Flutter Avançado - Aula 7

Exercícios 17 a 20

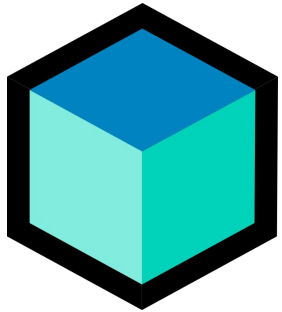
Fernando Duwe

v 1.0

Dicas de hoje

- Artigo: Dart no backend Ver mais

- BLoC: *Business Logic of Component* (Componente de regra de negócio
- Serve para separar a regra de negócio da *view* do usuário
- Baseada em *Streams*
- Aplicativo Reativo: É um aplicativo baseado em fluxos(streams), dados entram nesse fluxo, dados saem desse fluxo, esse fluxo pode ser observado de qualquer tela e parte do seu código.



1º vamos criar uma classe para o nosso *bloc*

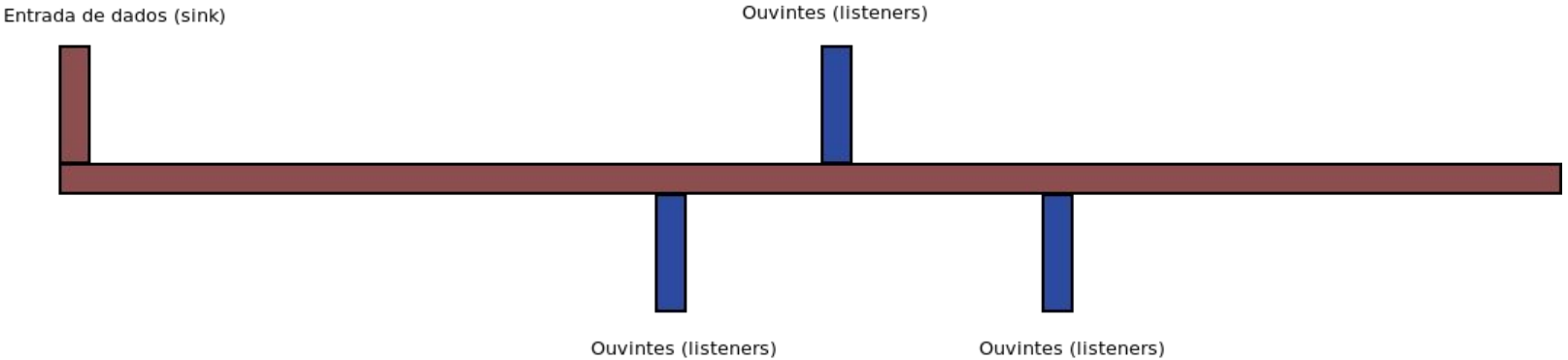
Criaremos um *stream*, fechando o stream quando destruirmos a classe

```
class ContactBloc extends BlocBase {
  List<ContactModel> _contactList = [];

  final StreamController<List<ContactModel>> _contactController =
StreamController<List<ContactModel>>();

  @override
  void dispose() {
    this._contactController.close();

    super.dispose();
  }
}
```

2º vamos criar um *getter* para o *stream*

Esta será uma porta de saída para o nosso *bloc*

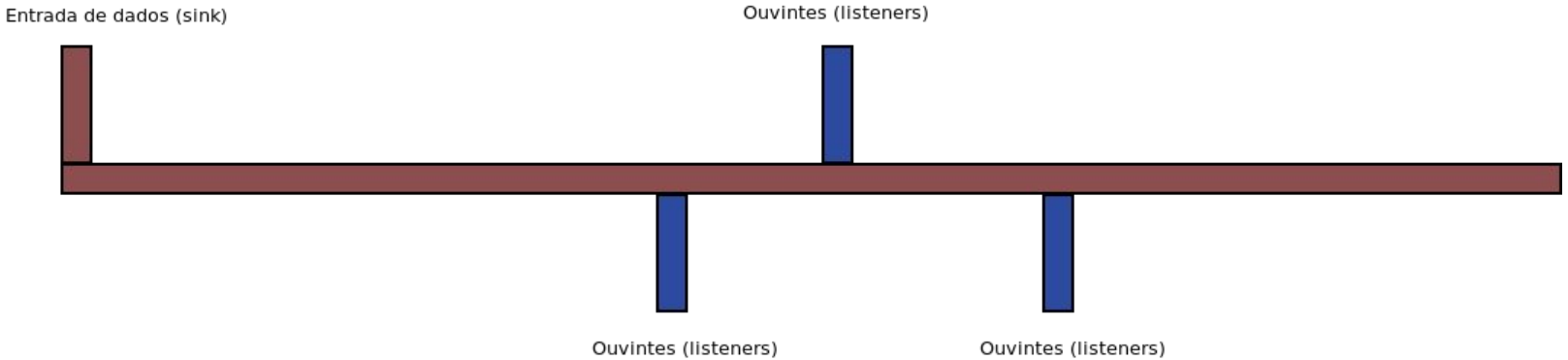
```
// Saída do stream
Stream<List<ContactModel>> get contacts => this._contactController.stream;
```



3º podemos criar também um getter para o *sink*

Possibilita adicionarmos informação dentro do nosso *stream*

```
// Entrada do stream
  Sink get inContacts => this._contactController.sink;
```



4º No nosso fonte principal, vamos criar um BlocProvider

O *BlocProvider* vai possibilitar o uso do nosso *bloc* por toda a aplicação

```
return BlocProvider(
        blocs: [Bloc((i) => ContactBloc())],
        child: MaterialApp(
          initialRoute: "/",
          routes: {
            "/": (context) => HomePage(),
          },
        ),
        dependencies: []);
```

5º Criando um StreamBuilder

Para que nossos *streams* fiquem disponíveis nos nossos *widgets*, podemos criar um StreamBuilder, que será atualizado sempre que houver uma mudança no nosso stream.

```
StreamBuilder<List<ContactModel>>(
     stream: BlocProvider.getBloc<ContactBloc>().contacts,
     builder: (context, snapshot) {
          if (snapshot.hasData) {

          }
     }
)
```

6º Usando o sink

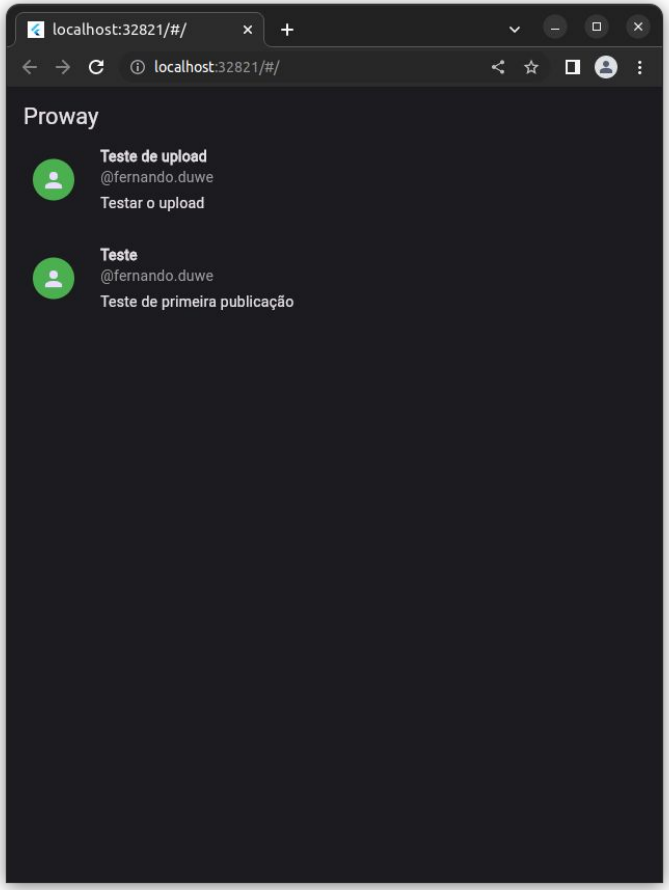
Podemos criar um novo fluxo, para inserir dados dentro do nosso bloc, para isso podemos usar o sink.

```
BlocProvider.getBloc<ContactBloc>().addContact.add(ContactModel(
                nome: this.nomeController.text,
                email: this.emailController.text
              ));
```

Exercício 17

Ver mais

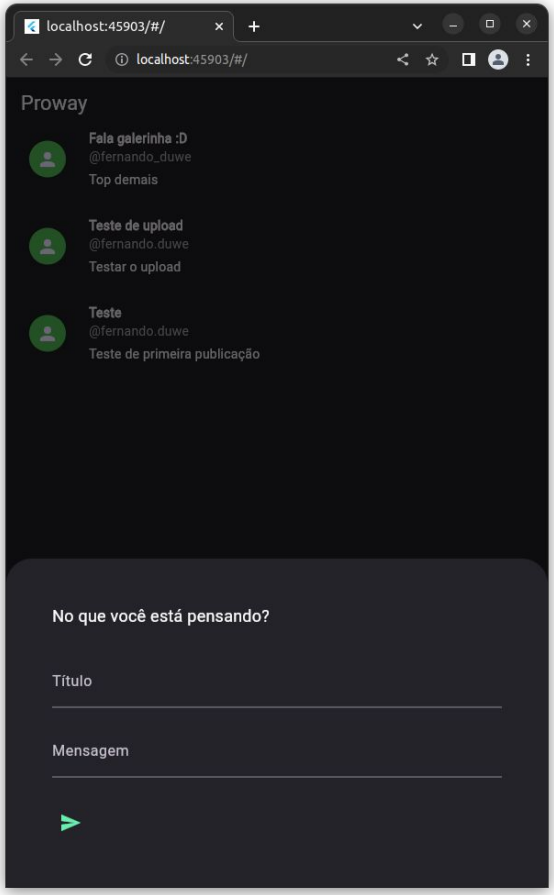
- Vamos criar um *twitter*
- Nosso *twitter* utilizará o *Firebase*
- Para listar os *posts*, vamos usar *bloc*
- Neste exercício, buscaremos os dados do Firebase
- Vamos exibir os posts em tela, usando Bloc



Exercício 18

Ver mais

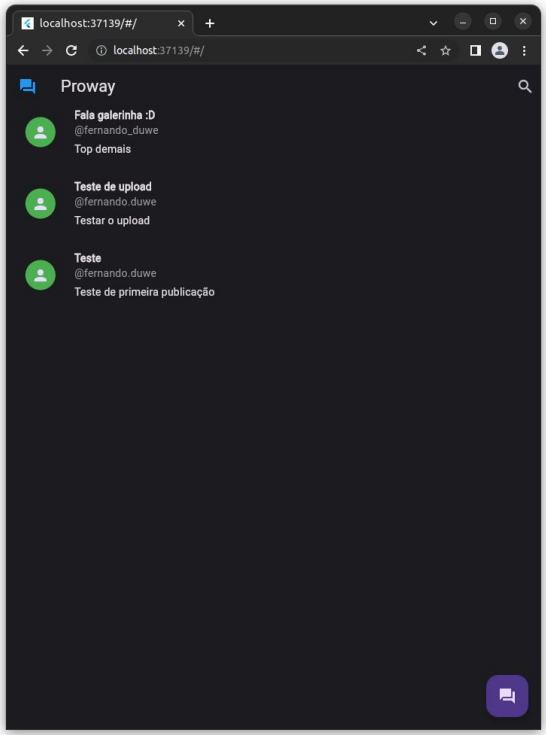
- Vamos agora implementar a criação de um post
- Usando blocs, crie um formulário onde o usuário poderá informar um título e uma mensagem
- Ao clicar em enviar, deve salvar o post no Firebase
- Após salvar o post, atualizar a lista de posts

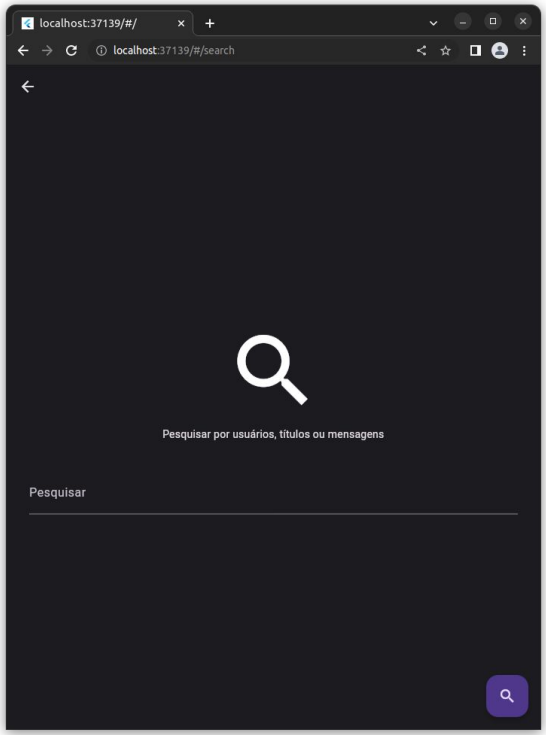


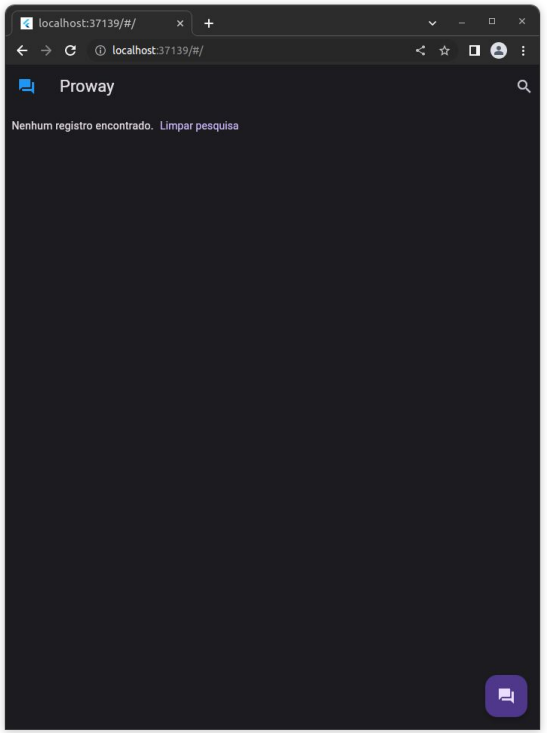
- Vamos agora criar uma pesquisa no nosso app
- Na AppBar vamos adicionar um botão de pesquisa
- Ao pesquisar, devem ser exibidos somente os posts referentes a pesquisa efetuada
- Caso nenhum registro corresponda a pesquisa efetuada, exiba uma mensagem tratando a situação

Exercício 19 - cont

Ver mais







- Vamos agora possibilitar a troca do nome do usuário
- Utilizando Redux
- Exiba o nome do usuário ao cadastrar um novo post
- No menu principal, adicione um botão para o usuário trocar seu nome
- Ao clicar nesse botão, apresente um formulário para que ele possa efetuar a troca

Exercício 20 - cont

Ver mais