



Flutter

Neste módulo, vamos aprender como configurar o ambiente de desenvolvimento necessário para criar aplicativos Flutter. Isso inclui a instalação do SDK do Flutter, do Dart, do Android Studio e da configuração de um emulador ou dispositivo físico para testar aplicativos. Garantir que o ambiente de desenvolvimento esteja corretamente configurado é o primeiro passo para desenvolver de forma eficiente com Flutter.

Configurando o Ambiente de Desenvolvimento

Passo 1: Instalar o SDK do Flutter

1. Download do SDK do Flutter

- Acesse flutter.dev e faça o download do SDK do Flutter para o seu sistema operacional.

2. Descompactar o SDK

- Descompacte o arquivo baixado em um diretório de sua escolha.

3. Adicionar Flutter ao PATH

- Adicione o caminho do SDK do Flutter ao PATH do seu sistema:

No Windows:

```
setx PATH "%PATH%;C:\path\to\flutter\bin"
```

No macOS e Linux:

```
export PATH="$PATH:/path/to/flutter/bin"
```

4. Verificar a Instalação

- Execute o comando abaixo para verificar se o Flutter foi instalado corretamente:

```
flutter doctor
```

Passo 2 : Instalar o Dart

1. Dart Incluído no Flutter

- O Dart é incluído automaticamente ao instalar o Flutter, não sendo necessário uma instalação separada.

Passo 3: Instalar o Android Studio

1. Download e Instalação

- Baixe e instale o Android Studio a partir de developer.android.com/studio.

2. Configurar Android Studio

- Abra o Android Studio e siga as instruções de configuração inicial.
- Instale o SDK do Android durante o processo de configuração.

3. Instalar Plugins do Flutter e Dart

- Vá em File > Settings > Plugins e procure por “Flutter” e “Dart”.
- Instale ambos os plugins.

Passo 4: Configurar um Emulador ou Dispositivo Físico

1. Configurar um Emulador

- Abra o Android Studio.
- Vá em AVD Manager e crie um novo emulador.

2. Usar um Dispositivo Físico

- Habilite a depuração USB no seu dispositivo Android.
- Conecte o dispositivo ao seu computador via USB.

3. Verificar Dispositivos Disponíveis

- Execute o comando abaixo para listar os dispositivos disponíveis:

flutter devices

4. Executar o Aplicativo de Exemplo

No diretório do projeto, execute o comando:

flutter run

Nesta aula, vamos explorar a estrutura de um projeto Flutter. Entender a organização dos arquivos e diretórios em um projeto Flutter é crucial para o desenvolvimento eficiente. Discutiremos a função de cada diretório e arquivo principal, e como eles se integram para formar um aplicativo Flutter funcional.

Estrutura de um Projeto Flutter

Estrutura Básica de Diretórios

1. Diretório lib

- Contém o código Dart do aplicativo.
- Arquivo principal: main.dart.

2. Diretório android

- Contém os arquivos específicos da plataforma Android.
- Inclui o AndroidManifest.xml e os arquivos de configuração do Gradle.

3. Diretório ios

- Contém os arquivos específicos da plataforma iOS.
- Inclui o Info.plist e os arquivos de configuração do Xcode.

4. Diretório test

- Contém os testes unitários do aplicativo.

5. Arquivos Principais

- pubspec.yaml: Arquivo de configuração do projeto, onde são especificadas as dependências e outras configurações.
- *README.md* : Arquivo de descrição do projeto.
- .gitignore: Arquivo que define quais arquivos e diretórios devem ser ignorados pelo controle de versão Git.

Explorando o pubspec.yaml

1. Configurações Básicas

name: my_flutter_app

description: A new Flutter project.

version: 1.0.0+1

2. Dependências

dependencies:

flutter:

 sdk: flutter

 provider: ^6.0.0 # Exemplo de dependência adicionada

3. Assets e Fonts

flutter:

 assets:

 - assets/images/

 fonts:

 - family: Roboto

 fonts:

 - asset: fonts/Roboto-Regular.ttf

Explorando o main.dart

1. Estrutura Básica do main.dart

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

```
void main() {  
  
  runApp(MyApp());  
  
}  
  
class MyApp extends StatelessWidget {  
  
  @override  
  
  Widget build(BuildContext context) {  
  
    return MaterialApp(  
  
      home: Scaffold(  
  
        appBar: AppBar(  
  
          title: Text('Meu Aplicativo Flutter'),  
  
        ),  
  
        body: Center(  
  
          child: Text('Olá, Flutter!'),  
  
        ),  
  
      ),  
  
    );  
  
  }  
  
}
```

Explicação: Este código inicializa o aplicativo Flutter e exibe uma tela com um título e um texto centralizado.

Nesta aula, vamos explorar os widgets e layouts básicos do Flutter. Widgets são os blocos de construção fundamentais de um aplicativo Flutter. Discutiremos como utilizar widgets básicos como Text, Image, Column e Row para construir layouts simples. Também veremos como organizar widgets dentro de containers para criar interfaces de usuário responsivas.

Widgets e Layouts Básicos

Widgets Básicos

1. Widget Text

```
Text(  
  'Olá, Flutter!',  
  
  style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),  
  
);
```

Explicação: Exibe um texto estilizado na tela.

2. Widget Image

```
Image.network('https://flutter.dev/images/flutter-logo-sharing.png');
```

Explicação: Exibe uma imagem a partir de uma URL.

Layouts Básicos

1. Widget Column

```
Column(  
  children: [
```

```
mainAxisAlignment: MainAxisAlignment.center,
```

```
children: <Widget>[
```

```
    Text('Texto 1'),
```

```
    Text('Texto 2'),
```

```
],
```

```
);
```

Explicação: Organiza widgets em uma coluna vertical.

2. Widget Row

```
Row(
```

```
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
```

```
    children: <Widget>[
```

```
        Text('Texto 1'),
```

```
        Text('Texto 2'),
```

```
    ],
```

```
);
```

Explicação: Organiza widgets em uma linha horizontal.

Containers

1. Widget Container


```
Container(  
  padding: EdgeInsets.all(16.0),  
  
  decoration: BoxDecoration(  
  
    color: Colors.blue,  
  
    borderRadius: BorderRadius.circular(8.0),  
  
  ),  
  
  child: Text('Conteúdo do Container', style: TextStyle(color: Colors.white)),  
  
);
```

Explicação: O Container é um widget versátil que pode ser utilizado para adicionar padding, margens, bordas e cor de fundo a outros widgets.

Nesta aula, aprenderemos sobre a interação básica do usuário em aplicativos Flutter. Discutiremos como utilizar widgets interativos como RaisedButton, TextField e GestureDetector. Veremos exemplos práticos de como capturar e responder a eventos de usuário, como cliques de botão e entrada de texto.

Interação Básica do Usuário

Widgets Interativos

1. Widget RaisedButton

```
RaisedButton(  
  onPressed: () {  
  
    print('Botão pressionado');
```

```
},  
  
child: Text('Pressione-me'),  
  
);
```

Explicação: Cria um botão que responde ao clique do usuário.

2. Widget TextField

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

Explicação: Cria um campo de texto que captura a entrada do usuário.

Detecção de Gestos

1. Widget GestureDetector

```
GestureDetector(  
  onTap: () {  
  
    print('Área tocada');
```

```
},  
  
child: Container(  
  
  color: Colors.blue,  
  
  padding: EdgeInsets.all(16.0),  
  
  child: Text('Toque aqui', style: TextStyle(color: Colors.white)),  
  
),  
  
);
```

Explicação: Detecta gestos de toque em uma área específica da interface do usuário.

Exemplo de Interação Completa

1. Código Completo

```
import 'package:flutter/material.dart';  
  
void main() {  
  
  runApp(MyApp());  
  
}  
  
class MyApp extends StatelessWidget {  
  
  @override  
  
  Widget build(BuildContext context) {  
  
    return MaterialApp(  

```

```
home: Scaffold(
```

```
  appBar: AppBar(
```

```
    title: Text('Interação Básica'),
```

```
  ),
```

```
  body: InteractionExample(),
```

```
),
```

```
);
```

```
}
```

```
}
```

```
class InteractionExample extends StatefulWidget {
```

```
  @override
```

```
  _InteractionExampleState createState() => _InteractionExampleState();
```

```
}
```

```
class _InteractionExampleState extends State<InteractionExample> {
```

```
  String _displayText = 'Pressione o botão';
```

```
  void _updateText() {
```

```
    setState(() {
```

```
      _displayText = 'Botão pressionado!';
```

```
    });
```

```
}
```

@override

Widget build(BuildContext context) {

return Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Text(_displayText),

SizedBox(height: 20),

RaisedButton(

onPressed: _updateText,

child: Text('Pressione-me'),

),

],

),

);

}

}

Explicação: Este exemplo cria um aplicativo com um texto que muda quando o botão é pressionado. Utiliza o `setState` para atualizar o estado do widget e redibujar a interface.

Este conteúdo fornece uma base sólida para iniciantes em Flutter aprenderem a configurar seu ambiente de desenvolvimento, entenderem a estrutura de um projeto Flutter, utilizarem widgets e layouts básicos e implementarem interações básicas do usuário. Para mais detalhes, consulte a documentação oficial do [Flutter: Flutter Documentation](#).

Materiais Extras

Você pode realizar o download do arquivo contendo os materiais extras utilizados ao longo das aulas por meio do seguinte link: <https://drive.google.com/file/d/1mg7lqMI8Pt2zl0rHlsFS0Qew00YN-sEX/view?usp=sharing>.

Conteúdo Bônus

Para aprofundar seus conhecimentos sobre o desenvolvimento com Flutter, sugiro o vídeo disponível no oficial do canal Flutter no YouTube, intitulado “How do I make my first Flutter app.” Esse material é especialmente indicado para iniciantes e apresenta, de forma prática e didática, como criar um aplicativo em Flutter desde o início. O vídeo, que conta com explicações detalhadas e exemplos passo a passo, aborda conceitos essenciais como widgets, o funcionamento do “hot reload” e uma demonstração completa da construção de um app. Esse conteúdo é gratuito e acessível, ideal para quem deseja iniciar no desenvolvimento com Flutter.

Referências Bibliográficas

BOYLESTAD, R. L.; NASHELSKY, L. **Dispositivos Eletrônicos e Teoria de Circuitos**. 11. ed. Pearson, 2013.

DEITEL, P. J.; DEITEL, H. M. **Ajax, Rich Internet Applications e Desenvolvimento Web para Programadores**. Pearson, 2008.

DUARTE, W. **Delphi para Android e iOS: Desenvolvendo Aplicativos Móveis**. Brasport, 2015.

FELIX, R.; SILVA, E. L. da. **Arquitetura para Computação Móvel**. 2. ed. Pearson, 2019.

LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações Móveis: Arquitetura, Projeto e Desenvolvimento**. Pearson, 2005.

MARINHO, A. L.; CRUZ, J. L. da. **Desenvolvimento de Aplicações para Internet**. 2. ed. Pearson, 2019.

MOLETTA, A. **Você na Tela: Criação Audiovisual para a Internet**. Summus, 2019.

SILVA, D. (Org.) **Desenvolvimento para dispositivos móveis**. Pearson, 2017.

Ir para exercício