



FLUTTER

TEMA: NAVEGAÇÃO ENTRE TELAS



INTRODUÇÃO

DEV FEST LINEUP

Crie um novo projeto **flutter**:

```
flutter create --org br.com.heiderlopes --description "O festival dos desenvolvedores" devfestlineup
```



FLUTTER NAVIGATOR



NAVIGATOR O QUE É?

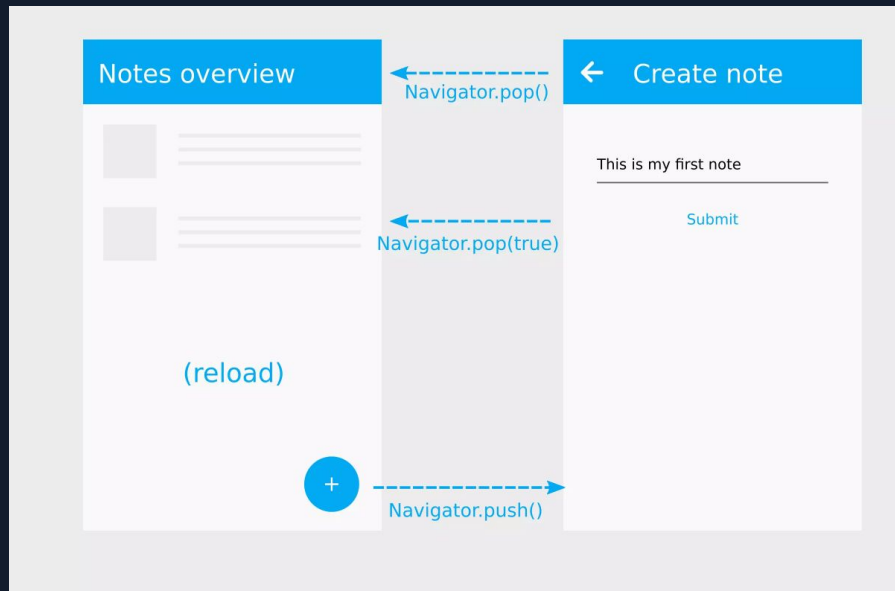
É o **gerenciador de navegação** de telas (rotas) no **Flutter**.

Ele controla a pilha de páginas

Quando você navega para uma nova tela, ela é **empilhada**.

Quando você volta, ela é **removida da pilha**.

***Analogia:** como empilhar e remover páginas de um livro.*



PRINCIPAIS MÉTODOS

Método	O que faz
push()	Adiciona uma nova tela na pilha.
pop()	Remove a tela atual e volta à anterior.
pushReplacement()	Substitui a tela atual por uma nova.
pushAndRemoveUntil()	Navega e remove todas as anteriores.

NAVIGATOR

NAVIGATOR.PUSH

Navega para **uma nova tela** (por exemplo: SegundaTela).

Adiciona essa nova tela por cima da atual.



```
Navigator.push(  
    context,  
    MaterialPageRoute(builder: (context) =>  
        SegundaTela()),  
);
```

NAVIGATOR

NAVIGATOR.POP

Volta para tela anterior.

Útil em botões como "Voltar".




```
Navigator.pop(context);
```


NAVIGATOR

NAVIGATOR.PUSHREPLACEMENT

Substitui à tela atual por uma nova.

A tela anterior é removida da pilha.




```
Navigator.pushReplacement(  
    context,  
    MaterialPageRoute(builder: (context) =>  
        TelaLogin()),  
);
```

NAVIGATOR

NAVIGATOR.PUSHANDREMOVEUNTIL

Navega para TelaInicial e limpa toda a pilha.

Ideal para após login ou logout.




```
Navigator.pushAndRemoveUntil(  
    context,  
    MaterialPageRoute(builder: (context) =>  
        TelaInicial()),  
    (route) => false,  
);
```

NAVIGATOR

NAVEGAR COM RETORNO DE DADOS

push(): pode esperar retorno com o `await`.

pop(): pode enviar dados de volta.



```
final resultado = await Navigator.push(
  context,
  MaterialPageRoute(builder: (context) =>
    TelaDetalhes()),
);

// TelaDetalhes pode retornar:
Navigator.pop(context, 'resultado');
```

NAVIGATOR

NAVIGATOR AVANÇADO

Usa uma estrutura declarativa com rotas nomeadas e controle de estado.

Útil para web, deep link e apps complexos.

Exige **Router**, **RouteInformationParser**, **RouterDelegate**.

Prefira **Navigator com rotas nomeadas** em apps grandes.

Use gestores de estados (Provider, Riverpod, etc) para facilitar a navegação global.

FLUTTER

SNIPPETS E PRODUTIVIDADE



INTRODUÇÃO

IDENTAR AO SALVAR

No VSCode abra as configurações

- **Atalho Ctrl + ,** (ou vá em File → Preferences → Settings)

Pesquise por: **Formate on save**

Marque a opção “**Editor: Format On Save**”

INTRODUÇÃO

FLUTTER SNIPPETS

Flutter Snippets são atalhos de código pré-definidos que ajudam você a escrever código Flutter mais rápido no seu editor, principalmente no **Visual Studio Code (VS Code)**.

Snippets são blocos de código que, ao digitar um atalho (como stf), são automaticamente expandidos para uma estrutura completa (como um **StatefulWidget**). Eles economizam tempo e padronizam o código.



Snippets

Após instalar a extensão `Flutter Snippets`, digite por exemplo:

stless → cria um widget `StatelessWidget`

stful → cria um widget `StatefulWidget`

init → cria o método `initState()`

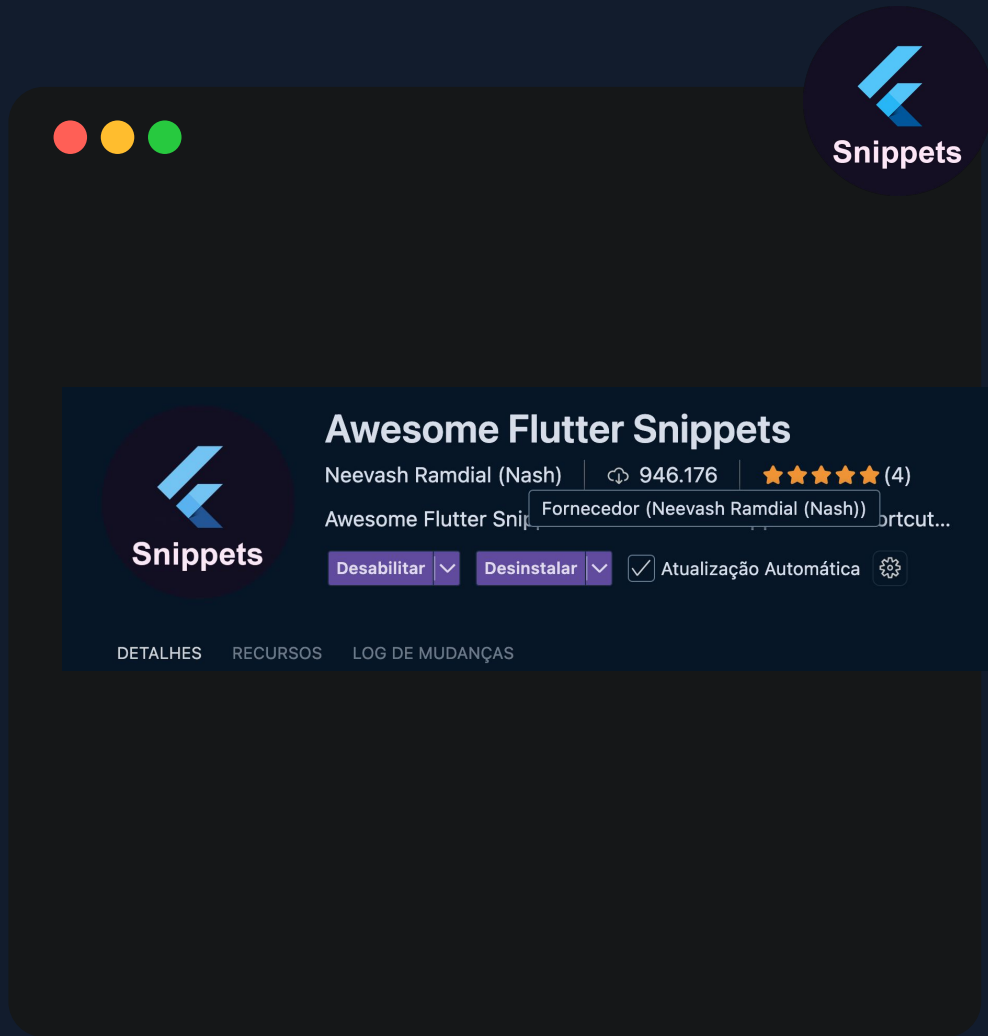
build → cria o método `build()`

cntr → cria um `Center(child: Text())`

INTRODUÇÃO

FLUTTER SNIPPETS

1. Acesse o **VS Code**
2. Vá para **Extensões**
(**Ctrl+Shift+X**)
3. Pesquise por: **Flutter Snippets**
4. Clique em **Instalar**



FLUTTER
MAIN



FUNÇÃO MAIN

A função **main** é o **ponto principal** de qualquer aplicativo **Flutter**.

void main: palavra chave obrigatória que define onde o programa vai começar.

runApp: Função do Flutter que inicializa o app e renderiza o widget passado como parâmetro. Geralmente este é o widget root.

MyApp: é o widget que você define. Ele retorna por exemplo um **MaterialApp**.



```
void main() {  
    runApp(const DevFestLineUp());  
}
```

FLUTTER

STATELESS WIDGET




DEVFEST LINEUP

STATELESS WIDGET

Abra o arquivo **main.dart**.

Digite **stless**.

Dê o nome de **DevFestLineUp** conforme código ao lado.



```
class DevFestLineUp extends StatelessWidget {  
  const DevFestLineUp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

STATELESS WIDGET

StatelessWidget é um tipo de widget que não muda com o tempo, ou seja, não possui estado interno mutável.

Ele é imutável: uma vez construído, nada dentro dele pode mudar a menos que um novo widget o substitua.



```
class DevFestLineUp extends StatelessWidget {  
  const DevFestLineUp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```


STATELESS WIDGET

QUANDO USAR?

O widget só mostra dados fixos (como texto, ícones, imagens).

Nenhuma interação precisa modificar o conteúdo (ex: clique, mudança de cor, contadores).

Os dados são passados como parâmetros e não mudam dentro do widget.



```
class DevFestLineUp extends StatelessWidget {  
  const DevFestLineUp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

DEVFEST LINEUP

STATELESS WIDGET

No arquivo **main.dart** e adicione o seguinte código ao lado o **MaterialApp**.

title: título da aplicação

debugShowCheckedModeBanner: false: remove o banner "debug" do canto superior direito.

theme: define o tema da aplicação. Aqui, ele usa um tema com cor primária índigo.

home: define a tela inicial da aplicação, que neste caso é `HomePage`.



```
class DevFestLineUp extends StatelessWidget {  
  const DevFestLineUp({super.key});  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'DevFest Lineup',  
      debugShowCheckedModeBanner: false,  
      theme: ThemeData(primarySwatch:  
Colors.indigo),  
      home: const HomePage(),  
    );  
  }  
}
```

FLUTTER

STATEFUL WIDGET



STATEFUL WIDGET

Ainda no **main.dart** e adicione o código ao lado. Use o **stful**

A **HomePage** é um widget com estado mutável. Ela estende **StatefulWidget**, o que significa que sua interface pode mudar com o tempo (por exemplo, com interações do usuário).

O método **createState()** cria a instância do estado associada a ela: **_HomePageState**.

State<HomePage> representa o estado interno da HomePage enquanto método **build()** é chamado sempre que o widget precisa ser redesenhado.

Scaffold é o esqueleto padrão de telas do Material Design. Ele fornece estrutura básica como: **appBar** (a barra superior da tela) e o **body**: o conteúdo principal da tela.



```
class HomePage extends StatefulWidget {  
  const HomePage({super.key});
```

```
  @override  
  State<HomePage> createState() =>  
    _HomePageState();  
}
```

```
class _HomePageState extends State<HomePage> {  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

DEVFEST LINEUP

HOME PAGE

Altere o **build** para construir um **Scaffold** no lugar do **Placeholder**.

Adicione a **appBar** e um **body**



```
class _HomePageState extends State<HomePage> {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(title: Text("DevFest  
LineUp")),  
      body: Placeholder(),  
    );  
  }  
}
```

FLUTTER

CLASSES E ENUMS

CLASSE E ENUM

Agora crie a classe **Talk** que irá representar as atrações:

O **construtor** é um construtor posicional, que recebe todos os parâmetros em ordem e inicializa os campos da classe.

Como os campos são **final**, eles devem ser inicializados na construção do objeto e **não podem ser modificados depois**.



```
enum TalkCategory { android, ios, flutter }
```

```
class Talk {  
    final String title;  
    final String speaker;  
    final String time;  
    final String description;  
    final TalkCategory category;  
  
    Talk(this.title, this.speaker, this.time,  
        this.description, this.category,  
        );  
}
```

FLUTTER

CRIANDO UMA LISTA



LISTA COM AS TALKS

Dentro da classe `_HomePageState` antes do **build** crie a lista de Palestras do evento.

Ao lado segue uma lista com as palestras que podem ser utilizadas para um festival de tecnologia.



```
final List<Talk> talks = [
  Talk(
    'Jetpack Compose e o Futuro das UIs Android', 'Heider
Lopes', '10:00', 'Como o Jetpack Compose está mudando a forma
como desenvolvemos interfaces para Android.',
    TalkCategory.android),
  Talk(
    'SwiftUI: Interfaces com Menos Código', 'Eric Brito',
    '11:00', 'Como o SwiftUI tem revolucionado o desenvolvimento de
interfaces no ecossistema Apple.', TalkCategory.ios),
  Talk(
    'Flutter Clean Architecture na Prática', 'Ricardo Ogliari',
    '12:00', 'Uma introdução prática à arquitetura limpa com
Flutter e Dart.', TalkCategory.flutter),
  Talk(
    'Integração Flutter com Plataformas Nativas', 'Heider
Lopes', '13:00', 'Utilizando Platform Channels para acessar
recursos Android e iOS no Flutter.', TalkCategory.flutter),
];
```

DEVFEST LINEUP

LISTA COM AS TALKS


Troque o **Placeholder** definido no body pelo **ListView** ao lado ⇒



```
body: ListView.builder(  
  itemCount: talks.length,  
  itemBuilder: (context, index) {  
    final talk = talks[index];  
    return ListTile(  
      title: Text(talk.title),  
      subtitle: Text('${talk.speaker} • ${talk.time}'),  
      leading: CircleAvatar(  
        backgroundColor: Colors.indigo.shade100,  
        child: Icon(  
          _getCategoryIcon(talk.category),  
          color: Colors.indigo,  
        ),  
      ),  
      trailing: IconButton(onPressed: () {}, icon:  
        Icon(Icons.favorite)),  
    );  
  },  
),
```

ÍCONE DA CATEGORIA

Após o `}` do `_HomePageState`
adicione o método ao lado ⇒



```
IconData _getCategoryIcon(TalkCategory
category) {
    switch (category) {
        case TalkCategory.android:
            return Icons.android;
        case TalkCategory.ios:
            return Icons.apple;
        case TalkCategory.flutter:
            return Icons.flutter_dash;
    }
}
```



FLUTTER

LISTA DE FAVORITOS



LISTA DE FAVORITOS


Para criar a lista de favoritos, dentro da classe `_HomePageState`, instancie um objeto do tipo `List<Talk>`



```
class _HomePageState extends State<HomePage> {  
    final List<Talk> _talksFavoritas = [];
```

LISTA DE FAVORITOS

Dentro do **itemBuilder** e antes do return **ListTile** crie a variável que irá controlar se a talk é favorita ou não.




```
itemBuilder: (context, index) {  
    final talk = talks[index];  
    final isFavorito =  
_talksFavoritas.contains(talk);  
    return ListTile(  
        title: Text(talk.title),
```

DEVFEST LINEUP

CLIQUE FAVORITOS


Altere o **onPressed** do ícone referente ao favorito:



```
trailing: IconButton(  
  onPressed: () {  
    setState(() {  
      if (isFavorito) {  
        _talksFavoritas.remove(talk  
      );  
      } else {  
        _talksFavoritas.add(talk);  
      }  
    });  
  },
```

ÍCONE DE FAVORITOS

Altere o **icon** para utilizar o ícone de acordo com a marcação de favorito ou não.



```
icon: isFavorito
? const Icon(Icons.favorite, color: Colors.red)
: const Icon(Icons.favorite_border)),
```

FLUTTER

CRIANDO A TELA DE DETALHE



DEVFEST LINEUP

CLIQUE PARA DETALHES

Dentro de **main.dart** crie a classe **TalkPage**.

Nela iremos criar um construtor para receber a talk enviada da tela anterior.

← Detalhe da Talk



```
class TalkPage extends StatelessWidget {  
  final Talk talk;  
  
  const TalkPage({super.key, required this.talk});  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(title: Text("Detalhe da Talk")),  
      body: Padding(  
        padding: EdgeInsets.all(16),  
        child: Column(  
          crossAxisAlignment: CrossAxisAlignment.stretch,  
          children: [],  
        ),  
      ),  
    );  
  }  
}
```

DEVFEST LINEUP

CLIQUE PARA DETALHES

Dentro da **Column** de **TalkPage**
adicione o título da talk:

← Detalhe da Talk

Jetpack Compose e o Futuro
das UIs Android

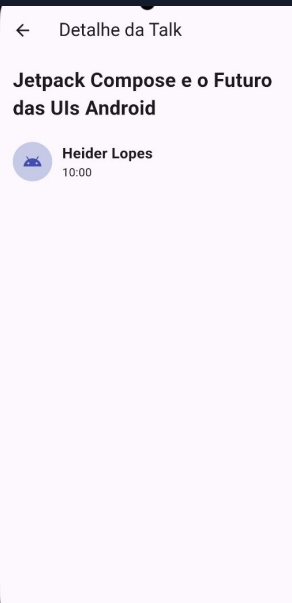


```
child: Column(  
    crossAxisAlignment: CrossAxisAlignment.stretch,  
    children: [  
        Text(  
            talk.title,  
            style: const TextStyle(fontSize:  
24, fontWeight: FontWeight.bold),  
        ),  
        const SizedBox(height: 24),
```


DEVFEST LINEUP

CLIQUE PARA DETALHES

Após o título, adicione a linha com ícone, nome do palestrante e horário:



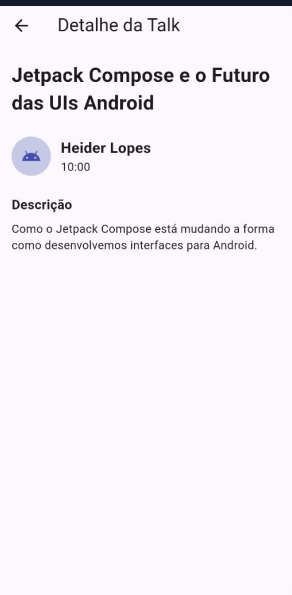
```
Row(
```

```
    children: [
        CircleAvatar(
            radius: 24,
            backgroundColor: Colors.indigo.shade100,
            child: Icon(
                _getCategoryIcon(talk.category), color: Colors.indigo,
            ),
        ),
        const SizedBox(width: 12),
        Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
                Text(
                    talk.speaker,
                    style: const TextStyle(
                        fontSize: 18, fontWeight: FontWeight.bold,
                    ),
                ),
                Text(talk.time),
            ],
        ),
    ],
),
```

DEVFEST LINEUP

CLIQUE PARA DETALHES

Após o a linha adicione a descrição:



```
const SizedBox(height: 24),  
    const Text(  
        "Descrição",  
        style: TextStyle(fontSize: 16,  
fontWeight: FontWeight.bold),  
    ),  
    const SizedBox(height: 8),  
    Text(talk.description, style: const  
TextStyle(fontSize: 14)),
```

DEVFEST LINEUP

CLIQUE PARA DETALHES

Após a descrição adicione o botão,
observe que o clique do botão
onPressed chama o:

Navigator.pop(context),
isso faz a ação para voltar
para a tela anterior.



```
const Spacer(),

    SizedBox(
      width: double.infinity,
      child: ElevatedButton.icon(
        onPressed: () => Navigator.pop(context),
        icon: const Icon(Icons.arrow_back),
        label: const Text("Voltar"),
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.indigo,
          foregroundColor: Colors.white,
        ),
      ),
    ),
```

DEVFEST LINEUP

CLIQUE PARA DETALHES

No **ListTile** da **HomePage** adicione o **onTap**



```
return ListTile(  
    onTap: () {  
        Navigator.push(  
            context,  
            MaterialPageRoute(builder:  
(context) => TalkPage(talk: talk)),  
        );  
    },  
);
```

FLUTTER

LOTTIE



ADICIONANDO A DEPENDÊNCIA

Abra o arquivo **pubspec.yaml** e adicione a dependência conforme imagem ao lado →

Após isso, no terminal digite

flutter pub get



```
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
```

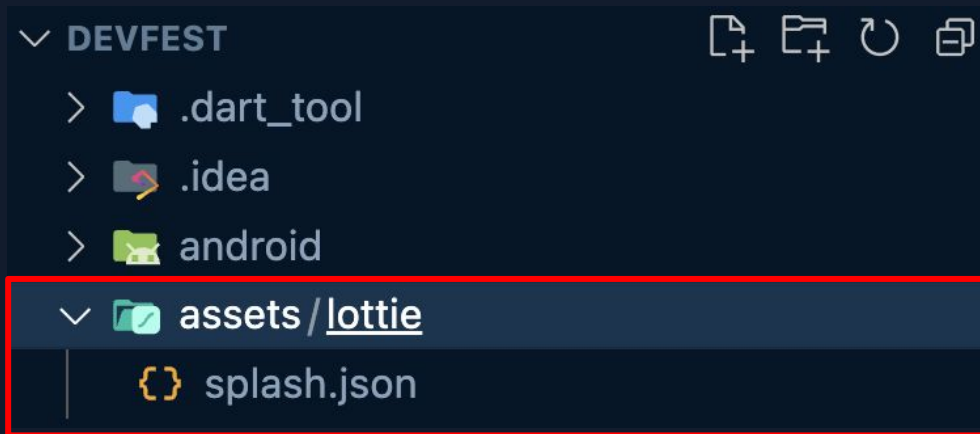
```
lottie: 3.3.1
```

ADICIONAR O JSON DA ANIMAÇÃO LOTTIE

Acesse o site <https://lottiefles.com/> e faça o **download** da animação que deseja colocar na splash.

Dê o nome de **splash.json** para o arquivo baixado.

Coloque o arquivo **splash.json** dentro da pasta do projeto, por exemplo:



DEVFEST LINEUP

ADICIONAR O JSON DA ANIMAÇÃO LOTTIE

Registre a animação no **pubspec.yaml**:

To add assets to your application, add an assets section, like this:

assets:

- **assets/lottie/splash.json**

- images/a_dot_ham.jpeg

DEVFEST LINEUP

EXEMPLO DE USO DO WIDGET DA LOTTIE

Exemplo de como adicionar no seu app:

```
Center(  
  child: Lottie.asset(  
    'assets/lottie/splash_animation.json',  
    width: 200,  
    height: 200,  
    fit: BoxFit.contain,  
  ),  
)
```

FLUTTER

EXERCÍCIO: CRIE A TELA SPLASH



DEVFEST LINEUP

CRIANDO A SPLASH

Crie a classe **SplashPage** como widget **StatefulWidget**

```
class SplashPage extends StatefulWidget {  
  const SplashPage({super.key});  
  
  @override  
  State<SplashPage> createState() => _SplashPageState();  
}
```

CRIANDO A SPLASH

Crie a classe **SplashPageState** que irá redirecionar para a tela da lineup após alguns segundos (continua no próximo slide)

```
class _SplashPageState extends State<SplashPage> {  
  @override  
  void initState() {  
    super.initState();  
    Future.delayed(Duration(seconds: 3), () {  
      if (!mounted) return; // verifica se o widget ainda está montado  
      Navigator.of(  
        context,  
      ).pushReplacement(MaterialPageRoute(builder: (_) => HomePage()));  
    });  
  }  
}
```

DEVFEST LINEUP

ADICIONANDO A ANIMAÇÃO AO SPLASH

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.black,
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Center(
            child: Lottie.asset(
              'assets/lottie/splash.json',
              width: 200,
              height: 200,
              fit: BoxFit.contain,
            ),
          ),
        ],
      ),
    ),
  );
}
```

DEVFEST LINEUP

CRIANDO A SPLASH

Altere a home do **DevFestLineUp** para carregar a Splash no início do app

```
class DevFestLineUp extends StatelessWidget {  
  const DevFestLineUp ({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'DevFest Lineup',  
      debugShowCheckedModeBanner : false,  
      theme: ThemeData(primarySwatch: Colors.indigo),  
      home: SplashPage(),  
    );  
  }  
}
```

OBRIGADO



heider-lobes-a06b2869

Copyright © 2025 | Professor (a) Heider Lopes
Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.