId: 0):  
11. A anotação @HiveType é usada para marcar a classe Aluno  
    como um tipo que será armazenado no Hive. O typeId é um
```

identificador único para o tipo de objeto, e você deve garantir que ele seja único para cada tipo de objeto armazenado no banco de dados.

12. O typeId ajuda o Hive a diferenciar os objetos de diferentes tipos quando armazenados. No exemplo, o tipo Aluno tem o typeId 0, mas se você tivesse outra classe de modelo (como Professor), ela teria outro typeId.
- 13.
14. @HiveField(0):
15. A anotação @HiveField é usada para mapear os campos da classe para os índices dentro do Hive. Cada campo da classe será armazenado com um índice associado que o Hive usa para organizar os dados.
- 16.
17. O número dentro dos parênteses (0, 1, 2, etc.) representa o índice do campo dentro da classe. O índice precisa ser único dentro da classe. O Hive usará esses índices para salvar os dados de forma eficiente.
- 18.
19. @HiveField(0): Marca o campo nome para ser armazenado com o índice 0.
- 20.
21. @HiveField(1): Marca o campo ra para ser armazenado com o índice 1.
- 22.
23. @HiveField(2): Marca o campo nota para ser armazenado com o índice 2.
- 24.
25. O construtor da classe Aluno exige que você forneça valores para os três campos: nome, ra e nota quando for criar uma instância de Aluno.
26. O modificador required garante que esses campos não podem ser nulos ao criar o objeto Aluno.
- 27.

28. Agora, com a estrutura completa, o código define uma classe Aluno que pode ser armazenada no Hive, com os seguintes campos:
  - 29.
  30. nome (String): Nome do aluno.
  31. ra (String): Registro Acadêmico do aluno.
  32. nota (double): Nota do aluno.
  33. Como o Hive usa isso
  34. Criação do Adaptador: Quando você executa o comando flutter packages pub run build\_runner build, o Hive gera um arquivo aluno.g.dart, que contém o adaptador necessário para a classe Aluno. Esse adaptador é responsável por serializar (converter o objeto para um formato que o Hive possa armazenar) e desserializar (conversão do formato de volta para um objeto Aluno) a classe Aluno.
  - 35.
  36. Armazenando e Recuperando Dados:
  - 37.
  38. Quando você armazenar um Aluno no Hive, ele será convertido pelo adaptador em um formato que pode ser salvo em uma “caixa” de dados (Box). O HiveField garante que os dados de cada campo sejam armazenados corretamente com os índices respectivos.
  39. Quando você recuperar um Aluno do Hive, o adaptador converte os dados de volta para um objeto Aluno que pode ser utilizado normalmente no seu código.
  40. \*/
  41. import 'package:hive/hive.dart';
  - 42.
  43. part 'aluno.g.dart'; // Isso será gerado automaticamente pelo Hive
  - 44.
  45. @HiveType(typeId: 0) // ID para o tipo de objeto
  46. class Aluno {
  47.     @HiveField(0)

```

48.     final String nome;
49.
50.     @HiveField(1)
51.     final String ra;
52.
53.     @HiveField(2)
54.     final double nota;
55.
56.     Aluno({
57.         required this.nome,
58.         required this.ra,
59.         required this.nota,
60.     });
61. }
62.
63.

```

Após criar o arquivo aluno.dart, é necessário executar o comando indicado a seguir para criar o arquivo aluno.g.dart, necessário para o banco de dados Hive funcionar corretamente.

Comando para gerar o arquivo .g do banco de dados

1. Comando para criar o arquivo responsável por criar o arquivo .g do banco de dados **Hive**
2. flutter packages pub run build\_runner build

Por fim, podemos verificar, a seguir, o código completo do aplicativo para salvar as informações de alunos.

Código completo para salvar informações utilizando Hive

1. import 'package:flutter/material.dart';
2. import 'package:hive/hive.dart';
3. import 'aluno.dart';
- 4.

```
5. class AlunosPage extends StatefulWidget {
6.   @override
7.   _AlunosPageState createState() => _AlunosPageState();
8. }
9.
10. class _AlunosPageState extends State<AlunosPage> {
11.   final TextEditingController _nomeController =
12.     TextEditingController();
13.   final TextEditingController _raController =
14.     TextEditingController();
15.   final TextEditingController _notaController =
16.     TextEditingController();
17.   @override
18.   void initState() {
19.     super.initState();
20.     _openBox();
21.   }
22.
23.   Future<void> _openBox() async {
24.     _alunosBox = await Hive.openBox<Aluno>('alunos');
25.     setState(() {});
26.   }
27.
28.   void _adicionarAluno() {
29.     final nome = _nomeController.text;
30.     final ra = _raController.text;
31.     final nota = double.tryParse(_notaController.text) ??
32.       0.0;
33.     final aluno = Aluno(nome: nome, ra: ra, nota: nota);
34.     _alunosBox.add(aluno);
35. }
```

```
36.      // Limpar campos após adicionar
37.      _nomeController.clear();
38.      _raController.clear();
39.      _notaController.clear();
40.      setState(() {});
41.    }
42.
43.    @override
44.    Widget build(BuildContext context) {
45.      return Scaffold(
46.        appBar: AppBar(
47.          title: Text('Cadastro de Alunos'),
48.        ),
49.        body: Column(
50.          children: [
51.            Padding(
52.              padding: const EdgeInsets.all(8.0),
53.              child: TextField(
54.                controller: _nomeController,
55.                decoration: InputDecoration(labelText:
56.                  'Nome do Aluno'),
57.              ),
58.            Padding(
59.              padding: const EdgeInsets.all(8.0),
60.              child: TextField(
61.                controller: _raController,
62.                decoration: InputDecoration(labelText: 'RA
63.                  do Aluno'),
64.              ),
65.            Padding(
66.              padding: const EdgeInsets.all(8.0),
67.              child: TextField(
68.                controller: _notaController,
```

```
69.                 decoration: InputDecoration(labelText:
    'Nota do Aluno'),
70.                 keyboardType: TextInputType.number,
71.             ),
72.             ),
73.             ElevatedButton(
74.                 onPressed: _adicionarAluno,
75.                 child: Text('Adicionar Aluno'),
76.             ),
77.             Expanded(
78.                 child: ValueListenableBuilder(
79.                     valueListenable: _alunosBox.listenable(),
80.                     builder: (context, Box<Aluno> box, _) {
81.                         if (box.isEmpty) {
82.                             return Center(child: Text('Nenhum aluno
    cadastrado.'));
83.                         }
84.
85.                         return ListView.builder(
86.                             itemCount: box.length,
87.                             itemBuilder: (context, index) {
88.                                 final aluno = box.getAt(index);
89.                                 return ListTile(
90.                                     title: Text(aluno!.nome),
91.                                     subtitle: Text('RA: ${aluno.ra} -
    Nota: ${aluno.nota}'),
92.                                     );
93.                                 },
94.                                 );
95.                             },
96.                             ),
97.                             ),
98.                             ],
99.                         ),
100.                     );
101.                 }
```

```
102. }  
103.  
104.
```

A próxima opção para armazenar dados em um aplicativo mobile é o Drift, cujas funcionalidades serão apresentadas a seguir.

## Drift

O Drift (anteriormente conhecido como Moor) é uma biblioteca Flutter que oferece uma maneira eficiente e segura de gerenciar bancos de dados SQLite utilizando a linguagem Dart.

Ele permite escrever consultas SQL e manipular dados usando objetos Dart, proporcionando uma experiência de desenvolvimento mais fluida e menos propensa a erros em comparação com a abordagem SQL pura, como com a Sqflite.

### Vantagens do Drift

- **Integração com Dart:** permite manipular dados diretamente usando objetos e classes Dart, o que torna o código mais legível e menos suscetível a erros de sintaxe SQL;
- **Abstração:** oferece uma camada de abstração sobre o SQL, simplificando operações comuns de banco de dados e reduzindo a complexidade do código;
- **Migrations:** suporta migrações de banco de dados, facilitando a atualização e a evolução da estrutura do banco de dados ao longo do tempo;
- **Type safety:** garante a segurança de tipos em tempo de compilação, ajudando a evitar erros comuns de tipo que podem ocorrer com SQL puro;
- **Reactiveness:** suporta operações reativas, permitindo que as interfaces do usuário sejam automaticamente atualizadas quando os dados no banco de dados mudam.

## Desvantagens do Drift

- **Curva de aprendizado:** pode ter uma curva de aprendizado mais íngreme para desenvolvedores que não estão familiarizados com o conceito de ORMs (Object-Relational Mappers);
- **Desempenho:** a camada adicional de abstração pode introduzir uma leve sobrecarga de desempenho em comparação com consultas SQL diretas;
- **Complexidade inicial:** de todos os pacotes citados, o Drift é o que tem a configuração inicial mais complexa devido a vários fatores, incluindo a necessidade de definir entidades e gerenciar migrações.

## Quando usar o Drift?

O Drift é ideal para aplicações que:

- **beneficiam-se de abstração** – desenvolvedores que preferem trabalhar com objetos Dart em vez de escrever SQL diretamente podem se beneficiar da abstração oferecida pelo Drift;
- **necessitam de reactivity** – aplicativos que precisam reagir automaticamente a mudanças no banco de dados, como interfaces dinâmicas que se atualizam com novas entradas de dados;
- **demandam migrations** – projetos que exigem migrações para gerenciar mudanças na estrutura do banco de dados ao longo do tempo;
- **valorizam type safety** – para garantir que operações de banco de dados sejam verificadas em tempo de compilação, reduzindo o risco de erros.

Enquanto Sqflite é uma biblioteca robusta para acessar diretamente o SQLite, exigindo que os desenvolvedores escrevam consultas SQL manualmente, o Drift oferece uma abordagem mais integrada ao Dart, permitindo que os desenvolvedores manipulem dados usando objetos Dart. Essa

integração melhora a legibilidade do código e reduz a quantidade de erros comuns associados à escrita de SQL manual.

Além disso, o Drift suporta operações reativas e migrações de banco de dados, oferecendo uma solução mais completa e flexível para o gerenciamento de dados em aplicações Flutter.

O primeiro passo é adicionar o Drift como dependência do projeto, conforme mostrado na figura a seguir.

```
dependencies:
  flutter:
    sdk: flutter
  drift: ^2.6.0
  sqlite3_flutter_libs: ^0.5.13
  path_provider: ^2.1.1

dev_dependencies:
  drift_dev: ^2.6.0
  build_runner: ^2.4.6
```

**Figura 4.5 – Adicionando o Drift como dependência do projeto.**

Após adicionar o Drift como dependência do projeto, precisamos executar o comando que atualiza essa dependência.

Comando para atualizar as dependências do projeto

1. flutter pub get
2. Comando para atualizar as dependências do projeto
- 3.
- 4.

A seguir, é possível visualizar o código app\_database.dart que cria a tabela na qual serão salvas as informações no banco de dados Drift.

Código app\_database.dart cria a estrutura para salvar os dados

1. import 'package:drift/drift.dart';
2. import 'package:drift/native.dart';

```
3. import 'package:path_provider/path_provider.dart';
4. import 'dart:io';
5. import 'package:path/path.dart' as p;
6.
7. part 'app_database.g.dart';
8.
9. // Define a tabela para os alunos
10. class Students extends Table {
11.   IntColumn get id => integer().autoIncrement()(); // ID
        auto-incrementável
12.   TextColumn get name => text().withLength(min: 1, max: 50)
        (); // Nome do aluno
13.   TextColumn get ra => text().unique()(); // RA do aluno
14.   RealColumn get grade => real()(); // Nota
15. }
16.
17. // Classe do banco de dados
18. @DriftDatabase(tables: [Students])
19. class AppDatabase extends _$AppDatabase {
20.   AppDatabase() : super(_openConnection());
21.
22.   @override
23.   int get schemaVersion => 1;
24.
25.   // Insere um aluno
26.   Future<int> insertStudent(StudentsCompanion student) {
27.     return into(students).insert(student);
28.   }
29.
30.   // Busca todos os alunos
31.   Future<List<Student>> getAllStudents() {
32.     return select(students).get();
33.   }
34.
35.   // Atualiza um aluno
```

```
36.     Future<bool> updateStudent(Student student) {  
37.         return update(students).replace(student);  
38.     }  
39.  
40.     // Deleta um aluno  
41.     Future<int> deleteStudent(int id) {  
42.         return (delete(students)..where((tbl) => tbl.  
        id.equals(id))).go();  
43.     }  
44. }  
45.  
46. // Conexão com o banco  
47. LazyDatabase _openConnection() {  
48.     return LazyDatabase(() async {  
49.         final dbFolder = await  
        getApplicationDocumentsDirectory();  
50.         final file = File(p.join(dbFolder.path, 'app.sqlite'));  
51.         return NativeDatabase(file);  
52.     });  
53. }  
54.
```

Após criar o arquivo app\_database.dart, devemos executar o comando que gera os arquivos auxiliares para configurar o banco de dados Drift, conforme indicado a seguir.

Comando para gerar os arquivos necessários de inicialização  
do banco com Drift

1. Comando para gerar os arquivos de inicialização do banco
2. flutter pub run build\_runner build

O código completo do aplicativo utilizando Drift pode ser visualizado a seguir.

## Aplicativo utilizando Drift

```
1. import 'package:flutter/material.dart';
2. import 'app_database.dart';
3.
4. void main() {
5.   runApp(StudentApp());
6. }
7.
8. class StudentApp extends StatefulWidget {
9.   @override
10.  _StudentAppState createState() => _StudentAppState();
11. }
12.
13. class _StudentAppState extends State<StudentApp> {
14.   late AppDatabase db;
15.   List<Student> students = [];
16.
17.   final _nameController = TextEditingController();
18.   final _raController = TextEditingController();
19.   final _gradeController = TextEditingController();
20.
21.   @override
22.   void initState() {
23.     super.initState();
24.     db = AppDatabase();
25.     _loadStudents();
26.   }
27.
28.   Future<void> _loadStudents() async {
29.     final loadedStudents = await db.getAllStudents();
30.     setState(() {
31.       students = loadedStudents;
32.     });
33.   }
34.
```



```
66.                     SizedBox(height: 10),
67.                     ElevatedButton(onPressed: _addStudent,
68.                         child: Text('Adicionar Aluno')),
69.                         ],
70.                         ),
71.                         ),
72.                         Expanded(
73.                             child: ListView.builder(
74.                                 itemCount: students.length,
75.                                 itemBuilder: (context, index) {
76.                                     final student = students[index];
77.                                     return ListTile(
78.                                         title: Text('${student.name} (RA:
79. ${student.ra})'),
80.                                         subtitle: Text('Nota: ${student.grade.
81.                               toStringAsFixed(2)}'),
82.                                         trailing: IconButton(
83.                                             icon: Icon(Icons.delete, color:
84.                                               Colors.red),
85.                                             onPressed: () => _
86.                                                 deleteStudent(student.id),
87.                                                 ),
88.                                                 ),
89.                                                 ),
90.                                                 );
91.             }
92.         }
93.     }
```

## Firebase

O Firebase é uma plataforma oferecida pelo Google que fornece soluções backend integradas para desenvolvimento de aplicativos móveis e web. Ele utiliza um banco de dados não relacional baseado em JSON, como o Realtime Database ou o Firestore. A principal diferença entre eles é que o Firestore permite estruturas mais organizadas e consultas mais robustas.

### Vantagens do Firebase

- Integração nativa com Flutter, Android e iOS.
- Suporte a funcionalidades como autenticação, armazenamento de arquivos, notificações push e hospedagem na nuvem.
- Tempo real: excelente para aplicativos colaborativos (chat, jogos etc.).
- Escalabilidade automática sem necessidade de configurações complexas.

### Desvantagens do Firebase

- Dados não relacionais podem dificultar o armazenamento de informações complexas ou altamente conectadas.
- Pode se tornar caro conforme o número de leituras, gravações ou transferências de dados cresce.
- Estrutura baseada em JSON exige planejamento cuidadoso para evitar problemas de performance e redundância.

### Quando usar o Firebase

- Para aplicativos que exigem sincronização em tempo real (por exemplo: chats, dashboards, apps colaborativos).
- Em projetos que precisam ser desenvolvidos rapidamente com soluções backend prontas.

- Para aplicativos com interações simples entre os dados, sem a necessidade de operações complexas (como joins).

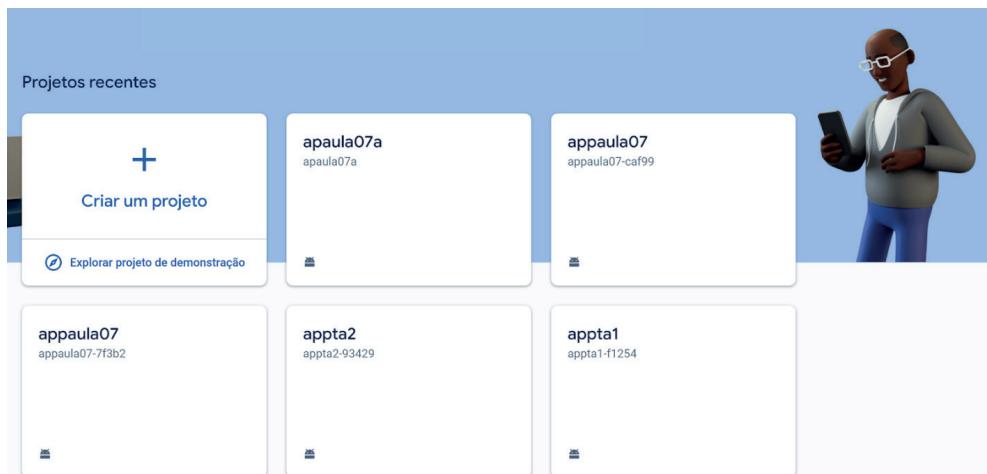
### SAIBA MAIS

A plataforma Firebase tem diversos recursos, como CloudFirestore, Firebase Authentication, Machine Learning e Datastore, entre outros. Para utilizá-los, é necessário realizar o cadastro no site, que pode ser acessado pelo QR Code ou pelo link disponível em:



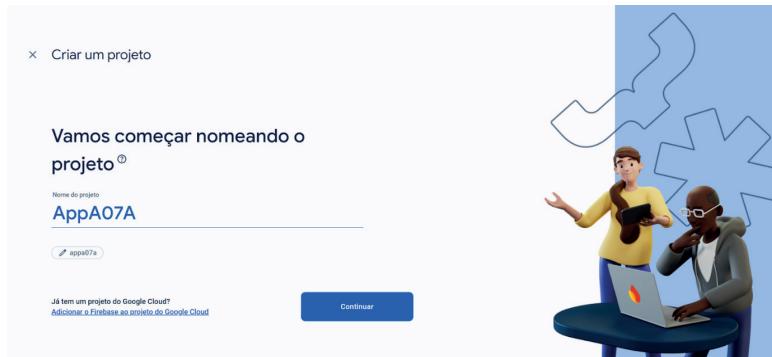
<https://firebase.google.com>. Acesso em: 20 ago. 2025.

Após acessar o link, clique em Go to console; o site abrirá a tela mostrada na figura a seguir.



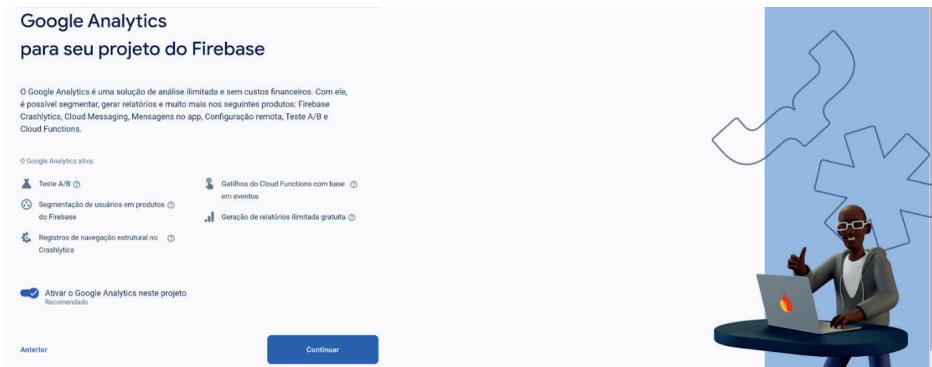
**Figura 4.6** – Página do Firebase para criar um projeto.

Ao clicar em Criar um projeto, o site abrirá a tela da figura a seguir.



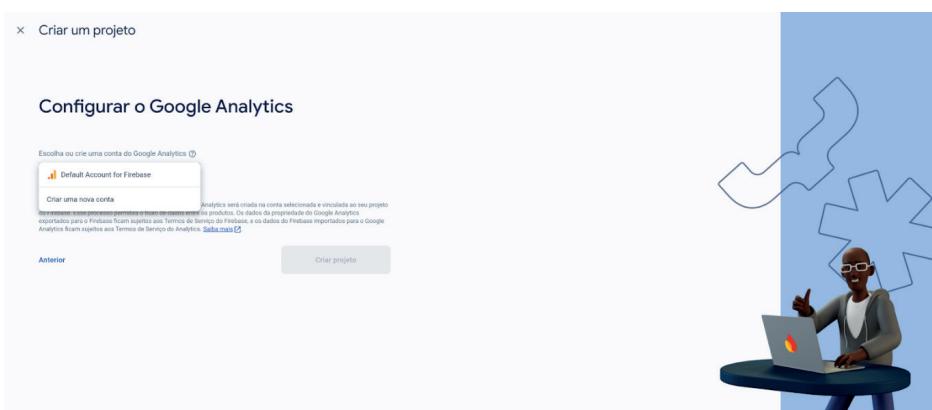
**Figura 4.7 – Tela com o projeto sendo criado.**

Ao clicar no botão Continuar, chegamos na tela que mostra o projeto criado e a configuração do Analytics (ver figura a seguir).



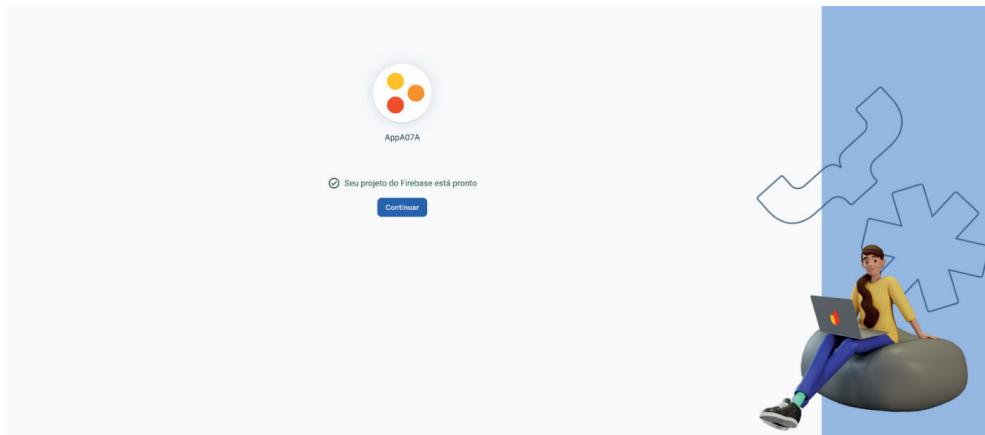
**Figura 4.8 – Tela com o projeto criado e configuração do Analytics.**

Após clicar em Continuar, o site irá para a tela mostrada na figura a seguir.



**Figura 4.9 – Configuração da conta para acessar o Firebase.**

Com a escolha da conta realizada, o projeto estará pronto (ver a figura a seguir).



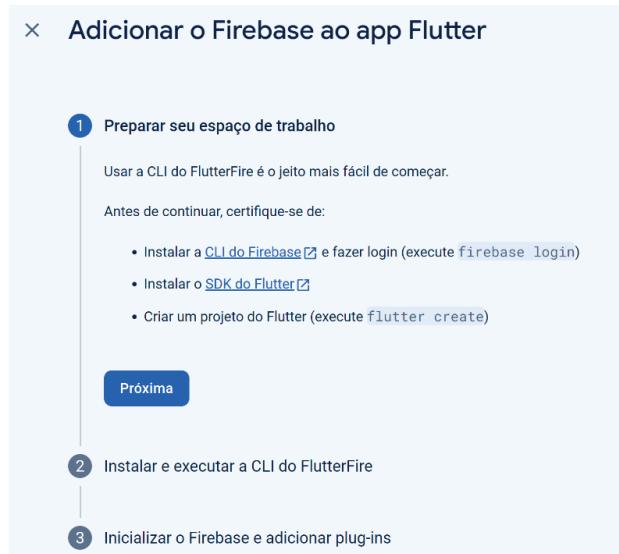
**Figura 4.10** – Tela que mostra o projeto criado.

Com o projeto criado, precisamos escolher a qual plataforma o Firebase vai se conectar: Android, Flutter, iOS ou Web (ver a figura a seguir).



**Figura 4.11** – Plataforma que realiza a conexão com Firebase.

A plataforma que escolheremos para realizar a conexão com o Firebase é o Flutter (ver a figura a seguir).



**Figura 4.12** – Tela de passo a passo para conectar o Firebase com o Flutter.

Para realizar a conexão do Firebase com o Flutter, é necessário primeiro realizar o download da CLI do Firebase, conforme a figura a seguir.

[Windows](#) [macOS](#) [Linux](#)

**janelas**

Você pode instalar a CLI do Firebase para Windows usando uma das seguintes opções:

Opção	Descrição	Recomendado para...
binário autônomo	Baixe o binário independente para a CLI. Em seguida, você pode acessar o executável para abrir um shell onde poderá executar o comando <code>firebase</code> .	Novos desenvolvedores Desenvolvedores que não usam ou não estão familiarizados com o <a href="#">Node.js</a>
npm	Use npm (o Node Package Manager) para instalar a CLI e ativar o comando <code>firebase</code> disponível globalmente.	Desenvolvedores usando <a href="#">Node.js</a>

**binário autônomo** [npm](#)

Para fazer download e executar o binário para a CLI do Firebase, siga estas etapas:

1. Faça download do [binário Firebase CLI para Windows](#).
2. Acesse o binário para abrir um shell onde você pode executar o comando `firebase`.

**Figura 4.13** – Tela de download do CLI.

CLI é um terminal para realizar as configurações do Firebase, como login e logout, e a conexão do aplicativo por meio do application ID.

Para instalar o CLI do Firebase, abriremos o Power Shell, conforme mostrado na figura a seguir.

binário autônomo [npm](#)

Para usar [npm](#) (o Node Package Manager) para instalar o Firebase CLI, siga estas etapas:

1. Instale o [Node.js](#) usando [nvm-windows](#) (o Node Version Manager). A instalação do Node.js instala automaticamente as ferramentas de comando [npm](#).
2. Instale a Firebase CLI via [npm](#) executando o seguinte comando:

```
$ npm install -g firebase-tools
```

Este comando ativa o comando `firebase` disponível globalmente.

Observação: se o comando `npm install -g firebase-tools` falhar, talvez seja necessário alterar as permissões do [npm](#).

**Figura 4.14** – Comando para instalar as ferramentas do Firebase.

Com o CLI do Firebase instalado e aberto, será possível realizar o logout e depois o login com o e-mail cadastrado, conforme mostra a figura a seguir.

**Figura 4.15** – Realizando login no Firebase.

Com o login realizado, aparece a opção de coleta de informações pessoais; optaremos por não partilhá-las, conforme a vemos na figura a seguir.

```
firebase login
```

- Let's make sure your Firebase CLI is ready...

- Looks like your CLI needs to be set up.

- This may take a moment

+ Alright, your CLI is set up!

Already logged in as danielvieira2006@gmail.com

+ You can now use the 'firebase' or 'npm' commands!

- For more help see <https://firebase.google.com/docs/cli/>

---

```
> firebase logout
```

+ Logged out from danielvieira2006@gmail.com

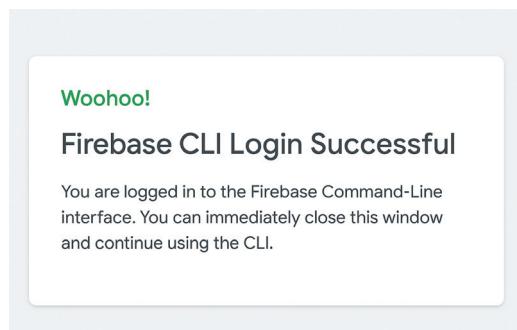
```
> firebase login
```

i Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve our product  
s It is collected in accordance with Google's privacy policy (<https://policies.google.com/privacy>) and is not used to identify you.

? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? (Y/n) n

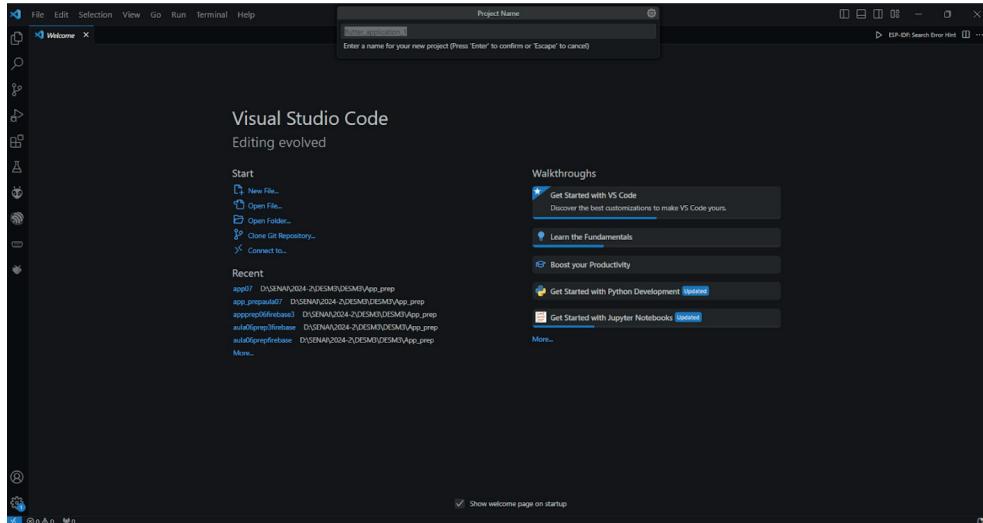
**Figura 4.16** – Selecionando a opção de não coletar dados.

Na figura a seguir, é possível visualizar a tela indicando que o login foi bem-sucedido.



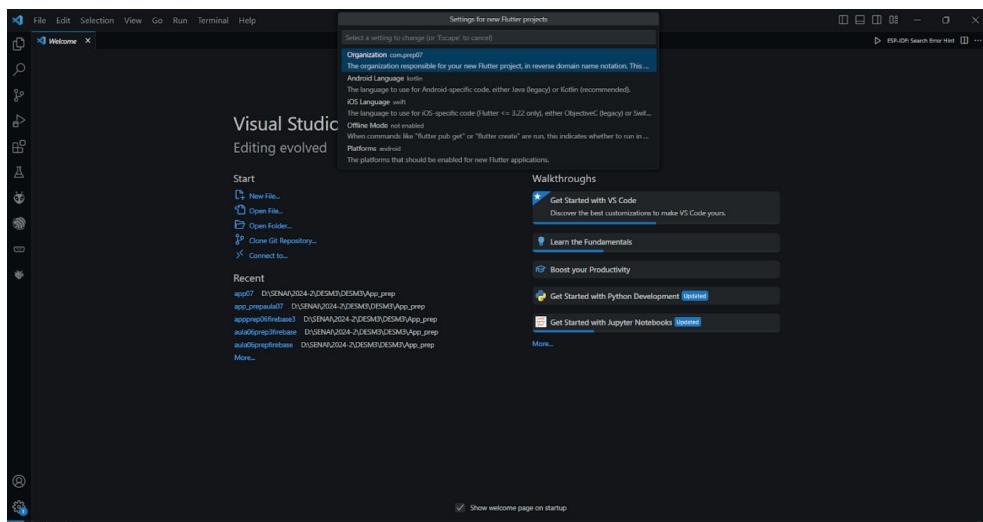
**Figura 4.17** – Login realizado no Firebase.

O próximo passo será criar o projeto Flutter no VS CODE. As etapas do processo podem ser acompanhadas nas figuras a seguir.



**Figura 4.18 – Projeto sendo criado no VS CODE.**

Nesta primeira tela, clicaremos na engrenagem; em organization, vamos digitar com.[nome do projeto] – para este exercício, usaremos com. prep7 (ver Figura 4.19). Esse parâmetro é importante, pois através dele realizaremos a conexão do aplicativo com o Firebase.



**Figura 4.19 – Configuração do parâmetro organization.**

Com a configuração realizada, pressionamos Esc e Enter: o projeto Flutter foi criado no VS CODE. Agora, podemos ativar o flutterfire, conforme mostra a figura a seguir.

The screenshot shows the VS Code interface with the following content:

- Preparar seu espaço de trabalho**
- 2 Instalar e executar a CLI do FlutterFire**
- Em qualquer diretório, execute o comando:  
\$ dart pub global activate flutterfire\_cli
- Em seguida, na raiz do diretório do seu projeto do Flutter, execute o comando:  
\$ flutterfire configure --project=appa07a
- Com isso, seus apps são registrados automaticamente por plataforma com o Firebase, e um arquivo de configuração `lib/firebase_options.dart` é adicionado ao seu projeto do Flutter.

At the bottom, there are two buttons: **Anterior** and **Próxima**.

**Figura 4.20** – Comandos para ativar o flutterfire no VS CODE.

Usaremos o comando `dart pub global activate flutterfire_cli` para ativar a CLI, conforme apresentado na figura a seguir.

A terminal window showing the command being run:

```
PS D:\SENAI\2024-2\DESENV\DESENV\app_prep\app_a08> dart pub global activate flutterfire_cli
```

**Figura 4.21** – Comando no terminal para ativar a CLI.

Na figura a seguir, é possível visualizar o Flutterfire ativado, que será responsável por conectar o aplicativo ao Firebase.

A terminal window showing the output of the activation command:

```
PS D:\SENAI\2024-2\DESENV\DESENV\app_prep\app_a08> dart pub global activate flutterfire_cli
Package flutterfire_cli is currently active at version 1.0.8.
Downloading packages... (2.9%)
The flutterfire_cli package is already activated at newest available version.
To recompile executables, first run 'dart pub global deactivate flutterfire_cli'.
Installed executable flutterfire.
Assets updated.
PS D:\SENAI\2024-2\DESENV\DESENV\app_prep\app_a08> [
```

**Figura 4.22** – Flutterfire ativado.

Com o flutterfire ativado, configuraremos o projeto utilizando o comando flutterfire conforme mostra a figura a seguir.

```
PS D:\SENAI\2024-2\DESM3\DESM3\App_prep\app_a08> flutterfire configure --project=app07a
Package flutterfire_cli is currently active at version 1.0.0.
Decompressing packages...
The flutterfire_cli is already activated at newest available version.
To recompile executables, first run 'dart pub global deactivate flutterfire_cli'.
flutterfire configure --project=app07a
Activated flutterfire_cli 1.0.0.
PS D:\SENAI\2024-2\DESM3\DESM3\App_prep\app_a08>
```

**Figura 4.23** – Configurando o projeto.

Na próxima figura, é possível visualizar a plataforma a que o projeto Firebase se conectará. As opções são mostradas – Android, iOS, macOS, web, Windows –, e a seleção é feita usando backspace para marcar ou desmarcar a escolha. Neste exemplo, a opção escolhida será Android.

```
PS D:\SENAI\2024-2\DESM3\DESM3\App_prep\app_a08> flutterfire configure --project=app07a
Found 9 Firebase projects. Selecting project app07a.
? Which platforms should your configuration support (use arrow keys & space to select)? 
  android
  ios
  macos
  web
  windows
PS D:\SENAI\2024-2\DESM3\DESM3\App_prep\app_a08>
```

**Figura 4.24** – Escolhendo a plataforma a que o projeto Firebase vai se conectar.

O próximo passo será escolher o projeto que se deseja configurar e realizar a conexão com o Firebase utilizando o applicationId configurado no organization do projeto, conforme mostra a próxima figura.

```
PS D:\SENAI\2024-2\DESM3\DESM3\App_prep\app_a08> flutterfire configure --project=app07a
1 Found 9 Firebase projects. Selecting project app07a.
? Which platforms should your configuration support (use arrow keys & space to select)? . android
? Which Android application id (or package name) do you want to use for this configuration, e.g. 'com.example.app'? > com.prep08
PS D:\SENAI\2024-2\DESM3\DESM3\App_prep\app_a08>
```

**Figura 4.25** – Realizando a conexão do Firebase com o projeto Flutter.

Na figura a seguir, é possível visualizar o projeto Flutter conectado ao Firebase.

```
Firebase configuration file lib.firebaseio_options.dart generated successfully with the following Firebase apps:
Platform  Firebase App Id
android  1:875321299317:android:7b28638067969c17e1833c

Learn more about using this file and next steps from the documentation:
> https://firebase.google.com/docs/flutter/setup
PS D:\SENAI\2024-2\DESM3\DESM3\App_prep\app_a08>
```

**Figura 4.26** – Projeto Flutter conectado ao Firebase.

Com todos os passos realizados anteriormente, podemos prosseguir com a inclusão do código da figura a seguir no projeto Flutter.

Para inicializar o Firebase, chame `Firebase.initializeApp` no pacote `firebase_core` com a configuração do seu novo arquivo `firebase_options.dart`:

```
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

// ...

await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
);
```



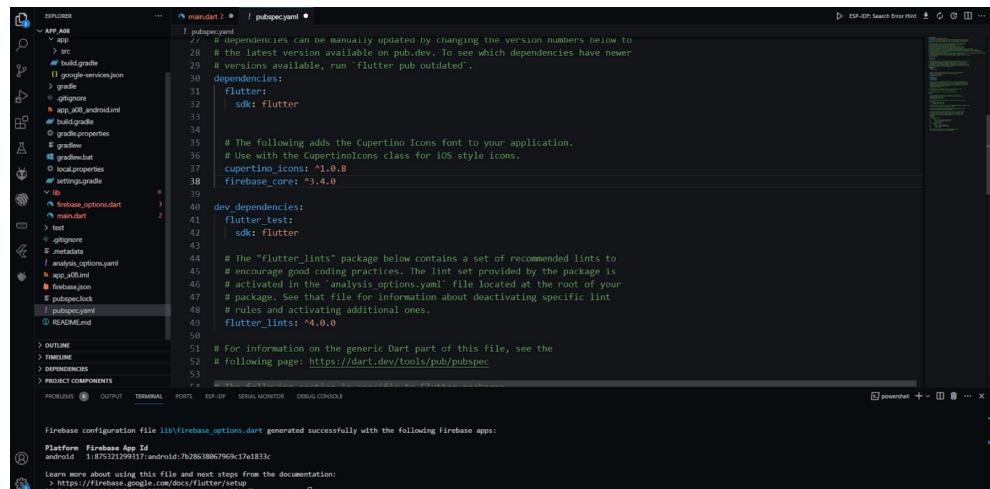
**Figura 4.27** – Código Firebase para incluir no projeto Flutter.

Na próxima figura, é possível visualizar as bibliotecas do Firebase incluídas.

```
1 import 'package:flutter/material.dart';
2 import 'package:firebase_core/firebase_core.dart';
3 import 'firebase_options.dart';
4
5 Run | Debug | Profile
6 void main() async{
7     WidgetsFlutterBinding.ensureInitialized(); // impede que o app de erro ao inicializar
8
9     await Firebase.initializeApp(
10         options: DefaultFirebaseOptions.currentPlatform,
11     );
12     runApp(const MyApp());
13 }
14
15 class MyApp extends StatelessWidget {
16     const MyApp({super.key});
17
18     // This widget is the root of your application.
19     @override
20     Widget build(BuildContext context) {
21         return MaterialApp(
22             title: 'Flutter Demo',
23             theme: ThemeData(
24                 // This is the theme of your application.
25                 // TRY THIS: Try running your application with "flutter run". You'll see
26                 // the application has a purple toolbar. Then, without quitting the app,
```

**Figura 4.28** – Código dart com as bibliotecas incluídas.

Precisamos, agora, ir até pubspec.yaml para adicionar a dependência firebase\_core: ^3.4.0 (ver a figura a seguir).



```

dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  firebase_core: ^3.4.0

dev_dependencies:
  flutter_test:
    sdk: flutter

# The "flutter_lints" package below contains a set of recommended lints to
# encourage good coding practices. The lint set provided by the package is
# activated in the "analysis_options.yaml" file located at the root of your
# package. See that file for information about deactivating specific lint
# rules and activating additional ones.
flutter_lints: ^4.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

```

Firebase configuration file lib/firebase\_options.dart generated successfully with the following Firebase apps:

Platform: Firebase App ID: android:1:97531290517:android:7b2638067960c17e183fc  
Learn more about using this file and next steps from the documentation:  
<https://firebase.google.com/docs/flutter/setup>

**Figura 4.29 – Arquivo pubspec.yaml.**

Na figura a seguir, é possível visualizar o aplicativo conectado ao Firebase.

```

1 import 'package:flutter/material.dart';
2 import 'package:firebase_core/firebase_core.dart';
3 import 'firebase_options.dart';
4
5 Run | Debug | Profile
6 void main() async{
7   WidgetsFlutterBinding.ensureInitialized(); // impede que o app de erro ao inicializar
8
9   await Firebase.initializeApp(
10     options: DefaultFirebaseOptions.currentPlatform,
11   );
12   runApp(const MyApp());
13
14 class MyApp extends StatelessWidget {
15   const MyApp({super.key});
16
17   // This widget is the root of your application.
18   @override
19   Widget build(BuildContext context) {
20     return MaterialApp(
21       title: 'Flutter Demo',
22       theme: ThemeData(
23         // This is the theme of your application.
24         //
25         // TRY THIS: Try running your application with "flutter run". You'll see
26         // the application has a purple toolbar. Then, without quitting the app,

```

**Figura 4.30 – Aplicativo conectado ao Firebase.**

Após conectar o aplicativo ao Firebase, é necessário realizar algumas configurações para poder executá-lo. A primeira delas é ir em Explorer no Vscode-Android -> App -> build.gradle e alterar o applicationId e minSdk de versões, conforme mostra a figura a seguir.



The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar shows the project structure for an "APP A/B" project. It includes files like `build.gradle`, `main.dart`, `pubspec.yaml`, `gradle`, `ignore`, `assets`, `src`, `build.gradle`, `google-services.json`, `gradle`, `gitignore`, `app_bar.xml`, `gradle.properties`, `gradlew`, `gradlew.bat`, `local.properties`, `setting.gradle`, `lib`, `flutter`, `flutter_options.dart`, `main.dart`, `test`, `flutter-plugins`, `flutter-plugins-dependencies`, `gitignore`, `gradle`, `analysis_options.yaml`, `bom.xml`, `outline`, `timeliner`, `dependency`, and `PROJECT COMPONENTS`.
- Code Editor:** The main editor window displays the `build.gradle` file for the `app` module. The code includes configuration for Java version compatibility, Kotlin options, default application ID, min SDK, target SDK, and signing configurations.
- Bottom Bar:** The bottom bar shows tabs for PROBLEMS, OUTPUT, TERMINAL, PORTS, ESP-IDF, SERIAL MONITOR, and DEBUT CONSOLE. The TERMINAL tab is active, showing the command `flutter app a00` and its output.

**Figura 4.31** – Ajuste da versão mínima do SDK e applicationId.

Na figura a seguir, é possível visualizar as configurações de versão e applicationId realizadas.

```
17     sourceCompatibility = JavaVersion.VERSION_1_8
18     targetCompatibility = JavaVersion.VERSION_1_8
19 }
20
21 kotlinOptions {
22     jvmTarget = JavaVersion.VERSION_1_8
23 }
24
25 defaultConfig {
26     // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).
27     applicationId = "com.purpleflame.purpleflame"
28     // You can update the following values to match your application needs.
29     // For more information, see: https://flutter.dev/to-review-gradle-config.
30     minSdk = 21
31     targetSdk = flutter.targetSdkVersion
32     versionCode = flutter.versionCode
33     versionName = flutter.versionName
34 }
35
36 buildTypes {
37     release {
38         // TODO: Add your own signing config for the release build.
39         // Signing with the debug keys for now, so 'flutter run --release' works.
40         signingConfig = signingConfigs.debug
41     }
42 }
43 }
```

**Figura 4.32** – Alteração de versão e applicationId realizada.

Essa configuração é necessária porque, para realizar a conexão, o Firebase exige que a versão mínima do SDK Android seja a 21. Além disso, deve-se utilizar o applicationId configurado no organization no início do projeto. O próximo passo é escolher o Cloudfirestore para armazenar as informações de meses do ano.

Na próxima figura, é possível visualizar a tela dos recursos do Firebase.



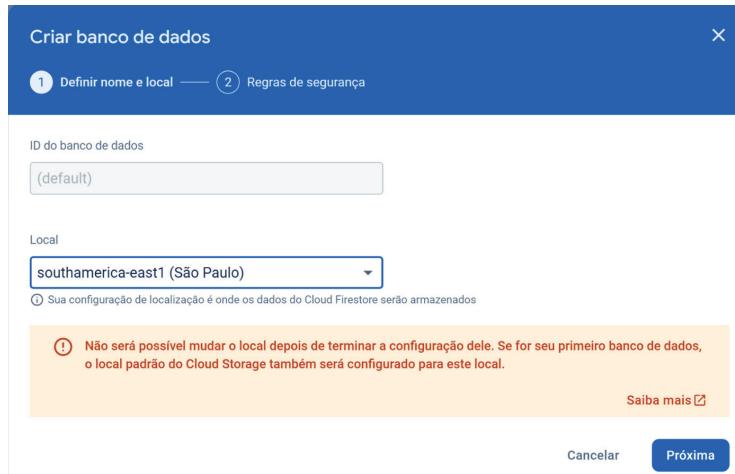
**Figura 4.33 – Recursos do Firebase.**

Nela, clicamos em Criação e selecionamos Firestore Database. Essa ação nos direciona para a tela de criação do banco de dados (ver a próxima figura).



**Figura 4.34 – Recurso Cloud Firestore.**

Após clicar em Criar banco de dados, devemos escolher o local em que este será instanciado, conforme indicado na figura a seguir. No nosso caso, escolheremos southamerica-east1 (São Paulo) por estarmos no Brasil e a latência ser menor do que em outras localidades.



**Figura 4.35 – Escolha da localização do banco de dados instanciado na nuvem.**

Feita a escolha da localização, devemos escolher com qual modo de operação o banco de dados funcionará (ver figura a seguir).

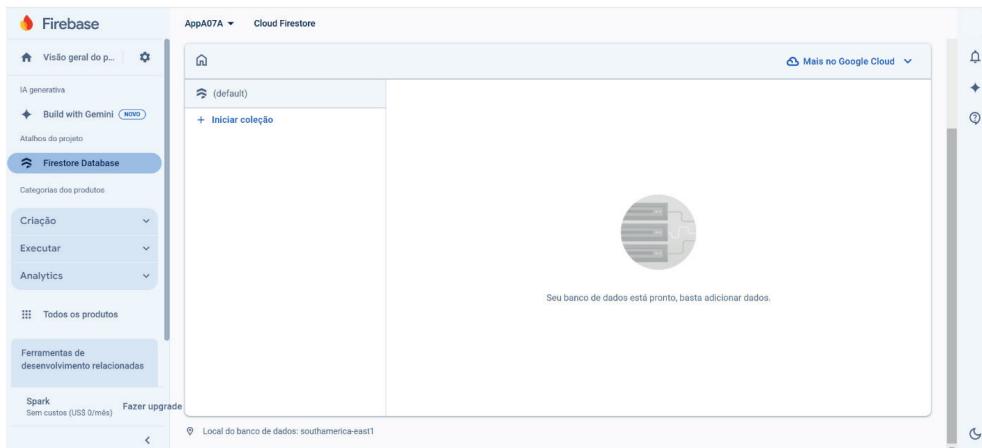


**Figura 4.36 – Escolha do modo de operação do banco de dados.**

No “modo produção”, os dados são salvos constantemente e não expiram; já no “modo de teste”, os dados expiram após trinta dias.

Para efeitos de exemplo, escolheremos o modo teste.

Esses passos finalizam a configuração e criam o banco de dados, conforme a figura a seguir.



**Figura 4.37 – Banco de dados criado no Firestore Database.**

Com o banco de dados criado no Firestore Database, o passo seguinte é adicionar a dependência `cloud_firestore: ^5.4.0` ao projeto, no arquivo `pubspec.yaml`, conforme mostrado a seguir. Depois, executaremos o comando `flutter pub get` para atualizar as dependências do projeto.

#### Arquivo `pubspec.yaml`

```

1. name: app_listacompras
2. description: "A new Flutter project."
3. # The following line prevents the package from being
   accidentally published to
4. # pub.dev using `flutter pub publish`. This is preferred
   for private packages.
5. publish_to: 'none' # Remove this line if you wish to
   publish to pub.dev
6.
  
```

```
7. # The following defines the version and build number for  
your application.  
8. # A version number is three numbers separated by dots,  
like 1.2.43  
9. # followed by an optional build number separated by a +.  
10. # Both the version and the builder number may be  
overridden in flutter  
11. # build by specifying --build-name and --build-number,  
respectively.  
12. # In Android, build-name is used as versionName while  
build-number used as versionCode.  
13. # Read more about Android versioning at https://developer.android.com/studio/publish/versioning  
14. # In iOS, build-name is used as CFBundleShortVersionString  
while build-number is used as CFBundleVersion.  
15. # Read more about iOS versioning at  
16. # https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html  
17. # In Windows, build-name is used as the major, minor, and  
patch parts  
18. # of the product and file versions while build-number is  
used as the build suffix.  
19. version: 1.0.0+1  
20.  
21. environment:  
22.   sdk: ^3.5.1  
23.  
24. # Dependencies specify other packages that your package  
needs in order to work.  
25. # To automatically upgrade your package dependencies to  
the latest versions  
26. # consider running `flutter pub upgrade --major-versions`.  
Alternatively,
```

```
27. # dependencies can be manually updated by changing the
   version numbers below to
28. # the latest version available on pub.dev. To see which
   dependencies have newer
29. # versions available, run `flutter pub outdated`.
30. dependencies:
31.   flutter:
32.     sdk: flutter
33.
34.
35.   # The following adds the Cupertino Icons font to your
      application.
36.   # Use with the CupertinoIcons class for iOS style icons.
37.   cupertino_icons: ^1.0.8
38.   firebase_core: ^3.4.0
39.   cloud_firestore: ^5.4.0
40.   uuid: ^3.0.7
41.   flutter_launcher_icons: ^0.10.0
42.   firebase_auth: ^4.2.9
43.
44. dev_dependencies:
45.   flutter_test:
46.     sdk: flutter
47.
48.   # The “flutter_lints” package below contains a set of
      recommended lints to
49.   # encourage good coding practices. The lint set
      provided by the package is
50.   # activated in the `analysis_options.yaml` file located
      at the root of your
51.   # package. See that file for information about
      deactivating specific lint
52.   # rules and activating additional ones.
53.   flutter_lints: ^4.0.0
54.
```

```
55. # For information on the generic Dart part of this file,  
    see the  
56. # following page: https://dart.dev/tools/pub/pubspec  
57.  
58. # The following section is specific to Flutter packages.  
59. flutter:  
60.  
61.   # The following line ensures that the Material Icons  
       font is  
62.   # included with your application, so that you can use  
       the icons in  
63.   # the material Icons class.  
64.   uses-material-design: true  
65. flutter_icons:  
66.   android: true  
67.   image_path: assets/icon/icon.png  
68.   # To add assets to your application, add an assets  
       section, like this:  
69.   # assets:  
70.   #   - images/a_dot_burr.jpeg  
71.   #   - images/a_dot_ham.jpeg  
72.  
73.   # An image asset can refer to one or more resolution-  
       specific “variants”, see  
74.   # https://flutter.dev/to/resolution-aware-images  
75.  
76.   # For details regarding adding assets from package  
       dependencies, see  
77.   # https://flutter.dev/to/asset-from-package  
78.  
79.   # To add custom fonts to your application, add a fonts  
       section here,  
80.   # in this “flutter” section. Each entry in this list  
       should have a  
81.   # “family” key with the font family name, and a “fonts”  
       key with a
```

```

82. # list giving the asset and other descriptors for the
     font. For
83. # example:
84. # fonts:
85. #   - family: Schyler
86. #     fonts:
87. #       - asset: fonts/Schyler-Regular.ttf
88. #       - asset: fonts/Schyler-Italic.ttf
89. #         style: italic
90. #   - family: Trajan Pro
91. #     fonts:
92. #       - asset: fonts/TrajanPro.ttf
93. #       - asset: fonts/TrajanPro_Bold.ttf
94. #         weight: 700
95. #
96. # For details regarding fonts from package
     dependencies,
97. # see https://flutter.dev/to/font-from-package
98.

```

A seguir, é possível visualizar o código do arquivo models.dart em que a estrutura (classe) para salvar as informações no banco de dados é criada.

#### Código do arquivo models.dart

```

1. class Lista{
2.   String id;
3.   String nome;
4.   Lista({required this.id, required this.nome});
5.   // cria o map para envio de informações para o firebase
6.   Lista.fromMap(Map<String,dynamic>map):id=map[“id”],
7.   nome = map[“nome”];
8.
9.   Map<String,dynamic> toMap(){
10.    return{
11.      “id”: id,

```

```
12.      “nome”:nome,  
13.  
14.    };  
15.  }  
16.}  
17.
```

Finalmente, a seguir, apresentamos o código completo do aplicativo, incluindo a inserção, a leitura e a exclusão de informações do banco de dados (CRUD).

#### Código completo do aplicativo realizando o CRUD no banco de dados Firebase

```
1. import ‘package:app_listacompras/auth.dart’;  
2. import ‘package:flutter/material.dart’;  
3. import ‘package:firebase_core/firebase_core.dart’;  
4. import ‘firebase_options.dart’;  
5. import ‘package:cloud_firestore/cloud_firestore.dart’;  
6. import ‘package:uuid/uuid.dart’; // gera os uuid para os documentos de cada post  
7. import ‘package:app_listacompras/models/models.dart’;  
8. // com.ap08  
9. void main() async{  
10.   // evita que o app trave a tela antes de inicializar o banco de dados  
11.   WidgetsFlutterBinding.ensureInitialized();  
12.   await Firebase.initializeApp(  
13.     options: DefaultFirebaseOptions.currentPlatform,  
14.   );  
15.   //FirebaseFirestore firestore = FirebaseFirestore.instance;  
16.   // cria uma instancia do banco de dados  
17.   //firestore.collection(‘Lista de compras’).doc(Uuid().v1()).  
18.   set({  
19.     // “mês”:“setembro”,  
20.     // “nome”:“carne”,
```

```
19. // "qtde": 10,
20. // "preço":50
21. //});
22. runApp(MaterialApp(
23.   home: HomeScreen(),
24. ));
25. }
26.
27. class HomeScreen extends StatefulWidget {
28.   const HomeScreen({super.key});
29.
30.   @override
31.   State<HomeScreen> createState() => _HomeScreenState();
32. }
33.
34. class _HomeScreenState extends State<HomeScreen> {
35.   // instancia o firebase
36.   FirebaseFirestore firestore = FirebaseFirestore.
37.     instance;
37.   List<Lista> listLista=[]; // lista para armazenar os
38.   dados enviados ao firebase
38.   _refresh()async{
39.     List<Lista>temp = []; // cria uma variavel vazia
40.     // vai armazenar os dados do firebase em snapshot
41.     QuerySnapshot<Map<String,dynamic>>snapshot = await
42.       firestore.collection("Listacompras").get();
42.     for(var doc in snapshot.docs){
43.       temp.add(Lista.fromMap(doc.data()));
44.     }
45.     setState(() {
46.       listLista = temp;
47.     });
48.   }
49.   // função que irá apagar um item do banco de dados
50.   void remove(Lista model){
```

```
51.      firestore.collection("Listacompras").doc(model.id).
52.          delete();
53.      }
54.  @override
55.  void initState() {
56.      _refresh();
57.      super.initState();
58.  }
59.  @override
60.  Widget build(BuildContext context) {
61.      return Scaffold(
62.          appBar: AppBar(
63.              title: Text("App aula08"),
64.              backgroundColor: Colors.red,
65.          ),
66.          floatingActionButton:
67.              FloatingActionButton(onPressed: (){
68.                  showFormModal();
69.              },
70.              child: Icon(Icons.add),),
71.              body: (listLista.isEmpty)?Center(
72.                  child: Text("Não temos listas salvas \n Vamos
73.                  criar a primeira ?",
74.                  textAlign: TextAlign.center,style:
75.                  TextStyle(fontSize: 18),),
76.                  ):RefreshIndicator(
77.                      // vai chamar uma função que atualiza os itens
78.                      da lista
79.                      onRefresh: (){
80.                          return _refresh();
81.                      },
```

```
81.         child: ListView(
82.             children:List.generate(
83.                 listLista.length,
84.                 (index){
85.                     Lista model = listLista[index];
86.                     // Dismissible comando para deslizar o
elemento da lista
87.                     return Dismissible(
88.                         key: ValueKey<Lista>(model),
89.                         // direção do fim para o começo
90.                         direction: DismissDirection.endToStart,
91.                         background: Container(
92.                             color: Colors.red,
93.                             alignment: Alignment.centerRight,
94.                             child:Padding(
95.                                 padding: EdgeInsets.only(right: 8.0,)

,
96.                                 child: Icon(Icons.delete,color:
Colors.white,),

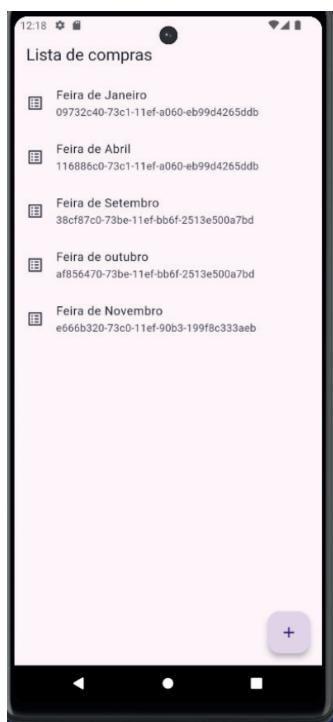
),
97.                         onDismissed:(direction){
98.                             remove(model);
99.                         } ,
100.                     }
101.                     child: ListTile(
102.                         onTap: (){
103.                             _refresh();
104.                         },
105.                         onLongPress: (){
106.                             showFormModal(model:model);
107.                         }
108.                         },
109.                         leading: Icon(Icons.list_alt_rounded),
110.                         title: Text(model.nome),
111.                         subtitle: Text(model.id),
```



```
147.           Text(title, style: Theme.of(context).textTheme.  
bodyLarge,),  
148.           TextFormField(  
149.               controller: nome_list,  
150.               decoration: InputDecoration(  
151.                   label: Text("Nome da lista"),  
152.               ),  
153.               ),  
154.           SizedBox(  
155.               height: 16,  
156.           ),  
157.           Row(  
158.               mainAxisAlignment: MainAxisAlignment.end,  
159.               children: [  
160.                   TextButton(onPressed:(){  
161.                       Navigator.pop(context);  
162.                   } , child: Text(skipButton)),  
163.                   SizedBox(  
164.                       height: 16,  
165.                   ),  
166.                   ElevatedButton(onPressed: (){  
167.                       // cria uma variavel compra do tipo  
lista  
168.                       Lista compra = Lista(id: Uuid().v1(),  
nome: nome_list.text);  
169.                       // no caso de editar uma lista verifica  
se o model não está vazio  
170.                       if(model!=null){  
171.                           compra.id = model.id;  
172.                       }  
173.                       firestore.collection("Listacompras").  
doc(compra.id).set(compra.toMap());  
174.                       _refresh();  
175.                   Navigator.pop(context);  
176.               }, child: Text(confirmButton)),
```

```
177.  
178.           ],  
179.         )  
180.       ],  
181.  
182.     ),  
183.   );  
184. });  
185.  
186.  
187. }  
188. }
```

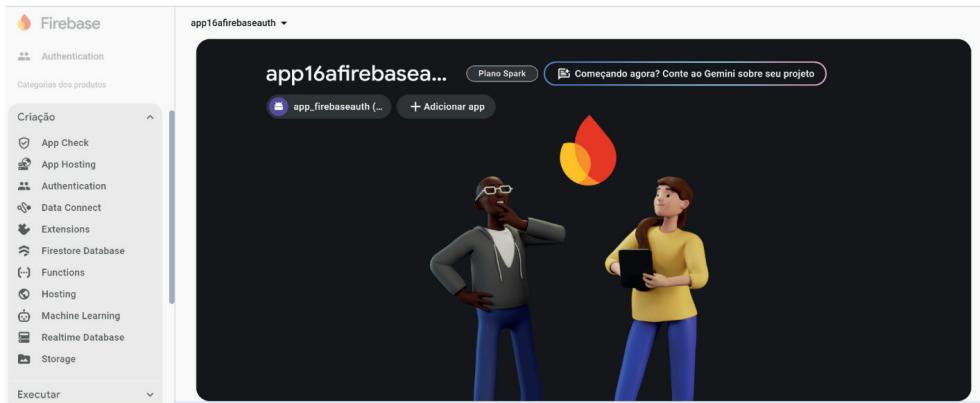
Na figura a seguir, é possível visualizar o aplicativo sendo executado no emulador.



**Figura 4.38 –** Aplicativo sendo executado no emulador.

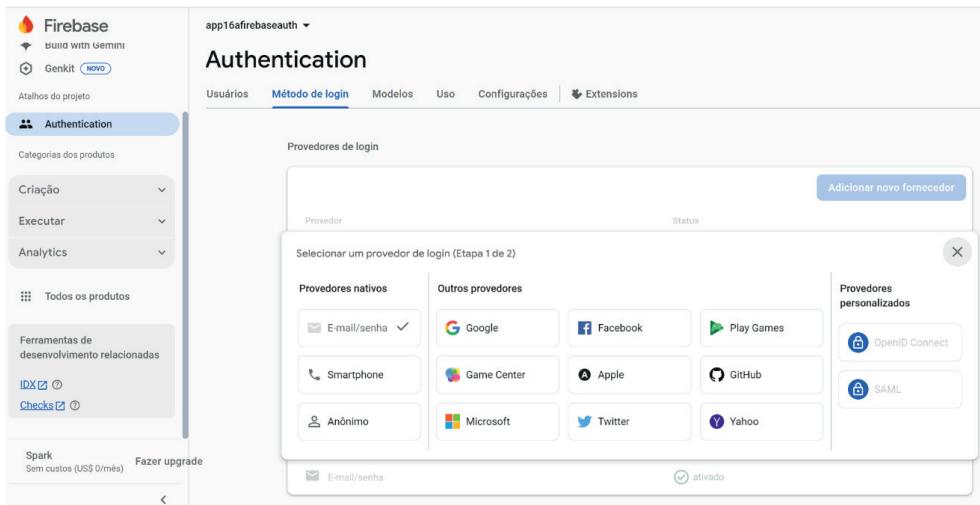
Outro recurso bem interessante do Firebase é o Authentication, que permite realizar a autenticação utilizando e-mail e serviços do Google.

Na próxima figura, é possível visualizar a página do Authentication do Firebase.



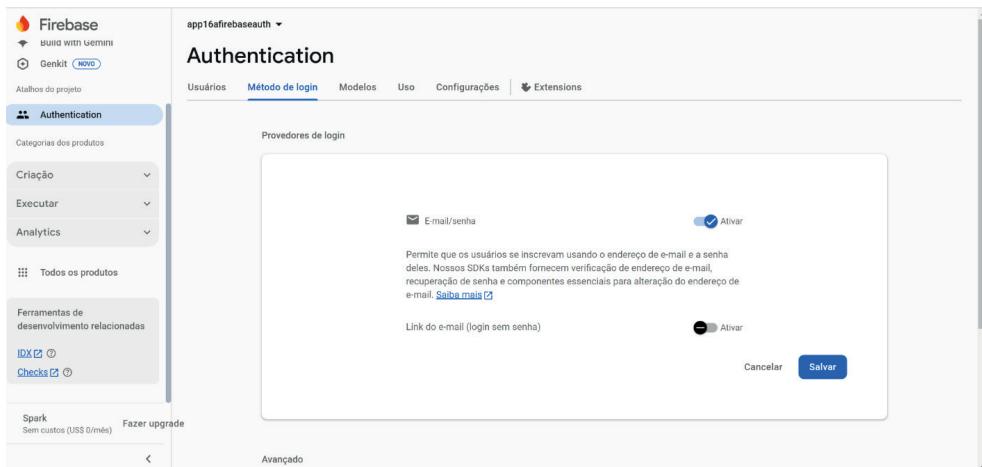
**Figura 4.39 – Authentication do Firebase.**

Após clicar em Authentication, somos direcionados para a tela mostrada na figura a seguir, a qual retrata a configuração do método de login.



**Figura 4.40 – Tela de configuração do método de login.**

Em nosso exemplo, o método escolhido será o e-mail, conforme a figura a seguir.



**Figura 4.41** – Escolha do serviço e-mail para realizar login.

Com o serviço e-mail escolhido, o próximo passo é desenvolver o código Flutter. A seguir, é possível ver o código do arquivo Auth.dart, que contém a classe com os métodos de autenticação, registro e logout utilizando Firebase Auth.

#### Código da classe Auth

```

1. import 'package:firebase_auth/firebase_auth.dart';
2. // Nessa classe auth service é onde os serviços de
3. // autenticação serão implementados
4. class AuthService{
5.
6.   Future<String?>registration({
7.     required String email,
8.     required String password,
9.
10. })async{
11.   try{
12.     await FirebaseAuth.instance.
        createUserWithEmailAndPassword(

```

```
13.         email: email,
14.         password: password,
15.     );
16.     return 'Sucess';
17. }on FirebaseAuthException catch (e) {
18.     if (e.code == 'weak-password') {
19.         return 'The password provided is too weak.';
20.     } else if (e.code == 'email-already-in-use') {
21.         return 'The account already exists for that
22.             email.';
23.     } else {
24.         return e.message;
25.     }
26. } catch (e) {
27.     return e.toString();
28. }
29.
30. Future<String?> login({
31.     required String email,
32.     required String password,
33. }) async {
34.     try {
35.         await FirebaseAuth.instance.
            signInWithEmailAndPassword(
36.             email: email,
37.             password: password,
38.         );
39.         return 'Success';
40.     } on FirebaseAuthException catch (e) {
41.         if (e.code == 'user-not-found') {
42.             return 'No user found for that email.';
43.         } else if (e.code == 'wrong-password') {
44.             return 'Wrong password provided for that user.';
45.         } else {
46.             return e.message;
```

```

47.      }
48.    } catch (e) {
49.      return e.toString();
50.    }
51.  }
52.
53.  // Logout method that calls AuthService().logout() and
54.  // returns the message
55. Future<String?> logout() async {
56.   final message = await AuthService().logout();
57.   return message;
58. }

```

A seguir, apresentamos o código da tela na qual um usuário novo é registrado.

#### Código register.dart

```

1. import 'package:app_authfirebase/screens/app.dart';
2. import 'package:app_authfirebase/services/Auth.dart';
3. import 'package:flutter/material.dart';
4.
5. class Registro extends StatefulWidget {
6.   const Registro({super.key});
7.
8.   @override
9.   State<Registro> createState() => _RegistroState();
10. }
11.
12. class _RegistroState extends State<Registro> {
13.   TextEditingController _email = TextEditingController();
14.   TextEditingController _senha = TextEditingController();
15.   @override
16.   Widget build(BuildContext context) {
17.     return Scaffold(
18.       appBar: AppBar(

```

```
19.         title: Text('Cadastrar usuário'),
20.     ),
21.     body: Center(
22.         child: Column(
23.             mainAxisAlignment: MainAxisAlignment.center,
24.             children: [
25.                 SizedBox(
26.                     width: MediaQuery.of(context).size.width/2,
27.                     child: TextField(
28.                         controller: _email,
29.                         decoration: InputDecoration(hintText:
30.                             'Digite seu email'),
31.                         ),
32.                     ),
33.                     SizedBox(
34.                         width: MediaQuery.of(context).size.width/2,
35.                         child: TextField(
36.                             controller: _senha,
37.                             decoration: InputDecoration(hintText:
38.                                 'Digite sua senha'),
39.                                 obscureText: true,
40.                                 obscuringCharacter: '*',
41.                                 ),
42.                             ),
43.                             Row(
44.                                 mainAxisAlignment: MainAxisAlignment.
45. spaceEvenly,
46.                                 children: [
47.                                     ElevatedButton(onPressed: ()async{
48.                                         print('${_email.text} e ${_senha.text}');
49.                                         final message = await AuthService().
registration(email: _email.text,
50.                                         password: _senha.text);
51.                                         if (message!.contains('Success')) {
52.                                             ScaffoldMessenger.of(context).
showSnackBar(SnackBar(content: Text('Email e Senha
53.                                         validos')));
54.                                         } else {
55.                                             ScaffoldMessenger.of(context).
showSnackBar(SnackBar(content: Text('Email ou Senha
56.                                         invalidos')));
57.                                         }
58.                                         },
59.                                         ),
60.                                         ),
61.                                         ),
62.                                         ),
63.                                         ),
64.                                         ),
65.                                         ),
66.                                         ),
67.                                         ),
68.                                         ),
69.                                         ),
70.                                         ),
71.                                         ),
72.                                         ),
73.                                         ),
74.                                         ),
75.                                         ),
76.                                         ),
77.                                         ),
78.                                         ),
79.                                         ),
80.                                         ),
81.                                         ),
82.                                         ),
83.                                         ),
84.                                         ),
85.                                         ),
86.                                         ),
87.                                         ),
88.                                         ),
89.                                         ),
90.                                         ),
91.                                         ),
92.                                         ),
93.                                         ),
94.                                         ),
95.                                         ),
96.                                         ),
97.                                         ),
98.                                         ),
99.                                         ),
100.                                         ),
101.                                         ),
102.                                         ),
103.                                         ),
104.                                         ),
105.                                         ),
106.                                         ),
107.                                         ),
108.                                         ),
109.                                         ),
110.                                         ),
111.                                         ),
112.                                         ),
113.                                         ),
114.                                         ),
115.                                         ),
116.                                         ),
117.                                         ),
118.                                         ),
119.                                         ),
120.                                         ),
121.                                         ),
122.                                         ),
123.                                         ),
124.                                         ),
125.                                         ),
126.                                         ),
127.                                         ),
128.                                         ),
129.                                         ),
130.                                         ),
131.                                         ),
132.                                         ),
133.                                         ),
134.                                         ),
135.                                         ),
136.                                         ),
137.                                         ),
138.                                         ),
139.                                         ),
140.                                         ),
141.                                         ),
142.                                         ),
143.                                         ),
144.                                         ),
145.                                         ),
146.                                         ),
147.                                         ),
148.                                         ),
149.                                         ),
150.                                         ),
151.                                         ),
152.                                         ),
153.                                         ),
154.                                         ),
155.                                         ),
156.                                         ),
157.                                         ),
158.                                         ),
159.                                         ),
160.                                         ),
161.                                         ),
162.                                         ),
163.                                         ),
164.                                         ),
165.                                         ),
166.                                         ),
167.                                         ),
168.                                         ),
169.                                         ),
170.                                         ),
171.                                         ),
172.                                         ),
173.                                         ),
174.                                         ),
175.                                         ),
176.                                         ),
177.                                         ),
178.                                         ),
179.                                         ),
180.                                         ),
181.                                         ),
182.                                         ),
183.                                         ),
184.                                         ),
185.                                         ),
186.                                         ),
187.                                         ),
188.                                         ),
189.                                         ),
190.                                         ),
191.                                         ),
192.                                         ),
193.                                         ),
194.                                         ),
195.                                         ),
196.                                         ),
197.                                         ),
198.                                         ),
199.                                         ),
200.                                         ),
201.                                         ),
202.                                         ),
203.                                         ),
204.                                         ),
205.                                         ),
206.                                         ),
207.                                         ),
208.                                         ),
209.                                         ),
210.                                         ),
211.                                         ),
212.                                         ),
213.                                         ),
214.                                         ),
215.                                         ),
216.                                         ),
217.                                         ),
218.                                         ),
219.                                         ),
220.                                         ),
221.                                         ),
222.                                         ),
223.                                         ),
224.                                         ),
225.                                         ),
226.                                         ),
227.                                         ),
228.                                         ),
229.                                         ),
230.                                         ),
231.                                         ),
232.                                         ),
233.                                         ),
234.                                         ),
235.                                         ),
236.                                         ),
237.                                         ),
238.                                         ),
239.                                         ),
240.                                         ),
241.                                         ),
242.                                         ),
243.                                         ),
244.                                         ),
245.                                         ),
246.                                         ),
247.                                         ),
248.                                         ),
249.                                         ),
250.                                         ),
251.                                         ),
252.                                         ),
253.                                         ),
254.                                         ),
255.                                         ),
256.                                         ),
257.                                         ),
258.                                         ),
259.                                         ),
260.                                         ),
261.                                         ),
262.                                         ),
263.                                         ),
264.                                         ),
265.                                         ),
266.                                         ),
267.                                         ),
268.                                         ),
269.                                         ),
270.                                         ),
271.                                         ),
272.                                         ),
273.                                         ),
274.                                         ),
275.                                         ),
276.                                         ),
277.                                         ),
278.                                         ),
279.                                         ),
280.                                         ),
281.                                         ),
282.                                         ),
283.                                         ),
284.                                         ),
285.                                         ),
286.                                         ),
287.                                         ),
288.                                         ),
289.                                         ),
290.                                         ),
291.                                         ),
292.                                         ),
293.                                         ),
294.                                         ),
295.                                         ),
296.                                         ),
297.                                         ),
298.                                         ),
299.                                         ),
300.                                         ),
301.                                         ),
302.                                         ),
303.                                         ),
304.                                         ),
305.                                         ),
306.                                         ),
307.                                         ),
308.                                         ),
309.                                         ),
310.                                         ),
311.                                         ),
312.                                         ),
313.                                         ),
314.                                         ),
315.                                         ),
316.                                         ),
317.                                         ),
318.                                         ),
319.                                         ),
320.                                         ),
321.                                         ),
322.                                         ),
323.                                         ),
324.                                         ),
325.                                         ),
326.                                         ),
327.                                         ),
328.                                         ),
329.                                         ),
330.                                         ),
331.                                         ),
332.                                         ),
333.                                         ),
334.                                         ),
335.                                         ),
336.                                         ),
337.                                         ),
338.                                         ),
339.                                         ),
340.                                         ),
341.                                         ),
342.                                         ),
343.                                         ),
344.                                         ),
345.                                         ),
346.                                         ),
347.                                         ),
348.                                         ),
349.                                         ),
350.                                         ),
351.                                         ),
352.                                         ),
353.                                         ),
354.                                         ),
355.                                         ),
356.                                         ),
357.                                         ),
358.                                         ),
359.                                         ),
360.                                         ),
361.                                         ),
362.                                         ),
363.                                         ),
364.                                         ),
365.                                         ),
366.                                         ),
367.                                         ),
368.                                         ),
369.                                         ),
370.                                         ),
371.                                         ),
372.                                         ),
373.                                         ),
374.                                         ),
375.                                         ),
376.                                         ),
377.                                         ),
378.                                         ),
379.                                         ),
380.                                         ),
381.                                         ),
382.                                         ),
383.                                         ),
384.                                         ),
385.                                         ),
386.                                         ),
387.                                         ),
388.                                         ),
389.                                         ),
390.                                         ),
391.                                         ),
392.                                         ),
393.                                         ),
394.                                         ),
395.                                         ),
396.                                         ),
397.                                         ),
398.                                         ),
399.                                         ),
400.                                         ),
401.                                         ),
402.                                         ),
403.                                         ),
404.                                         ),
405.                                         ),
406.                                         ),
407.                                         ),
408.                                         ),
409.                                         ),
410.                                         ),
411.                                         ),
412.                                         ),
413.                                         ),
414.                                         ),
415.                                         ),
416.                                         ),
417.                                         ),
418.                                         ),
419.                                         ),
420.                                         ),
421.                                         ),
422.                                         ),
423.                                         ),
424.                                         ),
425.                                         ),
426.                                         ),
427.                                         ),
428.                                         ),
429.                                         ),
430.                                         ),
431.                                         ),
432.                                         ),
433.                                         ),
434.                                         ),
435.                                         ),
436.                                         ),
437.                                         ),
438.                                         ),
439.                                         ),
440.                                         ),
441.                                         ),
442.                                         ),
443.                                         ),
444.                                         ),
445.                                         ),
446.                                         ),
447.                                         ),
448.                                         ),
449.                                         ),
450.                                         ),
451.                                         ),
452.                                         ),
453.                                         ),
454.                                         ),
455.                                         ),
456.                                         ),
457.                                         ),
458.                                         ),
459.                                         ),
460.                                         ),
461.                                         ),
462.                                         ),
463.                                         ),
464.                                         ),
465.                                         ),
466.                                         ),
467.                                         ),
468.                                         ),
469.                                         ),
470.                                         ),
471.                                         ),
472.                                         ),
473.                                         ),
474.                                         ),
475.                                         ),
476.                                         ),
477.                                         ),
478.                                         ),
479.                                         ),
480.                                         ),
481.                                         ),
482.                                         ),
483.                                         ),
484.                                         ),
485.                                         ),
486.                                         ),
487.                                         ),
488.                                         ),
489.                                         ),
490.                                         ),
491.                                         ),
492.                                         ),
493.                                         ),
494.                                         ),
495.                                         ),
496.                                         ),
497.                                         ),
498.                                         ),
499.                                         ),
500.                                         ),
501.                                         ),
502.                                         ),
503.                                         ),
504.                                         ),
505.                                         ),
506.                                         ),
507.                                         ),
508.                                         ),
509.                                         ),
510.                                         ),
511.                                         ),
512.                                         ),
513.                                         ),
514.                                         ),
515.                                         ),
516.                                         ),
517.                                         ),
518.                                         ),
519.                                         ),
520.                                         ),
521.                                         ),
522.                                         ),
523.                                         ),
524.                                         ),
525.                                         ),
526.                                         ),
527.                                         ),
528.                                         ),
529.                                         ),
530.                                         ),
531.                                         ),
532.                                         ),
533.                                         ),
534.                                         ),
535.                                         ),
536.                                         ),
537.                                         ),
538.                                         ),
539.                                         ),
540.                                         ),
541.                                         ),
542.                                         ),
543.                                         ),
544.                                         ),
545.                                         ),
546.                                         ),
547.                                         ),
548.                                         ),
549.                                         ),
550.                                         ),
551.                                         ),
552.                                         ),
553.                                         ),
554.                                         ),
555.                                         ),
556.                                         ),
557.                                         ),
558.                                         ),
559.                                         ),
560.                                         ),
561.                                         ),
562.                                         ),
563.                                         ),
564.                                         ),
565.                                         ),
566.                                         ),
567.                                         ),
568.                                         ),
569.                                         ),
570.                                         ),
571.                                         ),
572.                                         ),
573.                                         ),
574.                                         ),
575.                                         ),
576.                                         ),
577.                                         ),
578.                                         ),
579.                                         ),
580.                                         ),
581.                                         ),
582.                                         ),
583.                                         ),
584.                                         ),
585.                                         ),
586.                                         ),
587.                                         ),
588.                                         ),
589.                                         ),
590.                                         ),
591.                                         ),
592.                                         ),
593.                                         ),
594.                                         ),
595.                                         ),
596.                                         ),
597.                                         ),
598.                                         ),
599.                                         ),
600.                                         ),
601.                                         ),
602.                                         ),
603.                                         ),
604.                                         ),
605.                                         ),
606.                                         ),
607.                                         ),
608.                                         ),
609.                                         ),
610.                                         ),
611.                                         ),
612.                                         ),
613.                                         ),
614.                                         ),
615.                                         ),
616.                                         ),
617.                                         ),
618.                                         ),
619.                                         ),
620.                                         ),
621.                                         ),
622.                                         ),
623.                                         ),
624.                                         ),
625.                                         ),
626.                                         ),
627.                                         ),
628.                                         ),
629.                                         ),
630.                                         ),
631.                                         ),
632.                                         ),
633.                                         ),
634.                                         ),
635.                                         ),
636.                                         ),
637.                                         ),
638.                                         ),
639.                                         ),
640.                                         ),
641.                                         ),
642.                                         ),
643.                                         ),
644.                                         ),
645.                                         ),
646.                                         ),
647.                                         ),
648.                                         ),
649.                                         ),
650.                                         ),
651.                                         ),
652.                                         ),
653.                                         ),
654.                                         ),
655.                                         ),
656.                                         ),
657.                                         ),
658.                                         ),
659.                                         ),
660.                                         ),
661.                                         ),
662.                                         ),
663.                                         ),
664.                                         ),
665.                                         ),
666.                                         ),
667.                                         ),
668.                                         ),
669.                                         ),
670.                                         ),
671.                                         ),
672.                                         ),
673.                                         ),
674.                                         ),
675.                                         ),
676.                                         ),
677.                                         ),
678.                                         ),
679.                                         ),
680.                                         ),
681.                                         ),
682.                                         ),
683.                                         ),
684.                                         ),
685.                                         ),
686.                                         ),
687.                                         ),
688.                                         ),
689.                                         ),
690.                                         ),
691.                                         ),
692.                                         ),
693.                                         ),
694.                                         ),
695.                                         ),
696.                                         ),
697.                                         ),
698.                                         ),
699.                                         ),
700.                                         ),
701.                                         ),
702.                                         ),
703.                                         ),
704.                                         ),
705.                                         ),
706.                                         ),
707.                                         ),
708.                                         ),
709.                                         ),
710.                                         ),
711.                                         ),
712.                                         ),
713.                                         ),
714.                                         ),
715.                                         ),
716.                                         ),
717.                                         ),
718.                                         ),
719.                                         ),
720.                                         ),
721.                                         ),
722.                                         ),
723.                                         ),
724.                                         ),
725.                                         ),
726.                                         ),
727.                                         ),
728.                                         ),
729.                                         ),
730.                                         ),
731.                                         ),
732.                                         ),
733.                                         ),
734.                                         ),
735.                                         ),
736.                                         ),
737.                                         ),
738.                                         ),
739.                                         ),
740.                                         ),
741.                                         ),
742.                                         ),
743.                                         ),
744.                                         ),
745.                                         ),
746.                                         ),
747.                                         ),
748.                                         ),
749.                                         ),
750.                                         ),
751.                                         ),
752.                                         ),
753.                                         ),
754.                                         ),
755.                                         ),
756.                                         ),
757.                                         ),
758.                                         ),
759.                                         ),
760.                                         ),
761.                                         ),
762.                                         ),
763.                                         ),
764.                                         ),
765.                                         ),
766.                                         ),
767.                                         ),
768.                                         ),
769.                                         ),
770.                                         ),
771.                                         ),
772.                                         ),
773.                                         ),
774.                                         ),
775.                                         ),
776.                                         ),
777.                                         ),
778.                                         ),
779.                                         ),
780.                                         ),
781.                                         ),
782.                                         ),
783.                                         ),
784.                                         ),
785.                                         ),
786.                                         ),
787.                                         ),
788.                                         ),
789.                                         ),
790.                                         ),
791.                                         ),
792.                                         ),
793.                                         ),
794.                                         ),
795.                                         ),
796.                                         ),
797.                                         ),
798.                                         ),
799.                                         ),
800.                                         ),
801.                                         ),
802.                                         ),
803.                                         ),
804.                                         ),
805.                                         ),
806.                                         ),
807.                                         ),
808.                                         ),
809.                                         ),
810.                                         ),
811.                                         ),
812.                                         ),
813.                                         ),
814.                                         ),
815.                                         ),
816.                                         ),
817.                                         ),
818.                                         ),
819.                                         ),
820.                                         ),
821.                                         ),
822.                                         ),
823.                                         ),
824.                                         ),
825.                                         ),
826.                                         ),
827.                                         ),
828.                                         ),
829.                                         ),
830.                                         ),
831.                                         ),
832.                                         ),
833.                                         ),
834.                                         ),
835.                                         ),
836.                                         ),
837.                                         ),
838.                                         ),
839.                                         ),
840.                                         ),
841.                                         ),
842.                                         ),
843.                                         ),
844.                                         ),
845.                                         ),
846.                                         ),
847.                                         ),
848.                                         ),
849.                                         ),
850.                                         ),
851.                                         ),
852.                                         ),
853.                                         ),
854.                                         ),
855.                                         ),
856.                                         ),
857.                                         ),
858.                                         ),
859.                                         ),
860.                                         ),
861.                                         ),
862.                                         ),
863.                                         ),
864.                                         ),
865.                                         ),
866.                                         ),
867.                                         ),
868.                                         ),
869.                                         ),
870.                                         ),
871.                                         ),
872.                                         ),
873.                                         ),
874.                                         ),
875.                                         ),
876.                                         ),
877.                                         ),
878.                                         ),
879.                                         ),
880.                                         ),
881.                                         ),
882.                                         ),
883.                                         ),
884.                                         ),
885.                                         ),
886.                                         ),
887.                                         ),
888.                                         ),
889.                                         ),
890.                                         ),
891.                                         ),
892.                                         ),
893.                                         ),
894.                                         ),
895.                                         ),
896.                                         ),
897.                                         ),
898.                                         ),
899.                                         ),
900.                                         ),
901.                                         ),
902.                                         ),
903.                                         ),
904.                                         ),
905.                                         ),
906.                                         ),
907.                                         ),
908.                                         ),
909.                                         ),
910.                                         ),
911.                                         ),
912.                                         ),
913.                                         ),
914.                                         ),
915.                                         ),
916.                                         ),
917.                                         ),
918.                                         ),
919.                                         ),
920.                                         ),
921.                                         ),
922.                                         ),
923.                                         ),
924.                                         ),
925.                                         ),
926.                                         ),
927.                                         ),
928.                                         ),
929.                                         ),
930.                                         ),
931.                                         ),
932.                                         ),
933.                                         ),
934.                                         ),
935.                                         ),
936.                                         ),
937.                                         ),
938.                                         ),
939.                                         ),
940.                                         ),
941.                                         ),
942.                                         ),
943.                                         ),
944.                                         ),
945.                                         ),
946.                                         ),
947.                                         ),
948.                                         ),
949.                                         ),
950.                                         ),
951.                                         ),
952.                                         ),
953.                                         ),
954.                                         ),
955.                                         ),
956.                                         ),
957.                                         ),
958.                                         ),
959.                                         ),
960.                                         ),
961.                                         ),
962.                                         ),
963.                                         ),
964.                                         ),
965.                                         ),
966.                                         ),
967.                                         ),
968.                                         ),
969.                                         ),
970.                                         ),
971.                                         ),
972.                                         ),
973.                                         ),
974.                                         ),
975.                                         ),
976.                                         ),
977.                                         ),
978.                                         ),
979.                                         ),
980.                                         ),
981.                                         ),
982.                                         ),
983.                                         ),
984.                                         ),
985.                                         ),
986.                                         ),
987.                                         ),
988.                                         ),
989.                                         ),
990.                                         ),
991.                                         ),
992.                                         ),
993.                                         ),
994.                                         ),
995.                                         ),
996.                                         ),
997.                                         ),
998.                                         ),
999.                                         ),
1000.                                         ),
```

```
50.           Navigator.of(context).pushReplacement(
51.               MaterialPageRoute(builder: (context)
52.                   => TelaApp())));
53.           }
54.           ScaffoldMessenger.of(context).showSnackBar(
55.               SnackBar(
56.                   content: Text(message),
57.               ),
58.           );
59.
60.           }, child: Text('Cadastrar')),
61.           ElevatedButton(onPressed: (){
62.               _email.text="";
63.               _senha.text="";
64.               }, child: Text('Limpar')),
65.               ],
66.           ),
67.
68.           ],
69.           ),
70.           ),
71.       );
72.   }
73. }
74.
```

A seguir, é possível visualizar o código da tela para realizar login.

#### Código login.dart

```
1. import 'package:app_authfirebase/screens/app.dart';
2. import 'package:app_authfirebase/screens/register.dart';
3. import 'package:app_authfirebase/services/Auth.dart';
4. import 'package:flutter/material.dart';
5.
```

```
6. class Login extends StatefulWidget {
7.   const Login({super.key});
8.
9.   @override
10.  State<Login> createState() => _LoginState();
11. }
12.
13. class _LoginState extends State<Login> {
14.   TextEditingController email = TextEditingController();
15.   TextEditingController senha = TextEditingController();
16.   @override
17.   Widget build(BuildContext context) {
18.     return Scaffold(
19.       appBar: AppBar(
20.         title: Text('Login'),
21.       ),
22.       body: Center(
23.         child: Column(
24.           mainAxisAlignment: MainAxisAlignment.center,
25.           children: [
26.             SizedBox(
27.               width: MediaQuery.of(context).size.width/2,
28.               child: TextField(
29.                 controller: email,
30.                 decoration: InputDecoration(hintText:
31.                   'Digite seu email'),
32.               ),
33.             SizedBox(
34.               width: MediaQuery.of(context).size.width/2,
35.               child: TextField(
36.                 controller: senha,
37.                 decoration: InputDecoration(hintText:
38.                   'Digite sua senha'),
39.                 obscureText: true,
```

```
39.                 obscuringCharacter: '*',  
40.             ),  
41.         ),  
42.     Row(  
43.         mainAxisAlignment: MainAxisAlignment.  
spaceEvenly,  
44.         children: [  
45.             ElevatedButton(onPressed: ()async{  
46.                 print('${email.text} e ${senha.text}');  
47.                 final message = await AuthService().  
login(email: email.text, password: senha.text);  
48.  
49.                 if (message!.contains('Success')) {  
50.                     Navigator.of(context).  
pushReplacement(  
51.                         MaterialPageRoute(  
52.                             builder: (context) =>  
TelaApp(),  
53.                         ),  
54.                     );  
55.                 }  
56.             ScaffoldMessenger.of(context).  
showSnackBar(  
57.                 SnackBar(  
58.                     content: Text(message),  
59.                     ),  
60.                 );  
61.  
62.  
63.  
64.             }, child: Text('Login')),  
65.             ElevatedButton(onPressed: (){  
66.                 email.text='';  
67.                 senha.text='';  
68.             }, child: Text('Limpar')),
```

```

69.          ],
70.          ),
71.          TextButton(onPressed: (){
72.            Navigator.push(context,
73.              MaterialPageRoute(builder:
74.                (context)=>Registro())));
75.            }, child: Text('Novo por aqui ! Cadastre-se'))
76.          ],
77.          ),
78.        );
79.      }
80.  }

```

Por fim, a seguir vemos o código da tela do aplicativo app.dart.

#### Código app.dart

```

1. import 'package:app_authfirebase/screens/login.dart';
2. import 'package:flutter/material.dart';
3. import 'package:firebase_core/firebase_core.dart';
4. import 'firebase_options.dart';
5. // com.app16
6. void main() async{
7.   // evita que o app fique travado enquanto nao conecta no
    firebase
8.   WidgetsFlutterBinding.ensureInitialized();
9.   await Firebase.initializeApp(
10.     options: DefaultFirebaseOptions.currentPlatform,
11.   );
12.   runApp(Home());
13. }
14.
15. class Home extends StatelessWidget {
16.   const Home({super.key});
17.

```

```
18. @override
19. Widget build(BuildContext context) {
20.   return MaterialApp(
21.     home: Login(),
22.   );
23. }
24. }
25.
```

Esse foi o passo a passo da programação de um aplicativo utilizando o recurso Authentication do Firebase.

## Persistência local x Persistência nuvem

A persistência de dados na nuvem oferece vantagens como acesso de qualquer lugar e sincronização entre dispositivos, sendo ideal para aplicativos que exigem conectividade. No entanto, a persistência local é crucial em situações de conectividade limitada, garantindo acesso a informações off-line e oferecendo maior desempenho. Muitos aplicativos modernos adotam uma abordagem híbrida, combinando armazenamento local para acesso rápido com sincronização na nuvem para backup e acesso multiplataforma. Essa estratégia une o melhor dos dois mundos, garantindo dados acessíveis, seguros e uma experiência de usuário confiável, mesmo sem internet. A seguir veremos as principais vantagens e desvantagens de cada tipo de persistência de dados e quando utilizar cada uma.

### Persistência local

#### Vantagens:

- **Acesso offline** – os dados ficam disponíveis mesmo sem conexão com a internet, garantindo que o aplicativo funcione em qualquer lugar;

- **Alto desempenho** – a acesso aos dados é rápido, já que eles estão armazenados no dispositivo, reduzindo a latência;
- **Privacidade** – por não depender de servidores externos, os dados permanecem no dispositivo, o que pode oferecer maior controle e segurança em certas situações.

#### **Desvantagens:**

- **Limitações de armazenamento** – depende do espaço disponível no dispositivo do usuário;
- **Sincronização** – não permite acesso de múltiplos dispositivos ou backup automático sem integração adicional;
- **Risco de perda de dados** – se o dispositivo for perdido ou danificado, os dados podem ser irrecuperáveis, a menos que haja backup.

#### **Quando usar:**

- Aplicativos que precisam funcionar off-line, como listas de compras, gerenciadores de tarefas e rastreadores de saúde;
- Jogos móveis que armazenam progresso localmente;
- Aplicações voltadas para privacidade, em que os dados não devem ser enviados para servidores externos.

## Persistência na nuvem

#### **Vantagens:**

- **Acesso multiplataforma** – os dados podem ser acessados de qualquer dispositivo conectado à internet;
- **Backup automático** – reduz o risco de perda de dados, já que eles estão armazenados em servidores remotos;

- **Colaboração** – permite que vários usuários accessem e atualizem os mesmos dados em tempo real.

#### **Desvantagens:**

- **Dependência de conexão** – sem internet, os dados podem não estar acessíveis;
- **Latência** – o acesso aos dados pode ser mais lento, dependendo da qualidade da conexão;
- **Custo** – hospedagem na nuvem pode gerar custos recorrentes, dependendo do provedor.

#### **Quando usar:**

- Aplicativos que precisam de sincronização entre dispositivos, como serviços de streaming, aplicativos financeiros e redes sociais;
- Aplicações colaborativas, como editores de documentos ou gerenciamento de projetos;
- Sistemas que exigem armazenamento seguro e backup frequente.

## **Abordagem híbrida**

Muitos aplicativos modernos combinam as duas abordagens, adotando a **persistência local** para garantir acesso rápido e off-line e a **persistência na nuvem** para sincronização e backup. Por exemplo, um aplicativo de notas pode salvar as informações localmente para acesso imediato e sincronizar os dados na nuvem quando o dispositivo estiver conectado à internet. Essa abordagem é ideal para fornecer uma experiência confiável ao usuário, maximizando o desempenho e a acessibilidade, enquanto reduz os riscos de perda de dados.

## Atividade

A escola EduTec precisa de um aplicativo para gerenciar informações dos alunos de forma centralizada. O aplicativo deve:

- Permitir que professores façam login;
- Oferecer funcionalidades para cadastrar alunos com informações como:
  - Nome do aluno;
  - RA (Registro Acadêmico);
  - Notas;
  - Disciplinas matriculadas.
- Exibir a lista de alunos cadastrados e permitir edição ou exclusão de registros.

Os recursos necessários são:

- Computadores com IDE configurada (VS Code, Android Studio);
- Flutter e Firebase instalados e configurados;
- Conexão com a internet.

As etapas da situação de aprendizagem estão descritas a seguir.

### 1. Introdução ao Firebase e Flutter

- **Objetivo:** explicar o funcionamento do Firebase (Autenticação e Firestore) e como configurá-lo em um aplicativo Flutter.
- **Dica:** consultar a seção “Firebase” deste livro, na qual é mostrado o passo a passo para configurar esse recurso.
- **Atividade:**
  - Criar um projeto Firebase;
  - Configurar o arquivo google-services.json ou GoogleService-Info.plist;
  - Exemplificar a instalação de dependências como firebase\_auth e cloud\_firestore.

### 2. Implementação do login

- **Objetivo:** ensinar como criar um sistema de login para professores.

**• Atividade:**

- Criar uma interface para login (e-mail/senha);
- Implementar a funcionalidade com Firebase Authentication;
- Registrar novos usuários;
- Fazer login;
- Gerenciar erros (ex.: senha inválida, e-mail não cadastrado).

**3. Tela de cadastro de alunos**

- **Objetivo:** ensinar como salvar dados no Firebase Firestore.

**• Atividade:**

- Criar uma interface para inserir os dados do aluno:
  - Nome;
  - RA;
  - Disciplinas (uma lista de disciplinas);
  - Nota (um valor numérico).
- Implementar a funcionalidade para salvar os dados no Firestore.

Código exemplo para salvar os dados no Firestore.

```
1. void saveStudent(String name, String
       ra, double grade, List<String> subjects)
       async {
2.   CollectionReference students
       = FirebaseFirestore.instance.
       collection('students');
3.   await students.add({
4.     'name': name,
5.     'ra': ra,
6.     'grade': grade,
7.     'subjects': subjects,
8.   });
9. }
10.
```

#### 4. Tela de listagem de alunos

- **Objetivo:** ensinar como recuperar e exibir dados do Firestore.
- **Atividade:**
  - Criar uma interface para exibir a lista de alunos em um ListView;
  - Implementar a recuperação de dados;
  - Consultar os documentos no Firestore;
  - Converter os dados para um modelo.

Código exemplo para recuperar os dados salvos no Firestore.

```

1. Stream<List<Student>> fetchStudents() {
2.     return FirebaseFirestore.instance
3.         .collection('students')
4.         .snapshots()
5.         .map((snapshot) => snapshot.docs
6.             .map((doc) => Student.
7.                 fromMap(doc.data() as Map<String,
8.                     dynamic)))
9.             .toList());
10. }
```

#### 5. Funcionalidades de edição e exclusão

- **Objetivo:** ensinar como atualizar e excluir documentos no Firestore.
- **Atividade:**
  - Criar botões para editar ou excluir um aluno na lista.

Código exemplo para atualizar ou deletar dados no Firestore

```

1. void updateStudent(String id, Map<String,
2.     dynamic> data) {
3.     FirebaseFirestore.instance.
4.         collection('students').doc(id).
5.         update(data);
6. }
7. void deleteStudent(String id) {
```

```
6.   FirebaseFirestore.instance.  
      collection('students').doc(id).delete();  
7. }
```

## 6. Personalização e estilização

- **Objetivo:** aprimorar o design do aplicativo.
- **Atividade:**
  - Adicionar temas personalizados;
  - Criar um logo e ícone do app.

Os alunos serão avaliados com base nos seguintes critérios:

- Funcionamento correto do login e autenticação;
- Cadastro e salvamento de informações no Firestore;
- Recuperação e exibição de dados na lista;
- Implementação de edição e exclusão de dados;
- Usabilidade e design da interface.

O produto final deve ser entregue via repositório GitHub. Seu funcionamento e sua implementação devem ser demonstrados em sala de aula.

O aplicativo criado deve permitir:

- Login de professores;
- Cadastro, visualização, edição e exclusão de alunos.

# Conclusão

Neste capítulo, exploramos os principais conceitos relacionados à persistência de dados em Flutter, bem como os recursos disponíveis para armazenar informações de forma eficiente. Foram abordadas ferramentas como Shared Preferences, SqFlite, Hive, Drift e Firebase, além da implementação de operações CRUD, exemplos práticos de aplicativos e opções de armazenamento local e em nuvem.

No próximo capítulo, trataremos de funcionalidades adicionais como bluetooth, acelerômetro, mídia, entre outros recursos que ampliam as possibilidades de interação dos aplicativos com o ambiente físico e digital.



# 5

## CAPÍTULO 5

# RECURSOS DE HARDWARE

### Introdução

O desenvolvimento de aplicativos móveis, a utilização dos recursos de hardware do smartphone é uma etapa fundamental para criar soluções que aproveitem ao máximo as capacidades do dispositivo. Esse processo não é apenas técnico, pois também representa o momento em que sua ideia ganha vida ao interagir com funcionalidades como bluetooth, Wi-Fi, GPS, acelerômetro, áudio, multimídia e câmera.

Imagine um aplicativo que foi cuidadosamente projetado, com interfaces intuitivas e funcionalidades robustas. Para que ele alcance seu público, é necessário que passe por etapas críticas como a integração com esses recursos de hardware. Além disso, os aplicativos modernos precisam de identidade própria, como ícones que sejam visualmente marcantes e alinhados à proposta do projeto.

Também exploraremos os recursos de hardware do smartphone, como bluetooth, Wi-Fi, GPS, acelerômetro, áudio, multimídia e câmera, para desenvolver aplicativos inovadores que proporcionem experiências mais imersivas aos usuários. A integração desses recursos expande as possibilidades criativas e técnicas, permitindo a criação de soluções que interajam de maneira mais eficiente e integrada com o dispositivo móvel.

## Câmera

A integração de funcionalidades de câmera em aplicativos móveis é essencial para diversas aplicações, como captura de fotos, vídeos, escaneamento de documentos e leitura de QR Codes. No Flutter, duas opções populares para trabalhar com câmera são os pacotes **câmera** e **image\_picker**. O pacote **câmera** fornece acesso direto à câmera do dispositivo, permitindo controle avançado, como visualização em tempo real e gravação de vídeos. Já o **image\_picker** é mais indicado para casos em que o objetivo é selecionar imagens da galeria ou capturar fotos rapidamente, sem necessidade de controle contínuo da câmera. A escolha entre esses pacotes depende das necessidades específicas do aplicativo: para funcionalidades básicas, o **image\_picker** é suficiente, mas para controle avançado, a câmera é a melhor opção.

Para utilizar a câmera com o Flutter, é necessário adicionar a dependência **image\_picker** no arquivo **pubspec.yaml**, conforme a figura a seguir.

```
35 # The following adds the Cupertino Icons font to your application.
36 # Use with the CupertinoIcons class for iOS style icons.
37 cupertino_icons: ^1.0.8
38 firebase_core: ^3.6.0
39 cloud_firestore: ^5.4.4
40 image_picker: ^1.1.2
41 firebase_storage: ^12.3.4
42 uuid: ^4.5.1
43
```

**Figura 5.1** – Adicionando a dependência **image\_picker** ao arquivo **pubspec.yaml**.

A seguir, é possível visualizar o código utilizado para tirar foto com a câmera do smartphone.

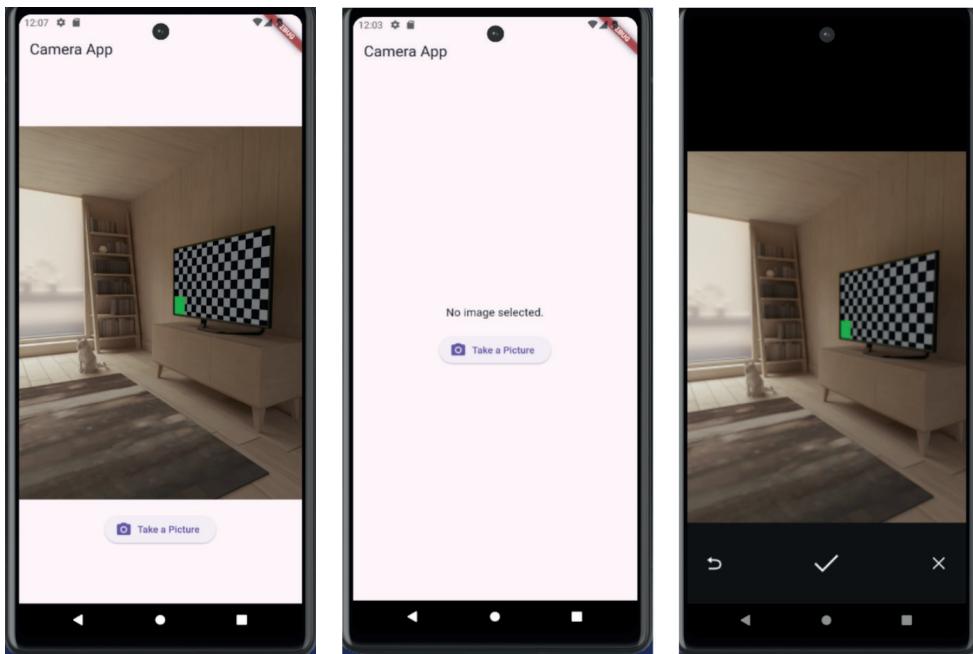
#### Código para utilizar a câmera do smartphone

```
1. // Importa os widgets principais do Flutter.  
2. import 'package:flutter/material.dart';  
3. // Importa o pacote para selecionar imagens da galeria ou  
câmera.  
4. import 'package:image_picker/image_picker.dart';  
5. // Importa a biblioteca para manipular arquivos no sistema  
de arquivos.  
6. import 'dart:io';  
7.  
8. void main() {  
9.   runApp(MyApp()); // Inicia o aplicativo executando a  
classe MyApp.  
10. }  
11.  
12. // Define a classe MyApp, que é um StatelessWidget (não  
muda de estado).  
13. class MyApp extends StatelessWidget {  
14.   @override  
15.   Widget build(BuildContext context) {  
16.     return MaterialApp(  
17.       title: 'Camera App', // Define o título do aplicativo.  
18.       theme: ThemeData(primarySwatch: Colors.blue), //  
Define o tema do app com a cor azul.  
19.       home: CameraScreen(), // Define a tela principal do  
app como CameraScreen.  
20.     );  
21.   }  
22. }  
23.  
24. // Define um StatefulWidget para a tela da câmera.
```

```
25. class CameraScreen extends StatefulWidget {  
26.   @override  
27.   _CameraScreenState createState() => _CameraScreenState();  
    // Cria o estado da tela da câmera.  
28. }  
29.  
30. // Classe que gerencia o estado da CameraScreen.  
31.  
32. class _CameraScreenState extends State<CameraScreen> {  
33.   File? _image; // Variável para armazenar a imagem  
    capturada.  
34.  
35. // Instância do ImagePicker para capturar imagens.  
36.   final ImagePicker _picker = ImagePicker();  
37.  
38.   // Função assíncrona para capturar uma imagem usando a  
    câmera.  
39.   Future<void> _takePicture() async {  
40.     // Abre a câmera e aguarda a captura da imagem  
41.     final pickedFile = await _picker.pickImage(source:  
      ImageSource.camera);  
42.  
43.     // Verifica se uma imagem foi capturada.  
44.     if (pickedFile != null) {  
45.       setState(() {  
46.         // Atualiza a variável _image com o caminho da imagem  
          capturada.  
47.  
48.           _image = File(pickedFile.path);  
49.         });  
50.       }  
51.     }  
52.  
53.   @override
```

```
54.     Widget build(BuildContext context) {  
55.         return Scaffold( // Estrutura básica da tela.  
56.             appBar: AppBar(  
57.                 title: Text('Camera App'), // Define o título da  
58.                     ),  
59.                 body: Center(  
60.                     child: Column(  
61.                         mainAxisAlignment: MainAxisAlignment.center, //  
62.                         Centraliza os widgets verticalmente.  
63.                         children: [  
64.                             _image != null  
65.                             ? Image.file(_image!) // Se houver uma  
66.                               imagem capturada, exibe a imagem.  
67.                             : Text('No image selected.', style:  
68.                               TextStyle(fontSize: 16)), // Caso contrário, exibe um texto  
69.                               informativo.  
70.                             SizedBox(height: 20), // Adiciona um  
71.                               espaçoamento de 20 pixels.  
72.                             ElevatedButton.icon(  
73.                                 onPressed: _takePicture, // Ao clicar, chama  
74.                                 a função _takePicture.  
75.                                 icon: Icon(Icons.camera_alt), // Ícone da  
76.                               câmera no botão.  
77.                                 label: Text('Take a Picture'), // Texto do  
78.                               botão.  
79.                             ),  
80.                         ],  
81.                     ),  
82.                 ),  
83.             );  
84.         }  
85.     }  
86. }
```

As telas que fazem o aplicativo acessar a câmera do smartphone estão representadas na figura a seguir.



**Figura 5.2** – Telas do aplicativo para acessar a câmera do smartphone.

## Áudio

O áudio desempenha um papel fundamental em aplicativos móveis e softwares modernos, enriquecendo a experiência do usuário (UX) e aumentando a funcionalidade das aplicações. De notificações sonoras em aplicativos de mensagens à reprodução de música em apps de streaming, o áudio agrupa um elemento dinâmico e interativo, permitindo que os aplicativos se tornem mais imersivos e funcionais.

O áudio é importante nos aplicativos móveis, pois proporciona:

- **comunicação imediata** – notificações sonoras, alertas e tons personalizados fornecem informações rápidas sem exigir que o usuário

olhe para a tela (por exemplo: um som de notificação em um app de mensagens instantâneas indica uma nova mensagem);

- **acessibilidade** – o áudio é essencial para usuários com deficiências visuais. Aplicativos podem fornecer feedback sonoro ou usar leitores de tela que tornam os apps mais inclusivos (por exemplo: um aplicativo de navegação que fornece instruções de direção por áudio);
- **engajamento do usuário** – aplicativos que utilizam sons e música podem criar experiências mais emocionantes e envolventes (por exemplo: jogos móveis utilizam efeitos sonoros e trilhas sonoras para aumentar a imersão do jogador);
- **funcionalidades essenciais** – muitos aplicativos dependem do áudio como recurso principal, como apps de streaming de música, rádio, audiobooks e podcasts (por exemplo: Spotify e Audible).

Os principais benefícios do uso de áudio são:

- **aprimoramento da experiência do usuário** – sons tornam os aplicativos mais intuitivos, fornecendo feedback imediato para as ações do usuário;
- **maior retenção e engajamento** – áudio em jogos ou notificações sonoras pode manter os usuários mais conectados ao aplicativo;
- **acessibilidade** – oferece suporte a usuários com deficiências visuais ou dificuldades de leitura;
- **imersão** – trilha sonora e efeitos sonoros aumentam a sensação de envolvimento no aplicativo.

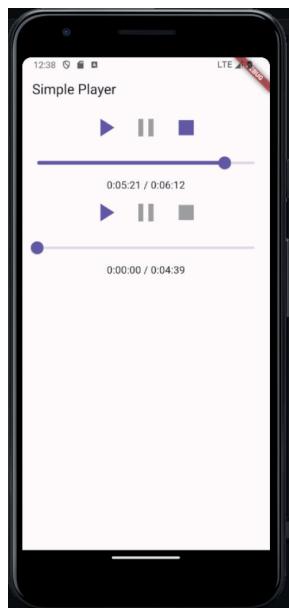
A biblioteca audioplayers é uma das mais populares no Flutter para tocar áudio, permitindo reproduzir, pausar, parar e até mesmo manipular áudio local e remoto. Ela é fácil de configurar e possui recursos avançados, como controle de volume, reprodução em loop e controle de posição.

Na figura a seguir, é possível visualizar a dependência audioplayers adicionada ao arquivo pubspec.yaml do projeto.

```
35 # The following adds the Cupertino Icons font to your application.
36 # Use with the CupertinoIcons class for iOS style icons.
37/cupertino_icons: ^1.0.6
38/audioplayers: ^6.0.0
```

**Figura 5.3** – Dependência audioplayers adicionada ao projeto.

A seguir, apresentamos o código para um aplicativo de reprodução de áudio como o representado na Figura 5.4.



**Figura 5.4** – Aplicativo com audioplayer.

### Código do aplicativo com audioplayer

1. `import 'package:flutter/material.dart'; // Importa os widgets principais do Flutter.`
2. `import 'package:audioplayers/audioplayers.dart'; // Pacote para reprodução de áudio.`
3. `import 'dart:async'; // Pacote para manipulação de operações assíncronas.`

```
4.
5. void main() {
6.   runApp(MaterialApp(
7.     home: Home(), // Define a tela inicial do app.
8.   ));
9. }
10.
11. // Define um StatefulWidget para a tela principal do
12. // aplicativo.
13. class Home extends StatefulWidget {
14.   const Home({super.key});
15.
16.   @override
17.   State<Home> createState() => _HomeState();
18.
19. // Classe que gerencia o estado da tela Home.
20. class _HomeState extends State<Home> {
21.   late AudioPlayer player = AudioPlayer(); // Cria um
22.   // objeto para reprodução de áudio.
23.   late AudioPlayer player1 = AudioPlayer(); // Segundo
24.   // objeto de áudio para tocar simultaneamente.
25.
26.   @override
27.   void initState() {
28.     super.initState();
29.     player = AudioPlayer(); // Inicializa o player.
30.     player.setReleaseMode(ReleaseMode.stop); // Configura o
31.     // player para parar após a reprodução.
32.     WidgetsBinding.instance.addPostFrameCallback((_) async
33.     {
34.       await player.resume(); // Inicia a reprodução.
```

```
33.         await player.setSource(UrlSource('https://www.
    soundhelix.com/examples/mp3/SoundHelix-Song-1.mp3')); // 
        Define a fonte do áudio.
34.
35.         await player1.resume(); // Inicia o segundo player.
36.         await player1.setSource(
37. UrlSource('https://www.soundhelix.com/examples/mp3/
    SoundHelix-Song-3.mp3')); // Define a fonte do segundo
        áudio.
38.     );
39. }
40.
41. @override
42. void dispose() {
43.     player.dispose(); // Libera os recursos do player ao
        fechar a tela.
44.     super.dispose();
45. }
46.
47. @override
48. Widget build(BuildContext context) {
49.     return Scaffold(
50.         appBar: AppBar(
51.             title: Text('App bar'), // Define o título da barra
        superior.
52.         ),
53.         body: Column(
54.             children: [
55.                 PlayerWidget(player: player), // Exibe o
        primeiro player.
56.                 PlayerWidget(player: player1), // Exibe o
        segundo player.
57.             ],
58.             ),
59.         );
60.     }
```

```
61. }
62.
63. // Classe para criar um widget de controle do player de
áudio.
64. class PlayerWidget extends StatefulWidget {
65.   final AudioPlayer player; // Variável para armazenar o
player.
66.
67.   // Construtor que recebe o player como parâmetro.
68.   const PlayerWidget({required this.player, super.key});
69.
70.   @override
71.   State<PlayerWidget> createState() =>
    _PlayerWidgetState();
72. }
73.
74. // Classe que gerencia o estado do widget PlayerWidget.
75. class _PlayerWidgetState extends State<PlayerWidget> {
76.   PlayerState? _playerState; // Estado atual do player
(tocando, pausado, parado).
77.   Duration? _duration; // Duração total do áudio.
78.   Duration? _position; // Posição atual da reprodução.
79.
80.   StreamSubscription? _durationSubscription; // Stream
para monitorar a duração do áudio.
81.   StreamSubscription? _positionSubscription; // Stream
para monitorar a posição do áudio.
82.   StreamSubscription? _playerCompleteSubscription; // Stream
para detectar quando o áudio termina.
83.   StreamSubscription? _playerStateChangeSubscription; // Stream
para monitorar mudanças de estado.
84.
85.   // Funções para verificar se o áudio está tocando ou
pausado.
86.   bool get _isPlaying => _playerState == PlayerState.
```

```
87.     bool get _isPaused => _playerState == PlayerState.  
           paused;  
88.  
89.     // Formata a duração e a posição do áudio para  
           exibição.  
90.     String get _durationText => _duration?.toString().  
           split('.').first ?? '';  
91.     String get _positionText => _position?.toString().  
           split('.').first ?? '';  
92.  
93.     AudioPlayer get player => widget.player; // Obtém o  
           player passado pelo widget pai.  
94.  
95.     @override  
96.     void initState() {  
97.         super.initState();  
98.         _playerState = player.state; // Obtém o estado  
           inicial do player.  
99.  
100.        // Obtém a duração e a posição inicial do áudio.  
101.        player.getDuration().then((value) => setState(() =>  
           _duration = value));  
102.        player.getCurrentPosition().then((value) => setState(()  
           => _position = value));  
103.  
104.        _initStreams(); // Inicializa os listeners para  
           monitorar mudanças no áudio.  
105.    }  
106.  
107.    @override  
108.    void dispose() {  
109.        _durationSubscription?.cancel(); // Cancela a escuta  
           da duração.  
110.        _positionSubscription?.cancel(); // Cancela a escuta  
           da posição.
```

```
111.      _playerCompleteSubscription?.cancel(); // Cancela a
           escuta de finalização.
112.      _playerStateChangeSubscription?.cancel(); // Cancela a
           escuta de mudanças no player.
113.      super.dispose();
114.    }
115.
116.    @override
117.    Widget build(BuildContext context) {
118.      final color = Theme.of(context).primaryColor; // Obtém
           a cor primária do tema.
119.      return Column(
120.        mainAxisSize: MainAxisSize.min,
121.        children: <Widget>[
122.          Row(
123.            mainAxisSize: MainAxisSize.min,
124.            children: [
125.              IconButton(
126.                key: const Key('play_button'),
127.                onPressed: _isPlaying ? null : _play, // Se
           não estiver tocando, inicia o áudio.
128.                iconSize: 48.0,
129.                icon: const Icon(Icons.play_arrow), // Ícone de play.
130.                color: color,
131.              ),
132.              IconButton(
133.                key: const Key('pause_button'),
134.                onPressed: _isPlaying ? _pause : null, // Se
           estiver tocando, pausa o áudio.
135.                iconSize: 48.0,
136.                icon: const Icon(Icons.pause), // Ícone de pausa.
137.                color: color,
138.              ),

```

```
139.           IconButton(
140.             key: const Key('stop_button'),
141.             onPressed: _isPlaying || _isPaused ? _stop
142.               : null, // Se estiver tocando ou pausado, para o áudio.
143.               iconSize: 48.0,
144.               icon: const Icon(Icons.stop), // Ícone de
145.                 stop.
146.               color: color,
147.             ],
148.           ),
149.           Slider(
150.             onChanged: (value) {
151.               final duration = _duration;
152.               if (duration == null) return;
153.               final position = value * duration.
154.                 inMilliseconds;
155.               player.seek(Duration(milliseconds: position.
156.                 round())); // Ajusta a posição do áudio.
157.             },
158.             value: (_position != null &&
159.               _duration != null &&
160.               _position!.inMilliseconds > 0 &&
161.               _position!.inMilliseconds < _duration!.
162.                 inMilliseconds)
163.               ? _position!.inMilliseconds / _duration!.
164.                 inMilliseconds
165.               : 0.0, // Atualiza a barra de progresso.
166.             ),
```

```
167.           : 'o',
168.           style: const TextStyle(fontSize: 16.0),
169.           ),
170.           ],
171.         );
172.       }
173.
174.   // Inicializa os listeners para monitorar mudanças no
175.   // estado do áudio.
176.   void _initStreams() {
177.     _durationSubscription = player.onDurationChanged.
178.       listen((duration) {
179.         setState(() => _duration = duration);
180.       });
181.
182.     _positionSubscription = player.onPositionChanged.
183.       listen((p) {
184.         setState(() => _position = p);
185.       });
186.
187.     _playerCompleteSubscription = player.
188.       onPlayerComplete.listen((event) {
189.         setState(() {
190.           _playerState = PlayerState.stopped;
191.           _position = Duration.zero;
192.         });
193.       });
194.
195.     _playerStateChangeSubscription = player.
196.       onPlayerStateChanged.listen((state) {
```

```
197.  
198. // Inicia a reprodução do áudio.  
199. Future<void> _play() async {  
200.     await player.resume();  
201.     setState(() => _playerState = PlayerState.playing);  
202. }  
203.  
204. // Pausa a reprodução do áudio.  
205. Future<void> _pause() async {  
206.     await player.pause();  
207.     setState(() => _playerState = PlayerState.paused);  
208. }  
209.  
210. // Para o áudio e reseta a posição.  
211. Future<void> _stop() async {  
212.     await player.stop();  
213.     setState(() {  
214.         _playerState = PlayerState.stopped;  
215.         _position = Duration.zero;  
216.     });  
217. }  
218. }
```

Com a ascensão de tecnologias como assistentes de voz e inteligência artificial, o áudio está se tornando ainda mais essencial. Recursos como reconhecimento de fala e geração de som em tempo real estão redefinindo a interação humano-computador.

O áudio é um componente essencial de muitos aplicativos, adicionando profundidade, funcionalidade e acessibilidade. Quando implementado corretamente, pode enriquecer significativamente a experiência do usuário, tornando os aplicativos mais interativos, envolventes e úteis. Desenvolvedores devem sempre considerar o contexto e as necessidades do público-alvo ao integrar áudio em seus aplicativos.

## Localização e GPS

O GPS (Sistema de Posicionamento Global) e as funcionalidades de localização são recursos fundamentais para uma ampla gama de aplicativos modernos, como navegação, entrega de alimentos, serviços de transporte, redes sociais, jogos e muito mais. No Flutter, essas funcionalidades podem ser implementadas de maneira eficiente com bibliotecas como geolocator e google\_maps\_flutter.

**GPS** é uma tecnologia que utiliza uma rede de satélites para determinar a posição geográfica de um dispositivo em qualquer lugar do mundo, oferecendo coordenadas precisas (latitude e longitude).

A **localização** em aplicativos, por sua vez, inclui recursos como GPS, mas não só ele; também podem ser usados o Wi-Fi, torres de celular ou bluetooth para identificar a posição de um dispositivo (por exemplo, a determinação da localização de um usuário dentro de um edifício usando-se Wi-Fi ou bluetooth).

As possibilidades de aplicação da localização são diversas, e incluem:

- **navegação e mapas** – aplicativos como Google Maps, Uber e Waze dependem do GPS para fornecer direções, rastrear rotas e calcular distâncias;
- **serviços baseados em localização** – aplicativos de serviços delivery utilizam localização para conectar clientes e entregadores;
- **experiências personalizadas** – redes sociais e e-commerce oferecem recomendações de conteúdo, produtos ou eventos com base na localização do usuário;
- **jogos** – jogos usam localização para criar experiências interativas baseadas em realidade aumentada;
- **monitoramento e rastreamento** – aplicativos de fitness monitoram trajetos de corrida ou caminhada;

- **apps corporativos** – que rastreiam entregas e frotas;
- **segurança e emergência** – aplicativos de segurança utilizam GPS para rastrear dispositivos ou enviar localização em casos de emergência.

Para exemplificar o uso do GPS em aplicativos, desenvolveremos um que mostra a localização de ONGs.

## Desenvolvimento do aplicativo ONG Maps

O primeiro passo é selecionar como biblioteca de geolocalização a google\_maps\_flutter. Para configurá-la como dependência, é necessário alterar algumas especificações na pasta android/app/build.gradle, conforme mostrado a seguir.

Configurações do arquivo android/app/build.gradle  
para uso do google\_maps\_flutter

```
1. // Declaração dos plugins utilizados no projeto
2. plugins {
3.     id "com.android.application" // Plugin necessário para
        compilar o app Android
4.     id "kotlin-android" // Plugin para suporte ao Kotlin
        no projeto
5.     // O plugin do Flutter deve ser aplicado depois dos
        plugins do Android e Kotlin
6.     id "dev.flutter.flutter-gradle-plugin" // Plugin
        específico do Flutter
7. }
8.
9. android {
10.    namespace = "com.example.ongmaps" // Define o namespace
        do aplicativo
11.    compileSdk = flutter.compileSdkVersion // Define a
        versão da API do Android usada para compilar o app
```

```
12.      ndkVersion = flutter.ndkVersion // Define a versão do
        NDK (Native Development Kit) utilizada
13.
14.      compileOptions {
15.          sourceCompatibility = JavaVersion.VERSION_1_8 // 
        Compatibilidade do código-fonte com Java 8
16.          targetCompatibility = JavaVersion.VERSION_1_8 // 
        Compatibilidade do código gerado com Java 8
17.      }
18.
19.      kotlinOptions {
20.          jvmTarget = JavaVersion.VERSION_1_8 // Define a
        versão do Kotlin JVM para Java 8
21.      }
22.
23.      defaultConfig {
24.          // TODO: Especificar um ID único para a aplicação
25.          applicationId = "com.example.ongmaps" // ID único
        do aplicativo no Android
26.          // Atualize os valores conforme a necessidade do
        app
27.          // Mais detalhes: https://flutter.dev/to/review-gradle-config
28.          minSdkVersion 23 // Define a versão mínima do SDK
        Android suportada pelo app
29.          targetSdkVersion 34 // Define a versão-alvo do SDK
        Android que o app pretende utilizar
30.          versionCode 1 // Código da versão do aplicativo
        (utilizado para controle de atualizações)
31.          versionName "1.0.0" // Nome da versão do
        aplicativo (visível para o usuário)
32.      }
33.
34.      buildTypes {
35.          release {
```

```

36.          // TODO: Adicionar configuração de assinatura
            personalizada para builds de release
37.          // No momento, está usando as chaves de
            depuração para que `flutter run --release` funcione.
38.          signingConfig = signingConfigs.debug // Usa a
            assinatura padrão de debug para compilar o app em release
39.      }
40.  }
41. }
42.
43. flutter {
44.     source = "../.." // Define o diretório raiz do projeto
            Flutter
45. }

```

As principais alterações realizadas no arquivo build.gradle são:

#### **Atualização do minSdkVersion para 23**

- Agora o aplicativo exige Android 6.0 (Marshmallow) ou superior;
- Isso pode ser necessário para utilizar bibliotecas modernas.

#### **Adicionado targetSdkVersion 34**

- Compatibilidade com a API mais recente do Android.

#### **Melhor organização de comentários e informações**

- Explicação sobre applicationId, versionCode e versionName;
- Explicação sobre signingConfig para futuras configurações de assinatura.

#### **Mantidos os plugins essenciais**

- com.android.application (para builds Android);
- kotlin-android (para suporte ao Kotlin);
- dev.flutter.flutter-gradle-plugin (para integração do Flutter com o Gradle).

Na pasta android/app/src/main/AndroidManifest também será necessário realizar uma alteração para colocar a chave da API do Google Maps, conforme indicado a seguir.

### Alteração no Android Manifest

1. Declaração do Manifest
- 2.
3. <manifest xmlns:android="http://schemas.android.com/apk/res/android">
4. Declara o XML do manifesto do Android e define o namespace para os atributos.
- 5.
6. Configuração da aplicação
- 7.
8. <application
9.     android:label="ongmaps"
10.    android:name="\${applicationName}"
11.    android:icon="@mipmap/ic\_launcher">
- 12.
13. android:label="ongmaps" → Define o nome visível do aplicativo.
14. android:name="\${applicationName}" → Substituído automaticamente pelo nome da aplicação durante o build.
15. android:icon="@mipmap/ic\_launcher" → Define o ícone do aplicativo
- 16.
17. Configuração da MainActivity (Atividade Principal)
- 18.
19. <activity
20.     android:name=".MainActivity"
21.     android:exported="true"
22.     android:launchMode="singleTop"
23.     android:taskAffinity=""
24.     android:theme="@style/LaunchTheme"

```
25.     android:configChanges="orientation|keyboardHidden|
           keyboard|screenSize|smallestScreenSize|locale|
           layoutDirection|fontScale|screenLayout|density|uiMode"
26.     android:hardwareAccelerated="true"
27.     android:windowSoftInputMode="adjustResize">
28.
29. android:name=".MainActivity" → Define a atividade principal
       do aplicativo.
30. android:exported="true" → Indica que essa atividade pode
       ser acessada por outros aplicativos.
31. android:launchMode="singleTop" → Garante que a atividade
       não será recriada se já existir no topo da pilha.
32. android:taskAffinity="" → Especifica que essa atividade
       pertence à tarefa padrão do sistema.
33. android:theme="@style/LaunchTheme" → Define o tema inicial
       da atividade enquanto o Flutter carrega.
34. android:configChanges="..." → Indica quais mudanças na
       configuração do dispositivo não reiniciarão a atividade.
35. orientation → Mudança de orientação da tela.
36. keyboardHidden → Ocultação do teclado.
37. screenSize → Mudança no tamanho da tela.
38. density → Alteração na densidade da tela.
39. uiMode → Modo de interface (escuroclaro, por exemplo).
40. android:hardwareAccelerated="true" → Habilita a aceleração
       de hardware para melhor desempenho gráfico.
41. android:windowSoftInputMode="adjustResize" → Faz com que o
       layout seja ajustado quando o teclado virtual aparece.
42. Configuração de Metadados
43.
44. <meta-data
45.     android:name="io.flutter.embedding.android.NormalTheme"
46.     android:resource="@style/NormalTheme"
47.   />
48. Define um tema para a inicialização do aplicativo antes do
       Flutter ser carregado.
```

```
49.  
50. <meta-data android:name="flutterEmbedding"  
      android:value="2" />  
51. Define a versão do sistema de embed do Flutter (usado para  
      carregar o Flutter no Android).  
52.  
53. <meta-data android:name="com.google.android.geo.API_KEY"  
      android:value="AIzaSyC6Z_VBEfOnK_Yw7kKtRXWid9DgawPITHw"/>  
54.  
55. Chave de API do Google Maps → Necessária para integrar  
      mapas ao aplicativo  
56.  
57. Configuração do Intent Filter (Permite o App Ser Iniciado)  
58.  
59. <intent-filter>  
60.     <action android:name="android.intent.action.MAIN"/>  
61.     <category android:name="android.intent.category.  
      LAUNCHER"/>  
62. </intent-filter>  
63.  
64. android.intent.action.MAIN → Define esta Activity como  
      ponto de entrada principal do aplicativo.  
65. android.intent.category.LAUNCHER → Faz com que o  
      aplicativo apareça na lista de apps instalados.  
66. Permissão para Consultar Outras Aplicações  
67.  
68. <queries>  
69.     <intent>  
70.         <action android:name="android.intent.action.  
      PROCESS_TEXT"/>  
71.             <data android:mimeType="text/plain"/>  
72.         </intent>  
73.     </queries>  
74.
```

- 75. **Permite** que o aplicativo veja quais outros aplicativos podem processar ações de texto (`android.intent.action.PROCESS_TEXT`).
- 76. `android:mimeType="text/plain"` → **Indica** que o aplicativo pode interagir com outros apps que manipulam texto.
- 77.
- 78.
- 79.

As principais alterações no Android Manifest estão resumidas a seguir.

- 1. Definição do aplicativo:** define nome, ícone e atividade principal (`MainActivity`).
- 2. Configuração de atividade (`MainActivity`):** define comportamento ao iniciar, ajustes ao mudar configurações do sistema e aceleração de hardware.
- 3. Uso de metadados:** define temas iniciais e integração com o Google Maps via chave de API.
- 4. Permissão para consultar aplicativos:** permite que o app veja quais apps podem processar texto.
- 5. Intent Filters para iniciar o aplicativo:** define o app como principal e o coloca na gaveta de aplicativos.

A seguir, é possível visualizar o código da tela inicial do aplicativo Ong Maps.

#### Código da tela inicial do aplicativo Ong Maps

```

1. import 'package:flutter/material.dart';
2. import 'package:google_maps_flutter/google_maps_flutter.dart';
   // Importando a biblioteca para exibir o Google Maps
3. import 'package:ongmaps/pages/Maps_page.dart'; //
   Importando a página onde o Google Maps será carregado
4.

```

```
5. // Definição da classe WelcomePage, que será um
   StatefulWidget
6. class WelcomePage extends StatefulWidget {
7.   const WelcomePage({super.key}); // Construtor da página
   de boas-vindas
8.
9.   @override
10.  _WelcomePageState createState() => _WelcomePageState();
   // Criação do estado da página
11. }
12.
13. // Definição do estado da WelcomePage
14. class _WelcomePageState extends State<WelcomePage>
15.   with SingleTickerProviderStateMixin {
16.     // Animação para o botão interativo
17.     late AnimationController _controller;
18.     late Animation<double> _animation;
19.
20.   @override
21.   void initState() {
22.     super.initState();
23.     // Inicializa o controlador da animação
24.     _controller = AnimationController(
25.       vsync: this, // O vsync previne o uso excessivo de
   recursos para animações
26.       duration: const Duration(milliseconds: 100), //
   Tempo da animação
27.       lowerBound: 0.0, // Valor inicial da animação
28.       upperBound: 0.1, // Valor máximo da animação
29.     );
30.     // Define a curva da animação (suavização)
31.     _animation = CurvedAnimation(parent: _controller,
   curve: Curves.easeInOut);
32.   }
33.
34.   @override
```

```
35. void dispose() {
36.     _controller.dispose(); // Libera recursos quando o
    widget é destruído
37.     super.dispose();
38. }
39.
40. // Método chamado quando o botão é pressionado
41. void _onTapDown(TapDownDetails details) {
42.     _controller.forward(); // Inicia a animação de
    pressionamento
43. }
44.
45. // Método chamado quando o botão é solto
46. void _onTapUp(TapUpDetails details) async {
47.     _controller.reverse(); // Reverte a animação de
    pressionamento
48.
49.     // Exibe uma tela de carregamento
50.     showDialog(
51.         context: context,
52.         barrierDismissible: false, // Impede que o usuário
        feche a tela de carregamento
53.         builder: (context) => const Center(child:
        CircularProgressIndicator()), // Exibe um indicador de
        progresso
54.     );
55.
56.     // Simula um atraso para o carregamento do mapa
57.     await Future.delayed(const Duration(seconds: 2));
58.
59.     // Fecha a tela de carregamento
60.     Navigator.of(context).pop();
61.
62.     // Navega para a tela do Google Maps
63.     Navigator.push(
```

```
64.      context,
65.      MaterialPageRoute(builder: (context) => const
66.          GoogleMapsFlutter()), // Muda para a tela do mapa
67.      );
68.
69.  @override
70.  Widget build(BuildContext context) {
71.  return Scaffold(
72.      body: Stack( // Stack permite sobrepor elementos
73.          children: [
74.              // Imagem de fundo da tela
75.              Container(
76.                  decoration: BoxDecoration(
77.                      image: DecorationImage(
78.                          image: AssetImage('assets/images/
background_welcome.png'), // Caminho da imagem de fundo
79.                          fit: BoxFit.cover, // Faz a imagem cobrir
80.                              toda a tela mantendo a proporção
81. ),
82. ),
83. ),
84. // Conteúdo central da tela
85. Center(
86.     child: Column(
87.         mainAxisAlignment: MainAxisAlignment.
center, // Centraliza o conteúdo na vertical
88.         children: [
89.             SizedBox(height: 100), // Adiciona espaço
90.                 acima do botão
91.             GestureDetector(
92.                 onTapDown: _onTapDown, // Detecta o
93.                     pressionamento do botão
94.                     onTapUp: _onTapUp, // Detecta quando o
botão é solto
```

```
92.           child: ScaleTransition( // Efeito de
    escala ao pressionar o botão
93.           scale: Tween<double>(begin: 1.0,
    end: 1.1).animate(_animation), // Animação de aumento de
    tamanho
94.           child: Container(
95.               padding: const EdgeInsets.
    symmetric(vertical: 15.0, horizontal: 50.0), // Define o
    tamanho do botão
96.               decoration: BoxDecoration(
97.                   color: Colors.white.
    withOpacity(0.8), // Define a cor de fundo do botão com
    transparência
98.               borderRadius: BorderRadius.
    circular(30.0), // Arredonda as bordas do botão
99.           ),
100.          child: const Text(
101.              'ONGMaps', // Texto exibido no
    botão
102.              style: TextStyle(
103.                  fontSize: 28.0, // Tamanho da
    fonte do botão
104.                  color: Color(0xFF0A0A0A), // 
    Cor do texto do botão (azul escuro quase preto)
105.                  fontWeight: FontWeight.bold,
    // Deixa o texto em negrito
106.              ),
107.              ),
108.              ),
109.              ),
110.              ],
111.              ],
112.              ],
113.              ],
114.              ],
115.          ),
```

```
116.      );
117.    }
118. }
```

A seguir, apresentamos o código que cria marcadores de latitude e longitude das ongs.

#### Código dos marcadores de latitude e longitude

```
1. import 'package:flutter/material.dart'; // Importa o
   pacote Flutter Material para widgets e temas.
2. import 'package:google_maps_flutter/google_maps_flutter.
   dart'; // Importa o Google Maps para Flutter.
3. import 'package:custom_info_window/custom_info_
   window.dart'; // Importa a biblioteca para janelas de
   informações personalizadas.
4. import 'package:ongmaps/pages/Descricao_page.dart'; //
   Importa a página de descrição dos locais.
5.
6. // Classe principal do Google Maps que será um
   StatefulWidget para permitir atualização dinâmica.
7. class GoogleMapsFlutter extends StatefulWidget {
8.   const GoogleMapsFlutter({super.key});
9.
10.  @override
11.  State<GoogleMapsFlutter> createState() =>
12.    _GoogleMapsFlutterState();
13.
14. // Estado associado ao GoogleMapsFlutter.
15. class _GoogleMapsFlutterState extends
   State<GoogleMapsFlutter> {
16.   // Controlador para a janela de informações
   personalizadas no mapa.
17.   final CustomInfoWindowController _  

   customInfoWindowController = CustomInfoWindowController();
```

```
18.
19. // Conjunto de marcadores que serão exibidos no mapa.
20. Set<Marker> markers = {};
21.
22. // Lista de coordenadas (latitude e longitude) para os
   pontos de interesse.
23. final List<LatLng> latlongPoint = [
24.   const LatLng(-22.858249818019033, -47.04822860807837),
   // Local ABRAES
25.   const LatLng(-22.900747199053033, -47.061201838471305),
   // Local IAPI
26. ];
27.
28. // Lista com os nomes dos locais.
29. final List<String> locationNames = ["ABRAES", "IAPI"];
30.
31. // Lista de widgets que representam as páginas de
   descrição dos locais.
32. final List<Widget> locationDescriptions = [const
   DISCRIPTION_ABRAES(), const DISCRIPTION_IAPI()];
33.
34. // Lista de URLs das imagens representativas dos
   locais.
35. final List<String> locationImages = [
36.   "https://imgs.search.brave.com/
   hMgEspYNZw3vsej4ugzJB227HNaFwErw1XZNDCE301Q/rs:fit:860:0:0:0/
   g:ce/aHR0cDovL3d3dy5v/bmdzYnJhc2lsLmNv/bS5ici9pbWFnZXMu/
   b25ncy1kZS1jcmlh/bmNhcy5qcGc",
37.   "https://imgs.search.brave.com/
   Q5LwmAOVKZb7mNtX7haSs1tvohCHVC7vSHH5d3XxTKA/
   rs:fit:860:0:0/g:ce/aHR0cHM6Ly9jbGFz/c2ljLmV4YW1lLmNv/
   bS93cC1jb250ZW50/L3VwbG9hZHMuMjAy/My8xMC9Gb3RvLTNF/
   T2ZpY2luYS1jb20t/Y3JpYW5jYXMuEt/UGluYWNUvdGVjYV9E/
   aXZ1bGdhY2FvLmpw/Zz9xdWFsaXR5PTcw/JnN0cmlwPWluZm8m/
   dz0xMDI0dW5kZWZp/bmVk",
38. ];
```

```
39.  
40.    @override  
41.    void initState() {  
42.        super.initState();  
43.        displayInfo(); // Inicializa e exibe os marcadores no  
        mapa.  
44.    }  
45.  
46.    // Método para configurar os marcadores e adicionar a  
    funcionalidade de janelas de informações personalizadas.  
47.    void displayInfo() {  
48.        for (int i = 0; i < latlongPoint.length; i++) {  
49.            markers.add(  
50.                Marker(  
51.                    markerId: MarkerId(i.toString()), // Define um ID  
                    único para o marcador.  
52.                    icon: BitmapDescriptor.defaultMarker, // Define  
                    o ícone padrão do marcador.  
53.                    position: latlongPoint[i], // Define a posição  
                    do marcador no mapa.  
54.                    onTap: () {  
55.                        // Exibe a janela de informações  
                        personalizada ao tocar no marcador.  
56.                        _customInfoWindowController.addInfoWindow(  
57.  
58.                            _buildInfoWindow(locationNames[i],  
                            locationImages[i], i),  
59.                            latlongPoint[i],  
60.                            );  
61.                            },  
62.                            ),  
63.                            );  
64.            }  
65.        setState(() {}); // Atualiza a interface para exibir  
        os marcadores corretamente.
```

```
66.      }
67.
68.      // Método para construir a janela de informações
69.      personalizada para cada marcador.
70.      Widget _buildInfoWindow(String name, String imageUrl,
71.          int index) {
72.          return Container(
73.              width: 250, // Define a largura da janela.
74.              decoration: BoxDecoration(
75.                  color: Colors.white, // Define a cor de fundo
76.                  branca.
77.                  borderRadius: BorderRadius.circular(15.0), //
78.                  Arredonda os cantos da janela.
79.              ),
80.              child: Column(
81.                  mainAxisAlignment: CrossAxisAlignment.start,
82.                  children: [
83.                      ClipRRect(
84.                          borderRadius: const BorderRadius.only(
85.                              topLeft: Radius.circular(15.0),
86.                              topRight: Radius.circular(15.0),
87.                          ),
88.                          child: Image.network(
89.                              imageUrl, // Carrega a imagem do local a
90.                              partir da URL.
91.                              height: 125, // Define a altura da imagem.
92.                              width: 250, // Define a largura da imagem.
93.                              fit: BoxFit.cover, // Ajusta a imagem para
94.                              cobrir toda a área definida.
95.                              loadingBuilder: (context, child,
96.                              loadingProgress) {
97.                                  if (loadingProgress == null) return
98.                                  child; // Exibe a imagem quando carregada.
```

```
91.           return const Center(child:  
92.             CircularProgressIndicator()); // Exibe um indicador de  
93.             carregamento enquanto a imagem carrega.  
94.           },  
95.           errorBuilder: (context, error, stackTrace)  
96.           {  
97.             return const Center(  
98.               child: Text(  
99.                 'Erro ao carregar imagem', //  
100.                Mensagem de erro caso a imagem não carregue.  
101.                style: TextStyle(color: Colors.red),  
102.              ),  
103.              );  
104.            },  
105.            ),  
106.            ),  
107.            Padding(  
108.              padding: const EdgeInsets.all(8.0), //  
109.              Adiciona espaçamento interno na janela.  
110.              child: Row(  
111.                mainAxisAlignment: MainAxisAlignment.  
112.                spaceBetween,  
113.                children: [  
114.                  Expanded(  
115.                    child: Text(  
116.                      name, // Nome do local.  
117.                      style: const TextStyle(  
118.                        fontWeight: FontWeight.bold,  
119.                        fontSize: 18,  
120.                      ),  
121.                      ),  
122.                      ),  
123.                      const SizedBox(width: 8), // Espaço entre  
124.                      o texto e o botão.  
125.                      ElevatedButton(  
126.                        onPressed: () {  
127.                          Navigator.pushNamed(context, '/');  
128.                        },  
129.                        child: Text('OK'),  
130.                      ),  
131.                    ),  
132.                  ),  
133.                ),  
134.              ),  
135.            ),  
136.          ),  
137.        ),  
138.      ),  
139.    ),  
140.  ),  
141.);
```

```
119.             onPressed: () {
120.                 // Navega para a página de descrição
121.                 // correspondente ao local.
122.                 Navigator.push(
123.                     context,
124.                     MaterialPageRoute(builder:
125.                         (context) => locationDescriptions[index]),
126.                     );
127.                 },
128.                 style: ElevatedButton.styleFrom(
129.                     backgroundColor: Colors.white, // Cor
130.                     de fundo do botão.
131.                     side: const BorderSide(color: Colors.
132.                     black), // Cor da borda do botão.
133.                     shape: RoundedRectangleBorder(
134.                         borderRadius: BorderRadius.
135.                         circular(10.0), // Bordas arredondadas do botão.
136.                         ),
137.                         ),
138.                         ],
139.                         ],
140.                         );
141.             }
142.
143.             @override
144.             Widget build(BuildContext context) {
145.                 return Scaffold(
146.                     appBar: AppBar(
147.                         title: const Text(
148.                             “ONGMaps”, // Título da barra superior.
```

```
149.           style: TextStyle(
150.             fontWeight: FontWeight.bold, // Define a fonte
151.               em negrito.
152.             ),
153.             ),
154.             backgroundColor: const Color.fromARGB(255, 20,
155.               68, 107), // Define a cor de fundo da AppBar.
156.             ),
157.             body: Stack(
158.               children: [
159.                 GoogleMap(
160.                   initialCameraPosition: const CameraPosition(
161.                     target: LatLng(-22.912847283240453,
162.                       -47.041346547261014), // Define a posição inicial do mapa.
163.                     zoom: 12, // Define o nível de zoom inicial.
164.                     ),
165.                     onTap: (LatLng position) {
166.                       _customInfoWindowController.
167.                         hideInfoWindow!(); // Esconde a janela de informações ao
168.                           tocar no mapa.
169.                         },
170.                         onCameraMove: (position) {
171.                           _customInfoWindowController.
172.                             onCameraMove!(); // Mantém a posição da janela de
173.                               informações ao mover o mapa.
174.                             },
175.                             onMapCreated: (GoogleMapController
176.                               controller) {
177.                                 _customInfoWindowController.
178.                                   googleMapController = controller; // Vincula o
179.                                     controlador do mapa ao controlador da janela de
180.                                       informações.
```

```
172.           },
173.           ),
174.           CustomInfoWindow(
175.             controller: _customInfoWindowController, //  
Define o controlador da janela de informações.  

176.             height: 190, // Define a altura da janela.  

177.             width: 250, // Define a largura da janela.  

178.             offset: 35, // Define o deslocamento da janela  
em relação ao marcador.  

179.           ),  

180.         ],  

181.       ),  

182.     );  

183.   }  

184. }  

185.  

186. Explicação da função _buildInfoWindow  

187.  

188. // Função responsável por construir a janela de  
informações personalizadas dos marcadores no mapa.  

189. Widget _buildInfoWindow(String name, String imageUrl, int  
index) {  

190.   return Container(  

191.     width: 250, // Define a largura da janela de  
informações.  

192.     decoration: BoxDecoration(  

193.       color: Colors.white, // Define a cor de fundo da  
janela.  

194.       borderRadius: BorderRadius.circular(15.0), // Aplica  
cantos arredondados para um design moderno.  

195.     ),  

196.     child: Column(  

197.       crossAxisAlignment: CrossAxisAlignment.start, //  
Alinha os elementos no início do eixo horizontal.  

198.       children: [  

199.         
```

```
200.          // Exibe a imagem do local dentro de um
              contêiner com bordas arredondadas na parte superior.
201.          ClipRRect(
202.            borderRadius: const BorderRadius.only(
203.              topLeft: Radius.circular(15.0), // Arredonda o
              canto superior esquerdo.
204.              topRight: Radius.circular(15.0), // Arredonda
              o canto superior direito.
205.            ),
206.            child: Image.network(
207.              imageUrl, // Carrega a imagem do local a
              partir de uma URL fornecida.
208.              height: 125, // Define a altura da imagem
              dentro da janela.
209.              width: 250, // Define a largura da imagem
              para preencher todo o espaço da janela.
210.              fit: BoxFit.cover, // Ajusta a imagem para
              cobrir toda a área disponível sem distorção.
211.
212.          // Exibe um indicador de carregamento
              enquanto a imagem ainda está sendo carregada.
213.          loadingBuilder: (context, child,
              loadingProgress) {
214.            if (loadingProgress == null) return child;
              // Se o carregamento estiver concluído, exibe a imagem.
215.            return const Center(child:
              CircularProgressIndicator()); // Caso contrário, exibe o
              indicador de carregamento.
216.          },
217.
218.          // Caso ocorra um erro ao carregar a imagem,
              exibe uma mensagem de erro.
219.          errorBuilder: (context, error, stackTrace) {
220.            return const Center(
              child: Text(
```

```
222.           'Erro ao carregar imagem', // Mensagem
    informando que a imagem não pôde ser carregada.
223.           style: TextStyle(color: Colors.red), //
    Define a cor do texto para vermelho para indicar erro.
224.           ),
225.           );
226.           },
227.           ),
228.           ),
229.
230.           // Adiciona um espaçamento interno ao redor do
    conteúdo da janela de informações.
231.           Padding(
232.             padding: const EdgeInsets.all(8.0),
233.             child: Row(
234.               mainAxisAlignment: MainAxisAlignment.
    spaceBetween, // Distribui os elementos horizontalmente.
235.               children: [
236.                 // Exibe o nome do local em um texto
    estilizado.
237.                 Expanded(
238.                   child: Text(
239.                     name, // Nome do local correspondente
    ao marcador.
240.                     style: const TextStyle(
241.                       fontWeight: FontWeight.bold, // Define
    o texto como negrito.
242.                       fontSize: 18, // Define o tamanho da
    fonte.
243.                     ),
244.                     ),
245.                     ),
246.
247.                     const SizedBox(width: 8), // Adiciona um
    pequeno espaçamento entre o nome e o botão.
248.
```

```
249.          // Botão que permite navegar para a página  
de descrição do local.  
250.          ElevatedButton(  
251.              onPressed: () {  
252.                  // Navega para a página de descrição  
correspondente ao local.  
253.                  Navigator.push(  
254.                      context,  
255.                      MaterialPageRoute(  
256.                          builder: (context) =>  
locationDescriptions[index], // Acessa a descrição correta  
do local pelo índice.  
257.                      ),  
258.                  );  
259.              },  
260.              style: ElevatedButton.styleFrom(  
261.                  backgroundColor: Colors.white, // Define  
a cor de fundo do botão como branco.  
262.                  side: const BorderSide(color: Colors.  
black), // Adiciona uma borda preta ao botão.  
263.                  shape: RoundedRectangleBorder(  
264.                      borderRadius: BorderRadius.  
circular(10.0), // Aplica cantos arredondados ao botão.  
265.                      ),  
266.                  ),  
267.                  child: const Icon(  
268.                      Icons.arrow_forward, // Ícone de seta  
indicando que leva a outra tela.  
269.                      color: Colors.black, // Define a cor do  
ícone como preta.  
270.                      ),  
271.                      ),  
272.                  ],  
273.                  ),  
274.              ),  
275.          ],
```

```
276.      ),  
277.    );  
278.  }  
279.  
280.  
281.
```

A seguir são apresentadas as explicações sobre cada Widget utilizado na função `_buildInfoWindow`.

- **Container** – define a estrutura da janela de informações, incluindo largura e bordas arredondadas.
- **BoxDecoration** – define a cor de fundo da janela e a aplicação de cantos arredondados.
- **Column** – organiza os elementos da janela verticalmente.
- **ClipRRect** – aplica bordas arredondadas apenas na parte superior da imagem.
- **Image.network** – exibe a imagem do local, carregada a partir de uma URL.
- **loadingBuilder** – exibe um CircularProgressIndicator enquanto a imagem carrega.
- **errorBuilder** – exibe uma mensagem, caso o carregamento da imagem falhe.
- **Padding** – adiciona espaçamento interno ao redor do conteúdo da janela.
- **Row** – organiza o nome do local e o botão lado a lado.
- **Expanded** – faz com que o nome do local ocupe o máximo de espaço disponível antes do botão.
- **SizedBox** – adiciona um pequeno espaço entre o nome e o botão.
- **ElevatedButton** – botão que permite a navegação para a página de descrição do local.

- **style** – define a aparência do botão, incluindo cor de fundo e bordas arredondadas.
- **child** – contém o ícone de seta (Icons.arrow\_forward).

A seguir, é possível visualizar o código da tela de descrição.

#### Código da tela de descrição

```
1. import 'package:flutter/material.dart'; // Importa o
   pacote principal do Flutter para UI.
2.
3. // Classe para a descrição da ONG ABRAES.
4. class DISCRIPTION_ABRAES extends StatelessWidget {
5.   const DISCRIPTION_ABRAES({Key? key}) : super(key: key);
6.
7.   @override
8.   Widget build(BuildContext context) {
9.     return Scaffold(
10.       appBar: AppBar(
11.         title: const Text(
12.           "Descrição ABRAES", // Título da AppBar.
13.           style: TextStyle(
14.             fontWeight: FontWeight.bold, // Define o
15.             texto como negrito.
16.             color: Colors.white, // Define a cor do texto
17.             como branca.
18.           ),
19.           ),
20.           body: Padding(
21.             padding: const EdgeInsets.all(16.0), // Define um
22.             espacamento interno.
23.             child: Column(
```

```
23.           crossAxisAlignment: CrossAxisAlignment.center,  
24.           // Centraliza os elementos horizontalmente.  
25.           children: [  
26.             // Exibe uma imagem da ONG ABRAES.  
27.             Image.network(  
28.               „https://img.s.search.brave.com/  
29.               hMgEspYNZw3vsej4ugzJB227HNaFwErw1XZNCE301Q/rs:fit:860:0:0:0/  
30.               g:ce/aHR0cDovL3d3dy5v/bmdzYnJhc2lsLmNv/bS5ici9pbWFnZXMy/  
31.               b25ncy1kZS1jcmlh/bmNhcy5qcGc“,  
32.               fit: BoxFit.cover, // Faz a imagem cobrir a  
33.               // área do contêiner.  
34.               height: 190, // Define a altura da imagem.  
35.               width: 280, // Define a largura da imagem.  
36.             ),  
37.             const SizedBox(height: 16), // Espaço entre a  
38.             // imagem e o texto.  
39.             // Nome da ONG.  
40.             const Text(  
41.               “ABRAES - Associação Brasileira de  
42.               Educação Social”,  
43.               style: TextStyle(fontSize: 24, fontWeight:  
44.               FontWeight.bold, color: Colors.white),  
45.               textAlign: TextAlign.center, // Centraliza  
46.               o texto.  
47.             ),  
48.             const SizedBox(height: 16), // Espaço entre  
49.             os textos.  
50.             const Text(  
51.               “Fundação: 2010”, // Ano de fundação da ONG.  
52.               style: TextStyle(fontSize: 18, color:  
53.               Colors.white),  
54.             ),  
55.             const SizedBox(height: 8), // Pequeno  
56.             // espaçamento entre os textos.  
57.             const Text(  
58.               “
```

```
46.           “Sede: R. das Orquídeas, 500 - Chácara
    Primavera, Campinas - SP, 13087-430”, // Endereço da ONG.
47.           style: TextStyle(fontSize: 18, color:
    Colors.white),
48.           ),
49.           const SizedBox(height: 8),
50.           const Text(
51.               “Atividades: Programas de reforço escolar,
    atividades culturais e esportivas.”, // Descrição das
    atividades da ONG.
52.           style: TextStyle(fontSize: 18, color:
    Colors.white),
53.           ),
54.           const SizedBox(height: 8),
55.           const Text(
56.               “Foco: Desenvolvimento integral de
    crianças e adolescentes em situação de vulnerabilidade.”,
    // Objetivo da ONG.
57.           style: TextStyle(fontSize: 18, color:
    Colors.white),
58.           ),
59.           ],
60.           ),
61.           ),
62.           backgroundColor: const Color.fromARGB(255, 30, 75,
    120), // Define a cor de fundo da tela.
63.       );
64.   }
65. }
66.
67. ///////////////////////////////// IAPI /////////////////////
    /////////////////////////////////
68.
69. /// Classe para a descrição da ONG IAPI.
70. class DISCRIPTION_IAPI extends StatelessWidget {
```

```
71.  const DISCRIPTION_IAPI({Key? key}) : super(key: key);
72.
73. @override
74. Widget build(BuildContext context) {
75.   return Scaffold(
76.     appBar: AppBar(
77.       title: const Text(
78.         "Descrição IAPI", // Título da AppBar.
79.         style: TextStyle(
80.             fontWeight: FontWeight.bold, // Define o
81.             texto como negrito.
82.             color: Colors.white, // Define a cor do texto
83.             como branca.
84.           ),
85.           ),
86.           backgroundColor: const Color.fromARGB(255, 20,
87.             68, 107), // Define a cor de fundo da AppBar.
88.           ),
89.           body: Padding(
90.             padding: const EdgeInsets.all(16.0), // Adiciona
91.             espaçamento interno.
92.             child: Column(
93.               crossAxisAlignment: CrossAxisAlignment.center,
94.               // Centraliza os elementos horizontalmente.
95.               children: [
96.                 // Exibe uma imagem da ONG IAPI.
97.                 Image.network(
98.                   "https://imgs.search.brave.com/
99. Q5LwmAOVKzb7mNtX7haSs1tvohCHVC7vSHH5d3XxTKA/
100. rs:fit:860:0:0:g:ce/aHR0cHM6Ly9jbGFz/c2ljLmV4YW1lLmNv/
101. bS93cC1jb250ZW50/L3VwbG9hZHMvMjAy/My8xMC9Gb3RvLTNF/
102. T2ZpY2luYS1jb20t/Y3JpYW5jYXMtbmEt/UGluYWNvdGVjYV9E/
103. aXZ1bGdhY2FvLmpw/Zz9xdWFsaXR5PTcw/JnN0cmlwPWluZm8m/
104. dz0xMDI0dW5kZWZp/bmVk",
```

```
94.           fit: BoxFit.cover, // Faz a imagem cobrir a
área do contêiner.
95.           height: 190, // Define a altura da imagem.
96.           width: 280, // Define a largura da imagem.
97.         ),
98.         const SizedBox(height: 16), // Espaço entre a
    imagem e o texto.
99.           // Nome da ONG.
100.          const Text(
101.            “Instituto de Assistência e Proteção à
    Infância (IAPI)”,  

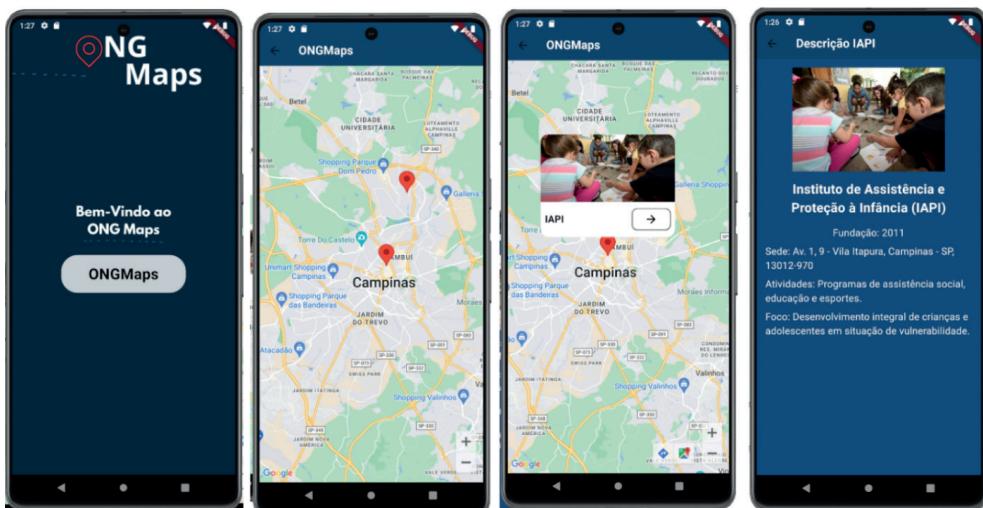
102.            style: TextStyle(fontSize: 24, fontWeight:
    FontWeight.bold, color: Colors.white),
103.            textAlign: TextAlign.center, // Centraliza
    o texto.
104.          ),
105.          const SizedBox(height: 16),
106.          const Text(
107.            “Fundação: 2011”, // Ano de fundação da ONG.
108.            style: TextStyle(fontSize: 18, color:
    Colors.white),
109.          ),
110.          const SizedBox(height: 8),
111.          const Text(
112.            “Sede: Av. 1, 9 - Vila Itapura, Campinas -
    SP, 13012-970”, // Endereço da ONG.
113.            style: TextStyle(fontSize: 18, color:
    Colors.white),
114.          ),
115.          const SizedBox(height: 8),
116.          const Text(
117.            “Atividades: Programas de assistência social,
    educação e esportes.”, // Descrição das atividades da ONG.
118.            style: TextStyle(fontSize: 18, color:
    Colors.white),
```

```

119.           ),
120.           const SizedBox(height: 8),
121.           const Text(
122.             “Foco: Desenvolvimento integral de
123.             crianças e adolescentes em situação de vulnerabilidade.”,
124.             // Objetivo da ONG.
125.             style: TextStyle(fontSize: 18, color:
126.               Colors.white),
127.             ),
128.             backgroundColor: const Color.fromARGB(255, 30, 75,
129.               120), // Define a cor de fundo da tela.
130.             );
131.           }

```

Na figura a seguir, as telas do aplicativo Ong Maps são representadas.



**Figura 5.5 – Telas do aplicativo Ong Maps.**

## Acelerômetro

O acelerômetro é um dos sensores mais comuns encontrados em smartphones e mede a aceleração linear de um dispositivo em relação ao eixo gravitacional. Essa medição se dá em três eixos (X, Y, Z) e é essencial para várias funcionalidades, desde detecção de movimento até interações avançadas com aplicativos.

O acelerômetro desempenha um papel fundamental em como integramos com dispositivos móveis e na criação de experiências dinâmicas e envolventes. Em smartphones, ele geralmente mede a aceleração no:

- eixo X (esquerda para a direita ou vice-versa);
- eixo Y (de baixo para cima ou vice-versa);
- eixo Z (frontal para traseira ou vice-versa, geralmente relacionado ao impacto da gravidade).

Além disso, ele pode detectar a inclinação e o movimento do dispositivo.

## Funcionamento

O acelerômetro utiliza microssistemas eletromecânicos (MEMS), que são pequenos componentes que detectam mudanças de movimento com base na força aplicada a massas internas sensíveis. Ele converte essas forças em sinais elétricos que são interpretados pelo sistema operacional do smartphone.

As principais características dos acelerômetros são:

- pequenas estruturas no sensor se movem com base na força da aceleração;
- essas mudanças são medidas e traduzidas em valores numéricos (geralmente, em metros por segundo ao quadrado,  $m/s^2$ );
- esses dados são utilizados pelo sistema para calcular a posição ou detectar movimentos.

## Aplicações do acelerômetro em smartphones

O acelerômetro é amplamente utilizado em diversas funcionalidades e aplicativos no smartphone. Algumas das principais aplicações incluem:

- **detecção de orientação** – determina se o dispositivo está em modo retrato (vertical) ou paisagem (horizontal); também usado para ajustar a interface do usuário (ex.: rotação automática da tela);
- **jogos** – muitos jogos usam o acelerômetro para controlar movimentos, como direção em jogos de corrida ou interação em jogos baseados em realidade aumentada;
- **saúde e fitness** – aplicativos de monitoramento de atividades físicas usam o acelerômetro para contar passos, calcular distâncias percorridas e até medir calorias queimadas;
- **detecção de queda** – recursos de segurança, como a detecção de quedas em smartwatches e smartphones, utilizam o acelerômetro para identificar mudanças bruscas de movimento;
- **realidade virtual e aumentada (VR/AR)** – em aplicativos de VR/AR, o acelerômetro ajuda a rastrear os movimentos do dispositivo no espaço;
- **economia de energia** – usado para detectar se o dispositivo está parado; alguns sistemas ajustam o consumo de energia com base na atividade detectada;
- **aplicativos de navegação** – em conjunto com o giroscópio e o GPS, o acelerômetro melhora a precisão da navegação e do rastreamento.

Para utilizar o acelerômetro, é necessário adicionar a dependência sensor\_plus e fl\_chart para plotar os gráficos de cada eixo, conforme a figura a seguir.

```

35  # The following adds the Cupertino Icons font to your application.
36  # Use with the CupertinoIcons class for iOS style icons.
37 /cupertino_icons: ^1.0.8
38  sensors_plus: ^6.1.1
39  fl_chart: ^0.69.2
40
41 dev_dependencies:
42   flutter_test:
43     sdk: flutter

```

**Figura 5.6 – Inclusão das dependências no arquivo pubspec.yaml para utilizar o acelerômetro.**

A seguir, é possível visualizar o código do aplicativo realizando leituras do acelerômetro e plotando as medidas nos gráficos.

### Aplicativo acelerômetro

```

1. import 'dart:async'; // Importa a biblioteca para
   manipulação de streams assíncronas.
2. import 'package:flutter/material.dart'; // Importa o
   pacote principal do Flutter.
3. import 'package:sensors_plus/sensors_plus.dart'; //
   Importa o pacote para acessar os sensores do dispositivo.
4. import 'package:fl_chart/fl_chart.dart'; // Importa a
   biblioteca para desenhar gráficos no Flutter.
5.
6. void main() {
7.   runApp(const MyApp()); // Inicia o aplicativo chamando a
   classe MyApp.
8. }
9.
10. // Classe principal do aplicativo.
11. class MyApp extends StatelessWidget {
12.   const MyApp({super.key});
13.
14.   @override
15.   Widget build(BuildContext context) {
16.     return MaterialApp(

```

```
17.         debugShowCheckedModeBanner: false, // Remove a
    faixa de debug.
18.         home: const AccelerometerApp(), // Define a tela
    inicial do app.
19.     );
20. }
21. }
22.
23. // Tela que exibe os dados do acelerômetro e gráficos.
24. class AccelerometerApp extends StatefulWidget {
25.     const AccelerometerApp({super.key});
26.
27.     @override
28.     State<AccelerometerApp> createState() =>
    _AccelerometerAppState();
29. }
30.
31. // Estado do widget AccelerometerApp.
32. class _AccelerometerAppState extends
    State<AccelerometerApp> {
33.     double x = 0.0, y = 0.0, z = 0.0; // Variáveis para
    armazenar os valores do acelerômetro.
34.     late StreamSubscription<AccelerometerEvent> _
    streamSubscription; // Assinatura para escutar os eventos
    do acelerômetro.
35.
36.     // Listas para armazenar os pontos dos gráficos.
37.     final List<FlSpot> _xData = [];
38.     final List<FlSpot> _yData = [];
39.     final List<FlSpot> _zData = [];
40.     int _time = 0; // Contador de tempo para os gráficos.
41.     final int maxDataPoints = 50; // Limite de pontos nos
    gráficos.
42.
43.     @override
```

```
44. void initState() {
45.     super.initState();
46.     // Inicia a escuta dos eventos do acelerômetro.
47.     _streamSubscription = accelerometerEvents.
48.         listen((event) {
49.             setState(() {
50.                 x = event.x; // Atualiza a leitura do eixo X.
51.                 y = event.y; // Atualiza a leitura do eixo Y.
52.                 z = event.z; // Atualiza a leitura do eixo Z.
53. 
54.                 // Adiciona os novos valores aos gráficos.
55.                 _xData.add(FlSpot(_time.toDouble(), x));
56.                 _yData.add(FlSpot(_time.toDouble(), y));
57.                 _zData.add(FlSpot(_time.toDouble(), z));
58. 
59.                 // Remove os pontos mais antigos para manter um
60.                 // número fixo de dados exibidos.
61.                 if (_xData.length > maxDataPoints) _xData.
62.                     removeAt(0);
63.                 if (_yData.length > maxDataPoints) _yData.
64.                     removeAt(0);
65.                 if (_zData.length > maxDataPoints) _zData.
66.                     removeAt(0);
67. 
68.             @override
69.             void dispose() {
70.                 _streamSubscription.cancel(); // Cancela a assinatura
71.                 do acelerômetro quando o widget for descartado.
72.             }
```

```
73.  
74.    @override  
75.    Widget build(BuildContext context) {  
76.        return Scaffold(  
77.            appBar: AppBar(  
78.                title: const Text("Acelerômetro com Gráficos"), //  
    Título da AppBar.  
79.            ),  
80.            body: Padding(  
81.                padding: const EdgeInsets.all(16.0), // Adiciona  
    espaçamento interno.  
82.                child: Column(  
83.                    children: [  
84.                        const Text(  
85.                            "Valores do Acelerômetro",  
86.                            style: TextStyle(fontSize: 20, fontWeight:  
    FontWeight.bold),  
87.                        ),  
88.                        const SizedBox(height: 20), // Espaçamento  
    entre elementos.  
89.  
90.                        // Gráfico que exibe os dados do  
    acelerômetro.  
91.                        Expanded(  
92.                            child: LineChart(  
93.                                LineChartData(  
94.                                    lineBarsData: [  
95.                                        // Linha do eixo X.  
96.                                        LineChartBarData(  
97.                                            spots: _xData,  
98.                                            isCurved: true, // Faz a linha do  
    gráfico ser suave.  
99.                                            barWidth: 2, // Define a espessura  
    da linha.  
100.                                           color: Colors.red, // Define a cor  
    da linha.
```

```
101.                                belowBarData: BarAreaData(show:  
102.          false), // Oculta a área abaixo da linha.  
103.          ),  
104.          // Linha do eixo Y.  
105.          LineChartBarData(  
106.            spots: _yData,  
107.            isCurved: true,  
108.            barWidth: 2,  
109.            color: Colors.green,  
110.            belowBarData: BarAreaData(show:  
111.              false),  
112.              dotData: FlDotData(show: false),  
113.            ),  
114.            // Linha do eixo Z.  
115.            LineChartBarData(  
116.              spots: _zData,  
117.              isCurved: true,  
118.              barWidth: 2,  
119.              color: Colors.blue,  
120.              belowBarData: BarAreaData(show:  
121.                false),  
122.                dotData: FlDotData(show: false),  
123.              ),  
124.            ],  
125.            titlesData: FlTitlesData(  
126.              leftTitles: AxisTitles(  
127.                sideTitles: SideTitles(  
128.                  showTitles: true, // Exibe os  
valores no eixo vertical.  
129.                  interval: 2, // Intervalo entre  
os valores exibidos.  
130.                  reservedSize: 40, // Espaço  
reservado para os valores.
```

```
129.                                         getTitlesWidget: (value, meta) =>
130.                                         Text(
131.                                           value.toStringAsFixed(1), //  
   Formata os valores para uma casa decimal.
132.                                         style: const  
   TextStyle(fontSize: 12),
133.                                         ),
134.                                         ),
135.                                         ),  
   // Oculta os títulos dos outros  
 eixos.  
136.                                         bottomTitles: AxisTitles(  
137.                                           sideTitles: SideTitles(  
138.                                             showTitles: false,  
139.                                             ),
140.                                             ),
141.                                             topTitles: AxisTitles(  
142.                                               sideTitles: SideTitles(  
143.                                                 showTitles: false,  
144.                                                 ),
145.                                                 ),
146.                                                 rightTitles: AxisTitles(  
147.                                                   sideTitles: SideTitles(  
148.                                                     showTitles: false,  
149.                                                     ),
150.                                                     ),
151.                                                     ),
152.                                                     borderData: FlBorderData(  
153.                                                       show: true, // Exibe a borda ao  
   redor do gráfico.  
154.                                                       border: Border.all(color: Colors.  
   black, width: 1),
155.                                                       ),
156.                                                       gridData: FlGridData(  
157.                                                         show: true, // Exibe a grade do  
   gráfico.
```

```
158.           drawVerticalLine: true, // Exibe
    linhas verticais.
159.           drawHorizontalLine: true, // Exibe
    linhas horizontais.
160.           getDrawingVerticalLine: (value) {
161.               return F1Line(
162.                   color: Colors.grey,
163.                   strokeWidth: 0.5,
164.               );
165.           },
166.           getDrawingHorizontalLine: (value) {
167.               return F1Line(
168.                   color: Colors.grey,
169.                   strokeWidth: 0.5,
170.               );
171.           },
172.           ),
173.           // Configuração dos tooltips ao tocar no
    gráfico.
174.           lineTouchData: LineTouchData(
175.               touchTooltipData:
    LineTouchTooltipData(
176.                   tooltipRoundedRadius: 8, //
    Arredondamento da tooltip.
177.                   tooltipPadding: const EdgeInsets.
    all(8), // Espaçamento interno da tooltip.
178.                   tooltipMargin: 8, // Margem da
    tooltip.
179.                   fitInsideHorizontally: true,
    // Garante que a tooltip fique dentro do gráfico
    horizontalmente.
180.                   fitInsideVertically: true,
    // Garante que a tooltip fique dentro do gráfico
    verticalmente.
181.                   tooltipBorder:
```

```
182.                                         BorderSide(color: Colors.  
183.                                         blueAccent, width: 1), // Cor da borda da tooltip.  
184.                                         getTooltipItems: (touchedSpots) {  
185.                                             if (touchedSpots == null) {  
186.                                                 return [];  
187.                                             }  
188.                                             return touchedSpots.map((spot) {  
189.                                                 return LineTooltipItem(  
190.                                                 'X:  
191.                                                 ${spot.x.toStringAsFixed(2)}\nY:  
192.                                                 ${spot.y.toStringAsFixed(2)}',  
193.                                         const TextStyle(color:  
194.                                         Colors.white), // Estiliza o texto da tooltip.  
195.                                         );  
196.                                         }).toList();  
197.                                         },  
198.                                         ),  
199.                                         ),  
200.                                         ),  
201.                                         const SizedBox(height: 20), // Espaçamento  
entre o gráfico e os valores numéricos.  
202.  
203.                                         // Exibe os valores atuais do acelerômetro  
para os eixos X, Y e Z.  
204.                                         Row(  
205.                                         mainAxisAlignment: MainAxisAlignment.  
spaceEvenly,  
206.                                         children: [  
207.                                         Text('X: ${x.toStringAsFixed(2)}',  
208.                                         style: const TextStyle(fontSize: 18,  
color: Colors.red)),
```

```
209.           Text(`Y: ${y.toStringAsFixed(2)}»,  
210.             style: const TextStyle(fontSize: 18,  
211.               color: Colors.green)),  
212.             Text(`Z: ${z.toStringAsFixed(2)}»,  
213.               style: const TextStyle(fontSize: 18,  
214.                 color: Colors.blue)),  
215.           ],  
216.         ),  
217.       ),  
218.     );  
219.   }  
220. }
```

Na figura a seguir, a tela do aplicativo que realiza a leitura do acelerômetro e plota as medidas no gráfico é representada.



**Figura 5.7 – Tela do aplicativo com as medidas coletadas com o acelerômetro.**

## Limitações do acelerômetro

Embora o acelerômetro seja versátil, ele tem algumas limitações:

- **precisão** – pequenos movimentos ou vibrações podem causar leituras inconsistentes;
- **consumo de energia** – uso contínuo pode aumentar o consumo de bateria;
- **necessidade de calibração** – mudanças de temperatura ou condições podem afetar a precisão.

## Diferenças entre acelerômetro e giroscópio

As diferenças entre acelerômetro e giroscópio são indicadas a seguir.

O acelerômetro:

- mede aceleração linear (movimento em linha reta);
- detecta mudanças de posição (inclinação).

O giroscópio:

- mede rotação e orientação angular;
- complementa o acelerômetro em aplicativos que exigem detecção precisa de movimentos rotacionais.

## O futuro do acelerômetro

Com a evolução dos sensores MEMS, o acelerômetro continuará a desempenhar um papel importante em tecnologias emergentes, como:

- IoT (Internet das Coisas) – sensores conectados para monitoramento remoto em dispositivos inteligentes;
- veículos autônomos – complemento em sistemas de navegação e estabilização;
- wearables – avanços em saúde e fitness com sensores mais precisos.

O acelerômetro é um componente essencial nos smartphones modernos, permitindo uma ampla gama de funcionalidades que vão além da detecção de movimento. Ele transforma dispositivos móveis em ferramentas dinâmicas e interativas, desempenhando um papel vital em jogos, saúde, realidade aumentada e muito mais. Com a integração em plataformas como Flutter, desenvolvedores podem facilmente criar aplicativos que aproveitam ao máximo esse sensor incrível.

## Wi-Fi

O Wi-Fi é essencial nos smartphones para realizar conexão com a internet. Porém, o Flutter não possui uma biblioteca nativa para lidar diretamente com essas funcionalidades. Para contornar isso, podemos usar pacotes de terceiros que realizam tarefas como verificar o status da conexão, listar pontos de acesso disponíveis, conectar a um Wi-Fi específico ou obter informações sobre a rede atual.

Para obter informações sobre a rede Wi-Fi, utilizaremos as dependências WiFi\_iot, network info plus e permission handler, conforme mostrado na figura a seguir. Essas dependências permitem obter informações como SSID, Endereço IP e Endereço Mac.

```
35 # The following adds the Cupertino Icons font to your application.
36 # Use with the CupertinoIcons class for iOS style icons.
37 cupertino_icons: ^1.0.8
38 wifi_iot: ^0.3.0
39 network_info_plus: ^4.0.3
40 permission_handler: ^10.3.0
41 dev_dependencies:
42   flutter_test:
43     sdk: flutter
```

**Figura 5.8** – Dependências adicionadas ao arquivo pubspec.yaml.

A seguir, é possível visualizar o código de um aplicativo escanear redes Wi-Fi.

## Código do aplicativo para varrer redes Wi-Fi

```
1. import 'package:flutter/material.dart'; // Importa os
   widgets e funcionalidades do Flutter.
2. import 'package:wifi_iot/wifi_iot.dart'; // Importa o pacote
   para acessar redes Wi-Fi no dispositivo.
3. import 'package:permission_handler/permission_handler.
   dart'; // Importa o pacote para gerenciar permissões.
4.
5. void main() => runApp(const MyApp()); // Função principal
   que inicia o aplicativo.
6.
7. /// Classe principal do aplicativo
8. class MyApp extends StatelessWidget {
9.   const MyApp({super.key});
10.
11.   @override
12.   Widget build(BuildContext context) {
13.     return MaterialApp(
14.       debugShowCheckedModeBanner: false, // Remove a
   faixa de “debug” no app.
15.       home: const WiFiScannerScreen(), // Define a tela
   inicial do app.
16.     );
17.   }
18. }
19.
20. /// Tela para escanear redes Wi-Fi disponíveis.
21. class WiFiScannerScreen extends StatefulWidget {
22.   const WiFiScannerScreen({super.key});
23.
24.   @override
25.   State<WiFiScannerScreen> createState() =>
   _WiFiScannerScreenState();
26. }
27.
```

```
28. /// Estado da tela de escaneamento de redes Wi-Fi.
29. class _WiFiScannerScreenState extends
   State<WiFiScannerScreen> {
30.   List<WifiNetwork?> networks = []; // Lista para armazenar
   as redes Wi-Fi encontradas.
31.
32.   @override
33.   void initState() {
34.     super.initState();
35.     scanNetworks(); // Inicia a varredura das redes Wi-Fi
   assim que a tela é carregada.
36.   }
37.
38.   /// Método assíncrono para escanear redes Wi-Fi
   próximas.
39.   Future<void> scanNetworks() async {
40.     // Solicita permissão de localização, necessária para
   escanear redes Wi-Fi.
41.     if (await Permission.location.request().isGranted) {
42.       try {
43.         final wifiNetworks = await WiFiForIoTPlugin.
   loadWifiList(); // Obtém a lista de redes Wi-Fi disponíveis.
44.         setState(() {
45.           networks = wifiNetworks; // Atualiza a lista de
   redes na interface.
46.         });
47.       } catch (e) {
48.         print("Erro ao escanear redes Wi-Fi: $e"); //
   Captura e exibe erros no console.
49.       }
50.     } else {
51.       print("Permissão de localização negada."); //
   Informa caso a permissão seja negada.
52.     }
53.   }
```

```
54.  
55. @override  
56. Widget build(BuildContext context) {  
57.   return Scaffold(  
58.     appBar: AppBar(  
59.       title: const Text("Scanner de Redes Wi-Fi"), //  
    Título da AppBar.  
60.       backgroundColor: Colors.red, // Define a cor da  
    AppBar.  
61.       actions: [  
62.         IconButton(  
63.           icon: const Icon(Icons.refresh), // Ícone para  
    atualizar a lista de redes.  
64.           onPressed: scanNetworks, // Ao pressionar,  
    reescaneia as redes disponíveis.  
65.         ),  
66.       ],  
67.     ),  
68.     body: networks.isEmpty  
69.       ? const Center(child: Text("Nenhuma rede  
    encontrada.")) // Exibe uma mensagem caso nenhuma rede  
    seja encontrada.  
70.       : ListView.builder(  
71.         itemCount: networks.length, // Define o  
    número de itens na lista.  
72.         itemBuilder: (context, index) {  
73.           final network = networks[index]; // Obtém  
    a rede correspondente ao índice.  
74.           return ListTile(  
75.             title: Text(network?.ssid ?? "SSID  
    Desconhecido"), // Exibe o nome da rede Wi-Fi (SSID).  
76.             subtitle: Text("Sinal: ${network?.level  
    ?? 'N/A'} dBm"), // Exibe o nível do sinal da rede Wi-Fi.  
77.           );  
78.         },  
79.       ),
```

```
80.      );
81.    }
82. }
```

Na figura a seguir, é possível visualizar a tela do aplicativo que detecta redes Wi-Fi.



scantail/Getty Images

Figura 5.9 – Tela do aplicativo Scanner Wi-Fi.

## Bluetooth

O bluetooth é uma tecnologia amplamente usada para conectar dispositivos sem fio de curto alcance, como fones de ouvido, dispositivos IoT e sensores. No Flutter, é possível implementar funcionalidades de bluetooth para criar aplicativos que interajam com outros dispositivos.

### Aplicações do bluetooth em Flutter

- **IoT e automação:** controle de dispositivos domésticos, como lâmpadas inteligentes ou sistemas de segurança.
- **Transmissão de dados:** troca de informações entre dispositivos, como transferência de arquivos ou mensagens.

- **Saúde e fitness:** comunicação com dispositivos como monitores de frequência cardíaca ou rastreadores de atividades.
- **Jogos multiplayer:** conexão entre dispositivos para experiências de jogos locais.
- **Audio streaming:** conexão com fones de ouvido ou sistemas de som via bluetooth.

## Principais bibliotecas para bluetooth no Flutter

### 1. flutter\_blue\_plus

- **Descrição:** biblioteca amplamente usada para interagir com dispositivos Bluetooth Low Energy (BLE).
- **Recursos:**
  - » escanear dispositivos BLE;
  - » conectar a dispositivos BLE;
  - » ler e escrever características;
  - » receber notificações.

### 2. flutter\_bluetooth\_serial

- **Descrição:** usada para comunicação Bluetooth Classic (em oposição ao BLE).
- **Recursos:**
  - » escanear dispositivos clássicos;
  - » conexão Serial Bluetooth (SPP);
  - » comunicação de dados simples.

### 3. blue\_thermal\_printer

- **Descrição:** usada para se conectar e imprimir em impressoras térmicas bluetooth.

- **Recursos:**

- » conectar-se a impressoras térmicas via bluetooth;
- » enviar dados/textos para impressão;
- » gerenciar o estado da conexão;
- » suporte a formatos básicos de impressão (texto, QR Code, cortar papel).

Para utilizar o bluetooth com Flutter, é necessário adicionar algumas permissões no seguinte caminho: android/app/src/main/AndroidManifest, conforme mostrado a seguir.

#### Permissões no arquivo Android Manifest

```
1. <!-- Permissão para usar o Bluetooth no dispositivo -->
2. <uses-permission android:name=>android.permission.
   BLUETOOTH> />
3.
4. <!-- Permissão para gerenciar configurações do Bluetooth
   (ativar/desativar, tornar o dispositivo detectável, etc.)
   -->
5. <uses-permission android:name=>android.permission.
   BLUETOOTH_ADMIN> />
6.
7. <!-- Permissão para acessar a localização precisa do
   dispositivo, necessária para detectar dispositivos
   Bluetooth e redes Wi-Fi -->
8. <uses-permission android:name=>android.permission.ACCESS_
   FINE_LOCATION> />
9.
10. <!-- Permissão para escanear dispositivos Bluetooth ao
    redor (necessário no Android 12+) -->
11. <uses-permission android:name=>android.permission.
   BLUETOOTH_SCAN> />
12.
```

```
13. <!-- Permissão para estabelecer conexão com dispositivos  
Bluetooth (necessário no Android 12+) -->  
14. <uses-permission android:name=>android.permission.  
BLUETOOTH_CONNECT> />
```

A seguir, apresentamos um resumo das permissões necessárias para utilizar o bluetooth.

### 1. BLUETOOTH

- Permite que o aplicativo use o bluetooth, mas não concede permissões avançadas como escanear ou conectar-se a dispositivos.
- Essencial para aplicativos que necessitam de qualquer funcionalidade básica de bluetooth.

### 2. BLUETOOTH\_ADMIN

- Permite que o aplicativo ative/desative o bluetooth e altere suas configurações.
- Requerida para tornar o dispositivo detectável por outros dispositivos.

### 3. ACCESS\_FINE\_LOCATION

- Necessária para escanear dispositivos bluetooth e redes Wi-Fi no Android 6.0 (API 23) ou superior.
- O Android exige essa permissão, pois a varredura de dispositivos bluetooth pode revelar informações sobre a localização do usuário.

### 4. BLUETOOTH\_SCAN (Android 12+ - API 31)

- Permite que o aplicativo escaneie dispositivos bluetooth próximos.
- Substitui a necessidade de ACCESS\_FINE\_LOCATION para escanear dispositivos bluetooth a partir do Android 12.

### 5. BLUETOOTH\_CONNECT (Android 12+ - API 31)

- Permite que o aplicativo estabeleça conexões com dispositivos bluetooth.

- Necessária para conectar-se a dispositivos bluetooth sem precisar da permissão BLUETOOTH\_ADMIN.

A seguir, são apresentadas algumas restrições e boas práticas para utilizar bluetooth.

- No **Android 12+**, as permissões BLUETOOTH\_SCAN e BLUETOOTH\_CONNECT **substituem** BLUETOOTH\_ADMIN para escanear e conectar dispositivos.
- **Se o aplicativo precisa escanear dispositivos bluetooth no Android 11 ou inferior**, a permissão ACCESS\_FINE\_LOCATION ainda é necessária.
- **Se o aplicativo apenas se conecta a dispositivos bluetooth previamente pareados**, BLUETOOTH\_CONNECT é suficiente.

A seguir, é possível visualizar o código completo do aplicativo que escaneia dispositivos bluetooth.

Código do aplicativo de varredura de dispositivos bluetooth

```
1. import 'package:flutter/material.dart'; // Importa o
   pacote principal do Flutter para criação da interface.
2. import 'package:flutter_blue_plus/flutter_blue_plus.dart';
   // Importa a biblioteca para interação com Bluetooth.
3. import 'package:permission_handler/permission_handler.
   dart'; // Importa a biblioteca para gerenciar permissões.
4.
5. void main() {
6.   runApp(MyApp()); // Inicia o aplicativo.
7. }
8.
9. /// Classe principal do aplicativo.
10. class MyApp extends StatelessWidget {
11.   @override
12.   Widget build(BuildContext context) {
```

```
13.     return MaterialApp(
14.         title: 'Bluetooth Scanner', // Define o título do
15.             aplicativo.
16.         theme: ThemeData(primarySwatch: Colors.blue), //
17.             Define o tema com cor azul.
18.         home: BluetoothScreen(), // Define a tela inicial
19.             como a de escaneamento Bluetooth.
20.
21. /// Tela principal para escaneamento de dispositivos
22. Bluetooth.
23. class BluetoothScreen extends StatefulWidget {
24.     @override
25.     _BluetoothScreenState createState() =>
26.         _BluetoothScreenState();
27.
28. /// Estado do widget BluetoothScreen.
29. class _BluetoothScreenState extends
30.     State<BluetoothScreen> {
31.     bool isScanning = false; // Variável para controlar o
32.         estado de escaneamento.
33.
34.     @override
35.     void initState() {
36.         super.initState();
37.         _checkPermissions(); // Verifica e solicita permissões
38.             ao iniciar a tela.
39.
40.     }
41.
42.     /// Método para verificar e solicitar permissões
43.     necessárias.
44.     Future<void> _checkPermissions() async {
45.
```

```
39.      // Para Android 12+ são necessárias permissões
        específicas de Bluetooth.
40.      if (await Permission.bluetoothScan.isDenied) {
41.          await Permission.bluetoothScan.request(); //
        Solicita permissão para escanear dispositivos Bluetooth.
42.      }
43.      if (await Permission.bluetoothConnect.isDenied) {
44.          await Permission.bluetoothConnect.request(); //
        Solicita permissão para conectar a dispositivos Bluetooth.
45.      }
46.      if (await Permission.locationWhenInUse.isDenied) {
47.          await Permission.locationWhenInUse.request();
        // Solicita permissão de localização (necessária para
        escanear Bluetooth).
48.      }
49.  }
50.
51.  /// Método para iniciar o escaneamento de dispositivos
        Bluetooth.
52. void startScan() async {
53.     setState(() {
54.         isScanning = true; // Define que o escaneamento
        está ativo.
55.     });
56.     await FlutterBluePlus.startScan(timeout: const
        Duration(seconds: 5)); // Inicia a varredura por
        dispositivos Bluetooth por 5 segundos.
57.     setState(() {
58.         isScanning = false; // Finaliza o escaneamento.
59.     });
60. }
61.
62. /// Método para parar o escaneamento de dispositivos
        Bluetooth.
63. void stopScan() async {
```

```
64.      await FlutterBluePlus.stopScan(); // Para a varredura
       de dispositivos Bluetooth.
65.      setState(() {
66.          isScanning = false; // Atualiza o estado para
       indicar que o escaneamento parou.
67.      });
68.  }
69.
70. @override
71. Widget build(BuildContext context) {
72.     return Scaffold(
73.         appBar: AppBar(
74.             backgroundColor: Colors.red, // Define a cor da
       AppBar.
75.             title: const Text('Bluetooth Scanner'), // Título
       da tela.
76.             actions: [
77.                 IconButton(
78.                     icon: Icon(isScanning ? Icons.stop : Icons.
       search), // Exibe o ícone de pesquisa ou de parada
       dependendo do estado.
79.                     onPressed: isScanning ? stopScan
       : startScan, // Alterna entre iniciar e parar o
       escaneamento.
80.                 ),
81.                 ],
82.                 ),
83.                 body: StreamBuilder<List<ScanResult>>(
84.                     stream: FlutterBluePlus.scanResults, // Stream
       que recebe os resultados do escaneamento Bluetooth.
85.                     builder: (context, snapshot) {
86.                         if (snapshot.connectionState ==
       ConnectionState.waiting) {
87.                             return const Center(child:
       CircularProgressIndicator())); // Exibe um indicador de
       carregamento enquanto busca dispositivos.
```

```
88.          }
89.          final scanResults = snapshot.data ?? [];
// Obtém os dispositivos encontrados ou uma lista vazia.
90.
91.          return ListView.builder(
92.              itemCount: scanResults.length, // Define
93.              o número de itens na lista com base nos dispositivos
94.              encontrados.
95.              itemBuilder: (context, index) {
96.                  final result = scanResults[index]; // Obtém
97.                  o dispositivo correspondente ao índice.
98.                  return ListTile(
99.                      title: Text(result.device.name.isNotEmpty
100.                         ? result.device.name
101.                         : 'Dispositivo sem Nome'), // Exibe o
102.                         nome do dispositivo, se disponível.
103.                      subtitle: Text(result.device.
104.                         id.toString()), // Exibe o ID do dispositivo Bluetooth.
105.                      trailing: ElevatedButton(
106.                          onPressed: () async {
107.                              try {
108.                                  await result.device.connect(); //
109.                                  Tenta conectar ao dispositivo.
110.                                  ScaffoldMessenger.of(context).
111.                                  showSnackBar(SnackBar(
112.                                      content: Text('Conectado ao
113. ${result.device.name}'))); // Exibe mensagem de sucesso.
114.                              } catch (e) {
115.                                  ScaffoldMessenger.of(context).
116.                                  showSnackBar(SnackBar(
117.                                      content: Text(
118.                                          'Erro ao conectar ao
119. ${result.device.name}'))); // Exibe mensagem de erro caso a
120.                                         conexão falhe.
121.                              }
122.                          }
123.                      }
124.                  )
125.              )
126.          )
127.      )
128.  )
129. }
```

```
111.           },
112.           child: const Text('Conectar'), // Botão
    para conectar ao dispositivo encontrado.
113.           ),
114.           );
115.           },
116.           );
117.           },
118.           ),
119.           );
120.       }
121. }
```

Na figura a seguir, está representada a tela do aplicativo realizando a varredura de dispositivos bluetooth.



**Figura 5.10** – Tela do aplicativo bluetooth scanner.

Os principais desafios e limitações do bluetooth com Flutter são:

- **permissões** – o bluetooth requer permissões específicas, que podem variar entre Android e iOS;

- **BLE vs. bluetooth clássico** – BLE é adequado para dispositivos de baixa energia, enquanto o bluetooth clássico é usado para conexões mais robustas, como áudio;
- **conexões simultâneas** – pode ser difícil gerenciar várias conexões ao mesmo tempo;
- **compatibilidade** – certifique-se de que o dispositivo é compatível com BLE se estiver usando o flutter\_blue\_plus.

A seguir, listamos alguns casos de uso reais.

- **Aplicativos IoT** – conexão com sensores de temperatura, dispositivos médicos ou automação residencial.
- **Wearables** – comunicação com relógios inteligentes ou rastreadores de fitness.
- **Impressoras térmicas** – conexão bluetooth para imprimir recibos ou relatórios.

O bluetooth no Flutter, quando combinado com bibliotecas como flutter\_blue\_plus, permite criar aplicativos robustos que interagem com dispositivos próximos de maneira eficiente. Com o crescente uso de dispositivos IoT e wearables, dominar o bluetooth em Flutter abre portas para aplicações inovadoras e dinâmicas.

## Atividade

Nesta atividade, o objetivo é criar um aplicativo de turismo interativo com GPS e áudio.

**Desenvolva um aplicativo Flutter que utiliza GPS para identificar a localização do usuário e que reproduz áudios personalizados quando o usuário se aproxima de pontos turísticos predeterminados. O aplicativo será voltado para turistas e incluirá uma interface simples e intuitiva.**

Os objetivos de aprendizagem são:

- **Tecnológicos**

- » entender o uso do GPS em aplicativos Flutter utilizando a biblioteca geolocator;
- » integrar áudios no aplicativo com a biblioteca audioplayers;
- » gerenciar permissões necessárias (localização e mídia).

- **Didáticos**

- » planejar a lógica de funcionamento de um aplicativo baseado em localização;
- » criar uma interface intuitiva para o usuário;
- » testar e depurar um aplicativo em um dispositivo físico.

- **Sociais** – estimular a criatividade na escolha de pontos turísticos e na personalização dos áudios.

As etapas da atividade estão descritas a seguir.

1. **Configuração do projeto**

- Configurar o ambiente Flutter: criar um projeto Flutter. Adicionar as dependências necessárias no arquivo pubspec.yaml, conforme indicado a seguir.

Dependências adicionadas ao projeto

1. dependencies:
2. geolocator: ^10.0.0
3. audioplayers: ^3.0.0
4. permission\_handler: ^10.0.0

- Incluir as permissões necessárias para utilizar o GPS, conforme mostrado a seguir.

Permissões para utilizar o GPS

1. <uses-permission android:name=>android.permission.ACCESS\_FINE\_LOCATION</>
2. <uses-permission android:name=>android.permission.ACCESS\_COARSE\_LOCATION>
- 3.

- Construir a lista dos pontos turísticos, conforme mostrado a seguir.

Programação da lista de pontos turísticos

```
1. final List<Map<String, dynamic>>
  pointsOfInterest = [
2.   {
3.     "name": "Museu de História",
4.     "latitude": -22.91216,
5.     "longitude": -47.06288,
6.     "audioPath": "assets/audio/museu.
  mp3",
7.   },
8.   {
9.     "name": "Praça Central",
10.    "latitude": -22.91150,
11.    "longitude": -47.06330,
12.    "audioPath": "assets/audio/praca.
  mp3",
13.   },
14. ];
```

- Programar as permissões de identificação da localização atual, os cálculos de distância entre dois pontos e a associação dos pontos turísticos aos áudios, conforme apresentado a seguir.

Permissão para acessar a localização

```
1. Future<void> checkPermissions() async {
2.   if (await Permission.location.isDenied)
3.   {
4.     await Permission.location.request();
5.   }
```

Código para obter a localização atual e calcular  
a distância entre dois pontos

```
1. Future<Position> getCurrentPosition()  
    async {  
2.     return await Geolocator.  
        getCurrentPosition(  
3.         desiredAccuracy: LocationAccuracy.  
            high,  
4.         );  
5.     }  
6.  
7. double calculateDistance(lat1, lon1,  
    lat2, lon2) {  
8.     return Geolocator.  
        distanceBetween(lat1, lon1, lat2, lon2);  
9. }  
10.  
11.
```

Código para associar cada áudio ao destino

```
1. final AudioPlayer audioPlayer =  
    AudioPlayer();  
2.  
3. Future<void> playAudio(String audioPath)  
    async {  
4.     await audioPlayer.  
        play(AssetSource(audioPath));  
5. }  
6.  
7. void checkProximity(Position  
    userPosition) {  
8.     for (var point in pointsOfInterest) {  
9.         double distance = calculateDistance(  
10.             userPosition.latitude,
```

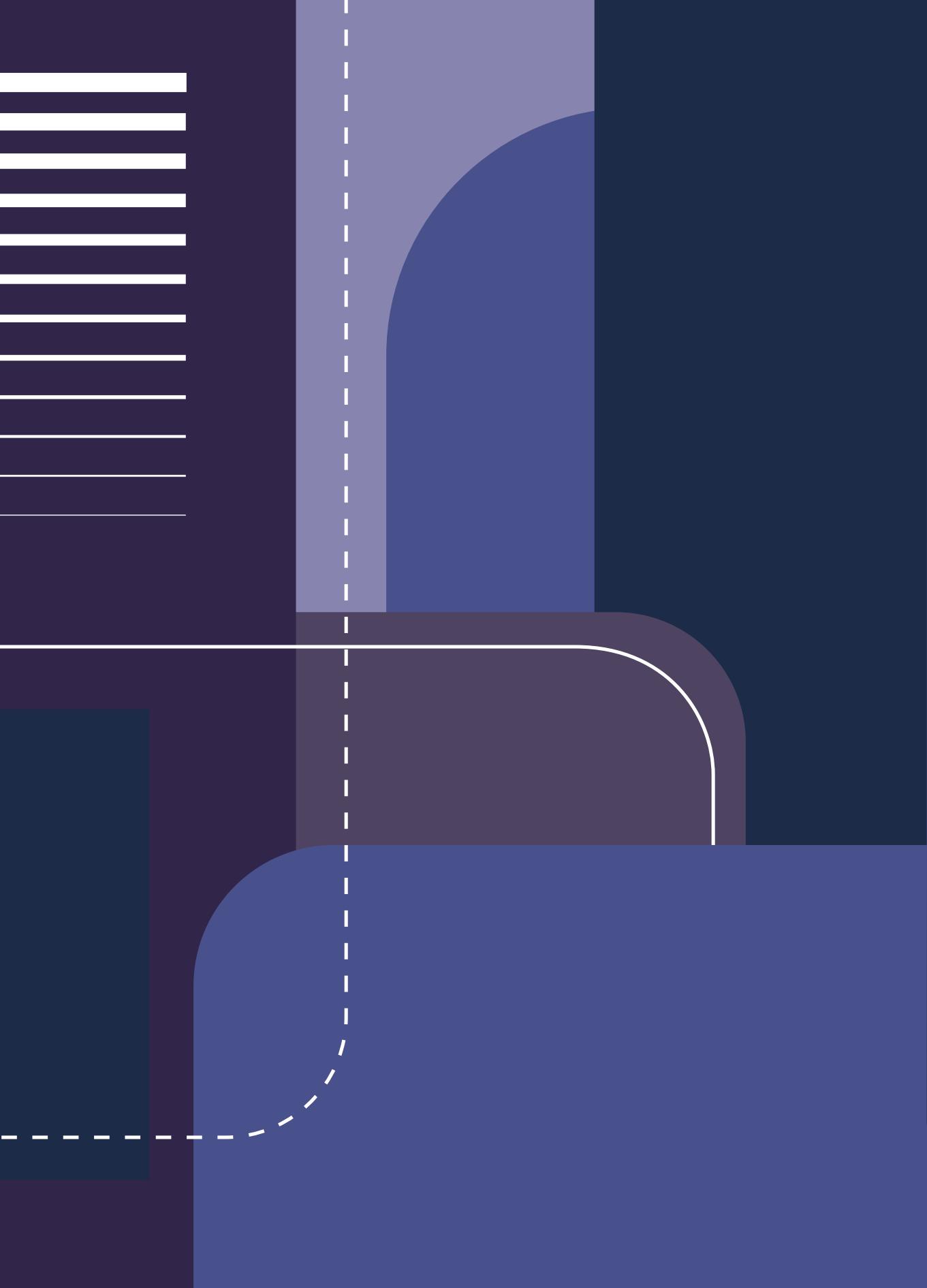
```
11.         userPosition.longitude,  
12.         point[«latitude»],  
13.         point[«longitude»],  
14.     );  
15.     if (distance < 50) { // Menos de 50  
16.         metros  
17.         playAudio(point[“audioPath”]);  
18.     break;  
19. }  
20. }  
21.  
22.
```

## 2. Critérios de avaliação

- Funcionamento correto do GPS e reprodução de áudio.
- Interface funcional e intuitiva.
- Código organizado e legível.

# Conclusão

Este capítulo apresentou conceitos sobre recursos de hardware do smartphone, como câmera, Wi-Fi, bluetooth, acelerômetro, GPS e áudio, bem como tratou das dependências necessárias na criação de projetos para dispositivos móveis.





## CAPÍTULO 6

# PUBLICAÇÃO DO APLICATIVO

### Introdução

**N**o desenvolvimento de aplicativos móveis, a compilação e a publicação são etapas fundamentais que conectam a criação do software à sua entrega ao usuário final. Esse processo não é apenas técnico, mas também representa o momento em que sua ideia ganha vida nas mãos dos usuários.

Imagine um aplicativo que foi cuidadosamente projetado, com interfaces intuitivas e funcionalidades robustas. Para que ele alcance seu público, é necessário que passe por etapas críticas, como configuração de ícones, escolha de nomes e otimização para diferentes plataformas. Além disso, os aplicativos modernos precisam de identidade própria, como ícones que sejam visualmente marcantes e alinhados à proposta do projeto.

Alterar o ícone de um aplicativo não é apenas uma questão estética; é também estratégica. É a primeira coisa que o usuário vê ao procurar pelo app em sua tela inicial. Neste capítulo, aprenderemos como ajustar esses elementos para garantir que seu aplicativo seja único e atraente no mercado.

Também exploraremos os passos necessários para compilar e publicar aplicativos nas lojas oficiais, como Google Play Store e Apple App Store. Essas etapas envolvem preparações técnicas e administrativas que asseguram a qualidade e o alcance do seu aplicativo.

Ao final deste capítulo, você terá uma compreensão clara sobre como configurar elementos visuais como ícones, preparar o aplicativo para distribuição e realizar as etapas necessárias para disponibilizá-lo aos seus usuários. Isso garantirá que o aplicativo não apenas funcione bem, mas também ofereça uma experiência marcante desde o primeiro contato.

## Como alterar o ícone do aplicativo?

Alterar o ícone do aplicativo envolve configurações específicas para cada plataforma, garantindo que ele seja exibido corretamente em diferentes dispositivos e resoluções. Essa personalização não apenas melhora a experiência do usuário, mas também aumenta as chances de seu aplicativo se destacar em um mercado competitivo. Um ícone genérico ou mal projetado pode passar uma impressão de falta de profissionalismo, enquanto um ícone exclusivo e bem elaborado pode atrair mais downloads e engajamento.

No **Android**, os ícones são armazenados em diferentes diretórios dentro da pasta `android/app/src/main/res/`, pois cada dispositivo pode ter uma densidade de tela diferente.

O Flutter gera os seguintes arquivos:

- **mipmap-mdpi** – ícones para telas de baixa densidade ( $48 \times 48$  px);
- **mipmap-hdpi** – ícones para telas de média densidade ( $72 \times 72$  px);

- **mipmap-xhdpi** – ícones para telas de alta densidade ( $96 \times 96$  px);
- **mipmap-xxhdpi** – ícones para telas de altíssima densidade ( $144 \times 144$  px);
- **mipmap-xxxhdpi** – ícones para telas de ultra-alta densidade ( $192 \times 192$  px).

Após a geração dos ícones, o arquivo AndroidManifest.xml (localizado em android/app/src/main/AndroidManifest.xml) é automaticamente atualizado para incluir a referência ao novo ícone.

```
<application
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
    ...>
</application>
```

No sistema operacional **iOS**, os ícones são gerados dentro do arquivo de assets Assets.xcassets, que é um diretório no projeto iOS: ios/Runner/Assets.xcassets/AppIcon.appiconset/.

Diferentemente do Android, o iOS requer ícones em vários tamanhos específicos para diferentes dispositivos e resoluções, conforme o quadro a seguir.

#### Quadro 6.1 – Dispositivos iOS e respectivos tamanhos de ícones

<b>iPhone Notification</b>	$40 \times 40$ px
<b>iPhone Spotlight</b>	$60 \times 60$ px
<b>iPhone App Icon</b>	$120 \times 120$ px
<b>iPad App Icon</b>	$152 \times 152$ px
<b>iPad Pro App Icon</b>	$167 \times 167$ px
<b>App Store Icon</b>	$1024 \times 1024$ px

Com base no exposto, explicaremos os comandos necessários para alterar o ícone do aplicativo.

Esses ícones são gerados prontamente pelo flutter\_launcher\_icons e armazenados no arquivo Contents.json dentro da pasta AppIcon.appiconset. Esse arquivo contém metadados sobre cada ícone gerado.

No iOS, o Info.plist (localizado em ios/Runner/Info.plist) contém a referência para os ícones do app.

```
<key>CFBundleIcons</key>
<dict>
    <key>CFBundlePrimaryIcon</key>
    <dict>
        <key>CFBundleIconFiles</key>
        <array>
            <string>AppIcon</string>
        </array>
        <key>UIPrerenderedIcon</key>
        <false/>
    </dict>
</dict>
```

O primeiro passo é adicionar o pacote flutter\_launcher\_icons ao arquivo pubspec.yaml, conforme mostrado a seguir.

Dependência flutter\_launcher no arquivo pubspec.yaml

```
# The following adds the Cupertino Icons font to your
# application.
# Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^1.0.8
firebase_core: ^3.4.0
cloud_firestore: ^5.4.0
uuid: ^3.0.7
flutter_launcher_icons: ^0.10.0
dev_dependencies:
  flutter_test:
    sdk: flutter
```

Ainda no arquivo pubspec.yaml, precisamos adicionar outra configuração para o ícone, como indicado a seguir.

#### Configuração para o ícone

```
flutter_icons:  
  android: true  
  image_path: assets/icon/icon.png  
  # To add assets to your application, add an assets section,  
  # like this:  
  # assets:  
  #   - images/a_dot_burr.jpeg  
  #   - images/a_dot_ham.jpeg
```

Veja, a seguir, a configuração realizada no pubspec.yaml para alterar o ícone do aplicativo.

#### Configuração para alterar o ícone do aplicativo

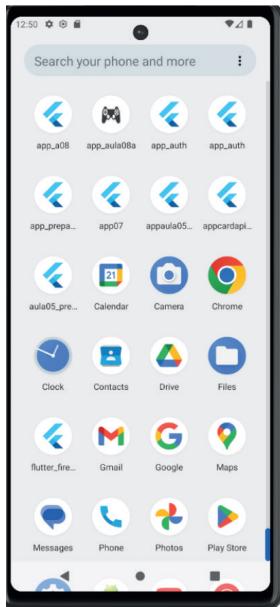
1. flutter\_icons:
2. android: true
3. image\_path
4. android: Define se o ícone será aplicado em aplicativos **Android**.
5. ios: Define se o ícone será aplicado em aplicativos iOS.
6. image\_path: Especifica o caminho da imagem do ícone que será usada (substitua “assets/icons/app\_icon.png” pelo caminho correto da sua imagem).
- 7.

Com a configuração realizada, o passo seguinte é criar a pasta asset/icon e colocar a imagem que será utilizada como ícone dentro dela. Então, poderemos alterar o ícone conforme comando mostrado a seguir.

#### Comando para alterar o ícone do app

1. flutter pub run flutter\_launcher\_icons: main

Na figura a seguir, podemos visualizar o ícone com a imagem alterada.



**Figura 6.1 – Aplicativo com o ícone alterado.**

## Compilação

A compilação em Flutter é o processo de transformar o código Dart e os recursos do projeto em um formato executável para a plataforma alvo, seja Android, iOS, web ou desktop. Aqui, abordaremos os principais passos para compilar o aplicativo.

### Configuração prévia

Antes de compilar, é essencial garantir que:

- o Flutter SDK está corretamente instalado e configurado;
- o ambiente de desenvolvimento está preparado, com ferramentas como Android Studio (ou VS Code) e Xcode (para iOS).

**SAIBA MAIS**

Nos links e QR Codes a seguir, é possível acessar os sites das ferramentas.

**Android Studio**

disponível em: <https://developer.android.com/studio?hl=pt-br>. Acesso em: 21 ago. 2025.

**VSCODE**

disponível em: <https://code.visualstudio.com/download>. Acesso em: 21 ago. 2025.

**SDK do Dart**

disponível em: <https://dart.dev/get-dart>.  
Acesso em: 21 ago. 2025.

**XCode**

disponível em: <https://developer.apple.com/documentation/safari-developer-tools/installing-xcode-and-simulators>. Acesso em: 21 ago. 2025.

## Modos de compilação

Existem três modos de compilação para um aplicativo Flutter (debug, profile e release), que descreveremos a seguir.

### Debug

Usado para desenvolvimento e testes, com suporte a ferramentas como Hot Reload e Debugging. Para executar o projeto Flutter, usa-se o comando apresentado a seguir.

#### Comando para executar o projeto Flutter

1. flutter run - **Modo padrão Flutter**, usado para desenvolvimento com hot reload e depuração ativada

- 2.
3. flutter run -d <device\_id>. - Para rodar em um dispositivo específico:
- 4.
5. flutter emulators --launch <emulator\_name> - Para rodar no emulador Android
- 6.
7. open -a Simulator && flutter run -d ios - Para rodar no simulador IOS

## Profile

Otimiza o desempenho do aplicativo enquanto mantém algumas ferramentas de análise. Para executar o projeto Flutter com ferramentas de análise, utilizamos o comando indicado a seguir.

Comando para executar o projeto Flutter com ferramentas de análise

1. flutter build apk -profile. - O modo profile é usado para medir desempenho, mantendo algumas ferramentas de depuração ativas.
- 2.
3. Esse comando compila o projeto Flutter para a plataforma iOS no modo debug. Isso significa que ele gera um build com recursos de depuração ativados, permitindo inspeção do código, logs detalhados e uso do hot reload.
- 4.
5. flutter build ios -debug
6. flutter build ios → Compila o projeto para iOS.
7. --debug → Especifica que o build será gerado no modo de depuração.
- 8.
9. flutter build ios -profile

## Release

Executa a compilação final otimizada para publicação. A seguir, é possível visualizar o comando utilizado para gerar o apk de forma otimizada para instalação.

Comando para gerar o apk de forma otimizada para instalação

1. flutter build apk --release

Após gerar o apk, é possível encontrá-lo no caminho \build\app\outputs\flutter-apk, conforme indicado na figura a seguir.

```
PS C:\Users\Eng. Daniel Vieira\Desktop\Aplicativos_Flutter\app_ex_aula10> flutter build apk --release
A new version of Flutter is available!
To update to the latest version, run "flutter upgrade".
Font asset "MaterialIcons-Regular.otf" was tree-shaken, reducing it from 1645184 to 1408 bytes (99.9% reduction). Tree-shaking can be disabled by providing the --no-tree-shake-icons flag when building your app.
Formato de pacotes inválido.
Formato de pacote apk: "apk_release"...
Buildfile: C:\Users\Eng. Daniel Vieira\Desktop\Aplicativos_Flutter\app_ex_aula10\build\app\apk\flutter_apk\app_release.apk (18.098).
PS C:\Users\Eng. Daniel Vieira\Desktop\Aplicativos_Flutter\app_ex_aula10> [REDACTED]
```

**Figura 6.2 – Caminho do apk gerado.**

A seguir, é possível visualizar o comando que gera o aplicativo em formato otimizado para disponibilização na Play Store.

Comando para gerar o aplicativo de forma otimizada

1. flutter build appbundle

A seguir, é possível visualizar o comando que gera o aplicativo para iOS.

Comando que gera o aplicativo para iOS

1. flutter build ios --release

- 2.

O Flutter gera os arquivos dentro da pasta **iOS/** do seu projeto. Para abrir e rodar no simulador ou dispositivo real, você precisa usar o Xcode.

## Distribuição

Distribuir um aplicativo envolve publicá-lo em plataformas como a Google Play Store e a Apple App Store. Aqui está um guia passo a passo para ambas.

### Distribuição no Android

Gerar o APK ou AAB

A seguir, é possível visualizar o comando que gera o aplicativo em formato otimizado para disponibilização na Play Store, como visto anteriormente.

Comando que gera o aplicativo de forma otimizada

```
1. flutter build appbundle
```

Antes da publicação, é necessário certificar-se de que o arquivo gerado está assinado corretamente.

Outro aspecto importante que vale destacar é que, além do cadastro como desenvolvedor, será necessário pagar uma taxa anual para disponibilizar o aplicativo na Play Store.

Para publicar, crie uma conta de desenvolvedor na Google Play. Após realizar o cadastro de desenvolvedor, as seguintes etapas serão necessárias:

- configuração das informações do app, como nome, descrição, screenshots e classificação etária;
- envio do AAB na aba “Versões de Produção”;
- preenchimento dos dados de política, privacidade e conteúdo.

Então, basta aguardar a aprovação.

## Distribuição no iOS

### Configuração no Xcode

O próximo passo é certificar-se de que o projeto está configurado com um perfil de provisionamento válido.

As próximas etapas do processo estão descritas a seguir.

#### 1. TestFlight

Use o TestFlight para testes com beta testers antes da publicação.

#### 2. Publicação na App Store

- Acesse o App Store Connect;
- Configure as informações do app, como ícone, screenshots e descrição;
- Envie a versão compilada usando o Xcode ou o comando flutter build ios.

#### 3. Dicas para sucesso na distribuição

- Revise os guias de publicação oficiais do Google e da Apple;
- Otimize o SEO do aplicativo nas lojas;
- Monitore as métricas e colete feedback dos usuários para futuras atualizações.

Assim como na Play Store, para publicar aplicativos na App Store, é necessário inscrever-se no Apple Developer Program, que possui uma taxa anual. Essa inscrição permite que desenvolvedores individuais e empresas distribuam seus aplicativos na plataforma da Apple. Além disso, a Apple disponibiliza o Programa para Pequenas Empresas da App Store, que oferece uma taxa de comissão reduzida em apps pagos e compras dentro do app para desenvolvedores com receita abaixo de um milhão de dólares. Organizações sem fins lucrativos, instituições educacionais e entidades governamentais podem se qualificar para isenção dessa taxa anual.

## Conclusão

Neste capítulo, foram apresentados conceitos sobre compilação de aplicativos mobile, distribuição, alteração de ícones e procedimentos para realizar a publicação nas principais lojas de aplicativos para smartphones (Play Store e Apple Store).

# REFERÊNCIAS

ALMEIDA, Jhoisnáyra Vitória Rodrigues de. Como obter dados da internet no Flutter usando HTTP. **Alura**, 9 fev. 2023. Disponível em: <https://www.alura.com.br/artigos/obter-dados-internet-flutter-usando-http>. Acesso em: 6 ago. 2025.

APPLE Developer Program. **Developer**, [2025]. Disponível em: <https://developer.apple.com/programs/>. Acesso em: 14 ago. 2025.

APPLE Store Connect. Disponível em: <https://appstoreconnect.apple.com/access/users>. Acesso em: 14 ago. 2025.

BUILD and release an Android app. **Flutter Docs**, 7 abr. 2025. Disponível em: <https://docs.flutter.dev/deployment/android>. Acesso em: 14 ago. 2025.

CONFIGURAR a API Maps JavaScript. **Google Maps Platform**, 8 jun. 2025. Disponível em: <https://developers.google.com/maps/documentation/javascript/get-api-key?hl=pt-br>. Acesso em: 13 ago. 2025.

CONTROLAR versões do seu app. **Developers**, 23 ago. 2024. Disponível em: <https://developer.android.com/studio/publish/versioning?hl=pt-br>. Acesso em: 13 ago. 2025.

COSTA, Félix. Diferença entre API e Web Service de maneira simples. **New Blog**(“Felix”,@fxcosta”);, 31 maio 2015. Disponível em: <https://fxcosta.wordpress.com/2015/05/31/diferenca-entre-api-e-web-service-de-maneira-simples/>. Acesso em: 6 ago. 2025.

COSTA, Laís. Qual a diferença entre API e web service? **Blog idwall**, 25 ago. 2020. Disponível em: <https://blog.idwall.co/qual-a-diferenca-entre-api-e-web-service/>. Acesso em: 6 ago. 2025.

DRIFT 2.28.1. **Pub.dev**, [jul. 2025]. Disponível em: <https://pub.dev/packages/drift>. Acesso em: 11 ago. 2025.

FLUTTER databases overview – updated 2025. **Green Robot**, [2025]. Disponível em: <https://greenrobot.org/database/flutter-databases-2023-overview/>. Acesso em: 11 ago. 2025.

FLUTTER docs. **Flutter documentation**. Disponível em: <https://docs.flutter.dev/>. Acesso em: 25 ago. 2025.

FLUTTER\_BLUE\_PLUS 1.35.5. **Pub.dev**, maio 2025. Disponível em: [https://pub.dev/packages/flutter\\_blue\\_plus](https://pub.dev/packages/flutter_blue_plus). Acesso em: 13 ago. 2025.

FLUTTER\_LAUNCHER\_ICONS 0.14.4. **Pub.dev**, jun. 2025. Disponível em: [https://pub.dev/packages/flutter\\_launcher\\_icons](https://pub.dev/packages/flutter_launcher_icons). Acesso em: 13 ago. 2025.

GOOGLE Play Console. Disponível em: <https://play.google.com/console/u/0/signup>. Acesso em: 14 ago. 2025.

HIVE 2.2.3. **Pub.dev**, [2022]. Disponível em: <https://pub.dev/packages/hive>. Acesso em: 11 ago. 2025.

ÍCONES adaptativos. **Developers**, 27 jul. 2025. Disponível em: [https://developer.android.com/develop/ui/views/launch/icon\\_design\\_adaptive?hl=pt\\_br](https://developer.android.com/develop/ui/views/launch/icon_design_adaptive?hl=pt_br). Acesso em: 13 ago. 2025.

INSTANTLY share code, notes, and snippets. **GitHub Gist**, 2019. Disponível em: <https://gist.github.com/blehr/d39288ca9640de7a98b02d9d0b493959>. Acesso em: 13 ago. 2025.

LIMA, Ricarth. Flutter: como persistir dados e quais ferramentas usar. **Alura**, 3 set. 2024. Disponível em: <https://www.alura.com.br/artigos/alternativas-de-persistencia-de-dados-com-flutter>. Acesso em: 11 ago. 2025.

MARCUS, Luiz. 5 Livros sobre Desenvolvimento de Apps com Flutter, 18 dez. 2020. Disponível em: <https://luizmarcus.com/android/5-livros-sobre-desenvolvimento-de-apps-com-flutter/>. Acesso em: 25 ago. 2025.

NETWORK\_INFO\_PLUS 6.1.4. **Pub.dev**, mar. 2025. [https://pub.dev/packages/network\\_info\\_plus](https://pub.dev/packages/network_info_plus). Acesso em: 13 ago. 2025.

NODE.JS. Disponível em: <https://nodejs.org/pt>. Acesso em: 6 ago. 2025.

OHASHI, Orlando. App Flutter utilizando a câmera e/ou acessando a galeria. **Medium**, 11 jul. 2020. Disponível em: <https://medium.com/@orlandoohashi/app-flutter-utilizando-a-camera-e-ou-acessando-a-galeria-647ad6237d77>. Acesso em: 13 ago. 2025.

QUAL é a diferença entre o SOAP e o REST? **Amazon – AWS**, [2024]. Disponível em: <https://aws.amazon.com/pt/compare/the-difference-between-soap-rest/>. Acesso em: 6 ago. 2025.

SEND data to the internet. **Flutter Docs**, 12 fev. 2025. Disponível em: <https://docs.flutter.dev/cookbook/networking/send-data>. Acesso em: 6 ago. 2025.

SENSORS\_PLUS 6.1.2. **Pub.dev**, 11 ago. 2025. Disponível em: [https://pub.dev/packages/sensors\\_plus](https://pub.dev/packages/sensors_plus). Acesso em: 13 ago. 2025.

SHARED Preferences no Flutter para armazenar dados. **Nine Labs**, 18 jan. 2023. Disponível em: [http://ninelabs.blog/shared-preferences-flutter-armazenar-dados/#google\\_vignette](http://ninelabs.blog/shared-preferences-flutter-armazenar-dados/#google_vignette). Acesso em: 11 ago. 2025.

SHARED\_PREFERENCES 2.5.3. **Pub.dev**, [abr. 2025]. Disponível em: [https://pub.dev/packages/shared\\_preferences](https://pub.dev/packages/shared_preferences). Acesso em: 11 ago. 2025.

SQFLITE 2.4.2. **Pub.dev**, [mar. 2025]. Disponível em: <https://pub.dev/packages/sqlite>. Acesso em: 11 ago. 2025.

TAKE a picture using the câmera. **Flutter Docs**, 19 maio 2025. Disponível em: <https://docs.flutter.dev/cookbook/plugins/picture-using-camera>. Acesso em: 13 ago. 2025.

TEIXEIRA, Giovany Frossar *et al.* **Fundamentos de Flutter e Dart para Desenvolvimento de Apps Móveis**. Vitória, Edifes, 2024.

WIFI\_IOT 0.3.19+2. **Pub.dev**, mar. 2025. Disponível em: [https://pub.dev/packages/wifi\\_iot](https://pub.dev/packages/wifi_iot). Acesso em: 13 ago. 2025.

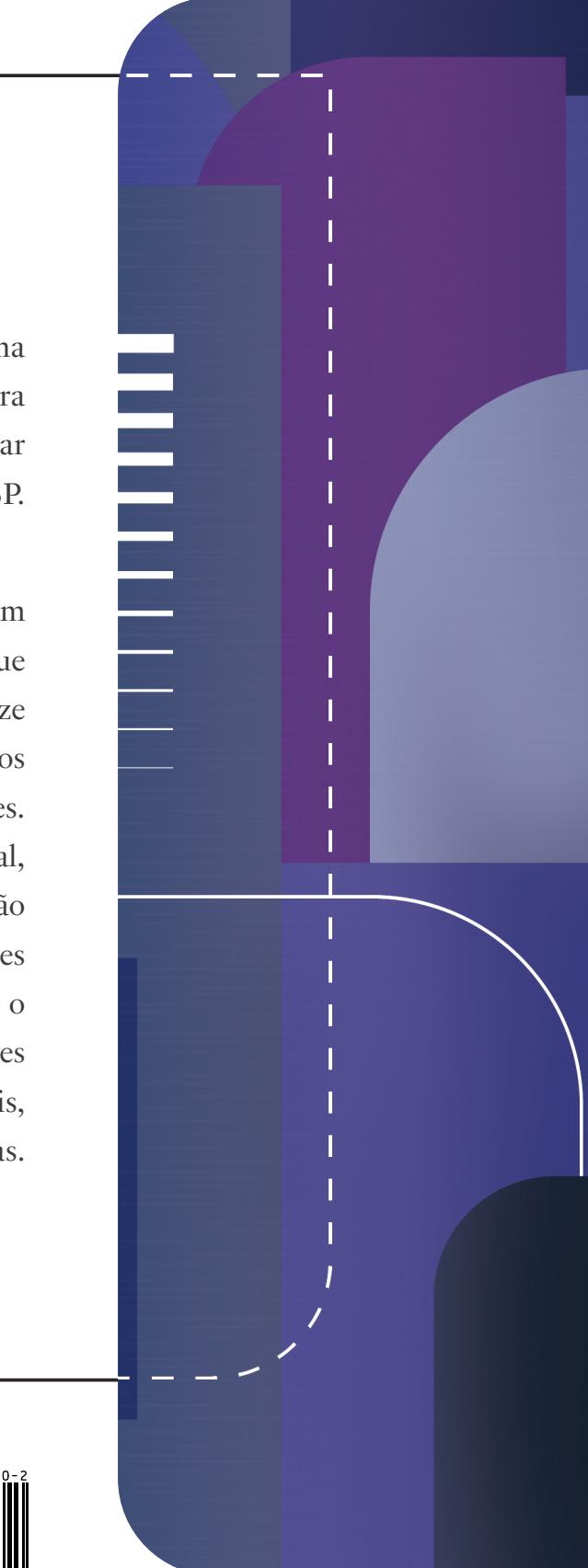
# SOBRE O ELABORADOR

## **Daniel Filipe Vieira**

Doutorando e mestre em Engenharia Elétrica pela Unicamp, com pesquisas voltadas para Internet das Coisas (IoT), sensores, análise de dados e metodologias de ensino aplicadas à IoT, utilizando abordagens como PBL (Aprendizagem Baseada em Problemas) e STEM. É engenheiro eletricista formado pelo Centro Universitário UniMetrocamp Wyden.

Atualmente, atua como professor universitário na Faculdade de Tecnologia Senai Roberto Mange, lecionando disciplinas como Inteligência Artificial, Big Data, desenvolvimento de aplicativos móveis, cibersegurança e computação em nuvem.

Tem ampla experiência no desenvolvimento de hardware, firmware, software, protocolos de comunicação e análise de dados com uso de Inteligência Artificial. Também atua como consultor em projetos eletrônicos, sistemas de telemetria e na elaboração de trabalhos científicos.



Esta publicação integra uma série da SENAI-SP Editora especialmente criada para apoiar os cursos do SENAI-SP.

O mercado de trabalho em permanente mudança exige que o profissional se atualize continuamente ou, em muitos casos, busque qualificações.

É para esse profissional, sintonizado com a evolução tecnológica e com as inovações nos processos produtivos, que o SENAI-SP oferece muitas opções em cursos, em diferentes níveis, nas diversas áreas tecnológicas.

