06 **Consumindo o Cubit na Home**

Transcrição

Finalizamos toda a parte lógica do nosso Cubit. O próximo passo será **mostrar** essas informações na nossa tela. Então, encerramos todo o nosso assunto com o pacote de BLoC, e agora começaremos a usar o pacote **Flutter BLoC**, que contém as informações para mostrar algo na tela por meio do `HomeCubit`.

Finalizamos o `home_cubit.dart`, então vamos para a nossa Home no arquivo `home.dart`.

Em algum momento, criaremos uma instância de `HomeCubit`. Então, dentro da classe `_HomeState`, na linha 14, criaremos a instância: `final HomeCubit homeCubit = HomeCubit()`. Em algum momento, usaremos essa instância para pegar as informações dos estados.

`home.dart`

```
class _HomeState extends State<Hom
  final HomeCubit homeCubit = Home
```

COPIAR CÓDIGO

Descendo rapidamente o nosso código, vamos tentar entender onde nossa lista está sendo construída. Na linha 38, temos um `SliverGrid.builder()` que contém um `itemBuilder` onde está o `movieCard()`. O `movieCard()` é o que aparece na nossa tela para cada cartão de filme.

Se prestarmos bem atenção, não queremos realmente desenhar toda a `Home`, mas redesenhar essa lista. Então, teremos que envolver esse `SliverGrid` com algum *widget* do próprio BLoC que dirá "apenas esse trecho de código precisa ser redesenhado, e não toda a aplicação". É para isso que temos o pacote de Flutter BLoC!

O `SliverGrid.builder()` ficará envolto de outro widget, que adicionamos clicando nele e selecionando "*Wrap with widget*", chamado `BlocBuilder()`. No autoimport, ele nos indica o pacote do qual importará esse widget: `from 'package:flutter_bloc...'`. Reforçando: terminamos a nossa parte lógica e agora estamos trabalhando com a parte visual.

```
const GenreFilter(),
BlocBuilder(
```

```
        child:SliverGrid.builder(

    // código omitido

          itemBuilder: (context, index
            return MovieCard(movie: Mo
          },
          itemCount: 5,
        ),
    ),

    //código omitido...
```

COPIAR CÓDIGO

Ao importar esse *widget*, surgem alguns erros. O `BlocBuilder` espera um *builder*, e nós estamos passando um `child` para ele. Então, nosso primeiro passo será chamar o `builder`, que espera uma função anônima que recebe um `context` e um `state`.

Quais são os estado que temos disponíveis, sobre os quais precisamos trabalhar na nossa `Home`? Ao todo, são quatro estados. Não precisamos, necessariamente, lidar com o primeiro porque é o estado inicial da Home; quando criamos a instância, o `HomeInitial` já é o estado inicial do nosso Cubit.

Dessa forma, precisaremos lidar com o `HomeLoading`, o `HomeSuccess` e o `HomeError`. Então, vamos chamar a função anônima que nos dará um bloco de código para trabalhar:

```
const GenreFilter(),
BlocBuilder(
    builder: (context, state) {

    },
    child: SliverGrid.builder(

//código omitido...
```

COPIAR CÓDIGO

O nosso primeiro estágio é o `HomeLoading`. Podemos criar uma condição para validar: se (`if`) o `state` atual for `HomeLoading`, ele nos retornará algo. Ele pode nos retornar um `Center()` que receberá como filho (`child`) um `CircularProgressIndicator()` para indicar o carregamento das informações:

```
const GenreFilter(),
BlocBuilder(
  builder: (context, state) {
    if (state == HomeLoading) {
      return Center(
        child: CircularProgressInd
```

```
        );
    // código omitido...
```

COPIAR CÓDIGO

Caso contrário (`else if`), vamos checar o próximo estágio. Se o `state` atual for `HomeSuccess` , ele nos retornará o nosso `SliverGrid.builder()` . Para isso, damos um `return` , removemos o `child` do `SliverGrid` e o movemos inteiro para dentro desse retorno. Depois o indentamos para organizar melhor:

```
  // ... código omitido

      } else if (state == HomeSucces
        return SliverGrid.builder(
          gridDelegate: const Sliver(
            crossAxisCount: 2,
            crossAxisSpacing: 16,
            mainAxisExtent: 240,
          ),
          itemBuilder: (context, ind(
            return MovieCard(
              movie: Movie(
                name: "James Bond",
                classification: Clas
                duration: "1h 22min"
                sinopse: "James Bond
                genre: "Suspense",
                imageURI: null,
```

```
                sessions: ["18:00"])
            },
            itemCount: 5,
        );
    }
```

COPIAR CÓDIGO

Retornaremos o `SliverGrid.builder` , ótimo! O que tem dentro dele? Lembrando que, se o estado for `HomeSuccess` , ele terá alguma informação dentro. Podemos acessar a informação através do `state` , chamando o objeto que existe dentro dele. Ou seja, se o estado for `HomeSuccess` , ele deve construir a lista de filmes; não deveria parar de funcionar, por exemplo.

Por último, inserimos outro `else` para verificar se o estado é igual a `HomeError` . Não precisamos do `if` porque, caso não seja nenhum dos outros estados, ele cairá automaticamente no `HomeError` . Retornaremos uma mensagem de erro com um `Center(child:)` contendo um `Text()` , cujo valor será `'Deu erro'` , por enquanto:

```
// ... código omitido

} else {
```

```
        return Center(child: Text('D
    }
```

COPIAR CÓDIGO

Agora, temos um retorno para todos os estados possíveis. Vamos salvar o código.

Ao salvar, recebemos um erro: `ProvideNotFoundException()`. Isso aconteceu porque o nosso `BlocBuilder()` não está especificado, ou seja, não dissemos qual `bloc` ele deve usar, o estado com que ele terá de lidar, não passamos quase nenhuma informação.

Por mais que tenha o `BlocBuilder()`, de onde estamos buscando esse `state`? E se tivermos mais de um `Cubit` dentro da `Home`? Precisamos dizer para o `BlocBuilder()` com o que ele está trabalhando.

Então, nós iremos tipar esse `BlocBuilder()` assim que o chamamos, na linha 39, com as duas informações com as quais irá trabalhar. A primeira informação é a `HomeCubit`, para dizer com que Cubit ele vai lidar, e `HomeStates`, para dizer com quais estados ele vai lidar: `BlocBuilder<HomeCubit, HomeStates>`.

Mas, só isso não resolve totalmente o nosso problema. É importante passar outra informação também, por meio da propriedade `bloc`. Ela recebe a referência de algum `bloc` ou `Cubit`. Nós já criamos uma instância dele lá no começo do nosso arquivo: `homeCubit`.

```
const GenreFilter(),
BlocBuilder<HomeCubit, HomeStates>
  bloc: homeCubit,
  builder: (context, state) {
    if (state == HomeLoading) {

// código omitido...
```

COPIAR CÓDIGO

Vamos salvar o arquivo novamente e recarregar a aplicação.

Agora, recebemos outro erro: `FluterError.fromParts<DiagnosticsNode>`. Isso aconteceu porque passamos um tipo errado de *render*.

Estamos trabalhando com um `CustomScrollView()`, que não recebe diretamente o *widget* `Center()` que colocamos no retorno de `HomeLoading`. Precisamos abraçá-lo com um widget chamado `SliverFillRemaining()`

Não precisamos fazer isso com o `Grid`, porque já é um `SliverGrid`. Por fim, faremos o mesmo com o `Center()` do retorno do último `else`:

```
// ... código omitido
    if (state == HomeLoading) {
      return const SliverFillRemai
        child: Center(child: Circu
      );
    } else if (state == HomeSucces
      return SliverGrid.builder(
// código omitido...
    } else {
      return SliverFillRemaining(c
// código omitido...
```

COPIAR CÓDIGO

Pronto! Salvamos novamente e recarregamos a aplicação.

Ao fazer isso, a tela da aplicação no emulador exibe a mensagem "Deu erro". É o que queríamos, de fato: **ver o estado acontecendo**.

O erro aconteceu porque ainda não chamamos a nossa função que recarrega e muda os estados. Então, o único estado que ele tem disponível é o `HomeInitial`. No entanto, o `HomeInitial` não consegue bater

em nenhum desses estados. Sendo assim, o retorno do último `else` é o que ele consegue exibir, porque é a exceção.

Nosso próximo passo, então, será chamar a função que muda os estados e, depois, lidar com as informações que estamos recebendo dentro da nossa API.

Nos encontramos no próximo vídeo!