1Exercício

Escreva uma nova aplicação Flutter com as seguintes características.

Dois campos textuais em que o usuário pode digitar nome e telefone de um contato seu.

Uma lista que mostra o nome e o telefone de cada contato.

Um botão que permite o cadastro de um contato.

Somente nomes válidos podem ser cadastrados. Um nome é válido somente se

- começa com maiúscula
- Tem, pelo menos, três letras
- Tem, pelo menos, um sobrenome

Somente números válidos podem ser cadastrados. Um número é válido somente se

- está no formato (xx) xxxxx-xxxx (o usuário tem que digitar todos os símbolos

Validações feitas com uma única instância Bloc global.

Refatoração utilizando Bloc com múltiplas instâncias de escopo restrito.

Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: novembro de 2023

Comece criando uma aplicação.

> **Nota**. O código completo se encontra em
>
> https://github.com/professorbossini/20232_pessoal_flutter_bloc_contatos_improve_it.git

```
flutter create gerenciador_de_contatos
```

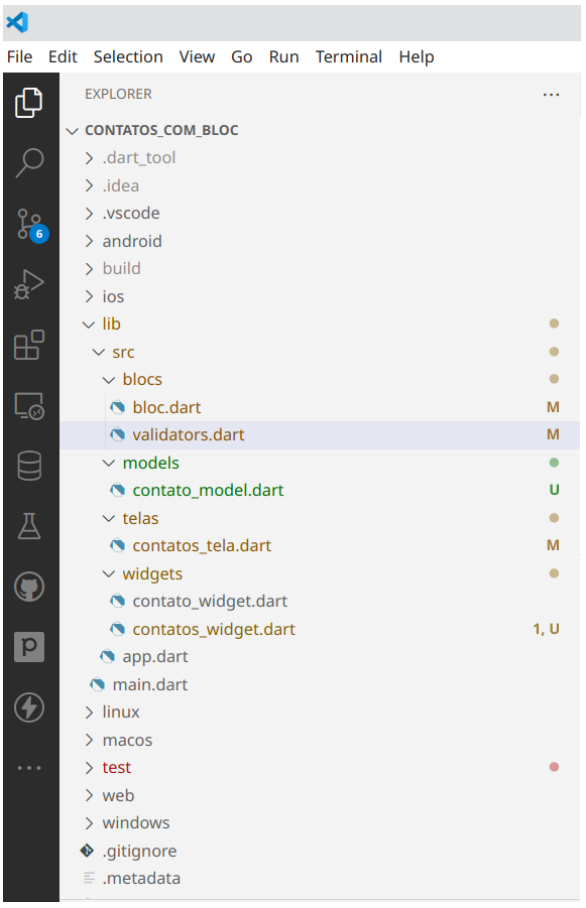
Vincule o VS Code à pasta criada.

```
code .
```

Clique Terminal >> New Terminal e abra um terminal interno do VS Code. Execute a aplicação.

```
flutter run
```

Crie a seguinte estrutura de arquivos e pastas.



- pasta blocs: abriga todo o conteúdo referente ao padrão Bloc

- arquivo bloc.dart: definição da classe Bloc e eventual instância Global
- arquivo validators.dart: transformers de validação
- pasta models: abriga as classes de modelo (tipo POJOs em Java)
- arquivo contato_model.dart: modelo que descreve o que é um contato (nome e número)
- pasta telas: abriga widgets que são telas
- contatos_tela.dart: tela principal da aplicação, vai exibir os campos para entrada de dados, botão e a lista de contatos
- pasta widgets: abriga widgets diversos que não são telas
- arquivo contato_widget.dart: widget para a exibição de um contato
- arquivo contatos_widget.dart: widget para a exibição de uma lista de contatos

Podemos começar criando a classe que descreve o que é um contato, no arquivo **contato_model.dart**.

```
class Contato{
 String _nome;
 String _numero;
 Contato(this._nome, this._numero);

 String get nome => _nome;
 String get numero => _numero;
 set nome(String nome) => _nome = nome;
 set numero(String numero) => _numero = numero;
}
```

A seguir, podemos definir o Bloc. Para tal, dado que ele será responsável por aplicar validações por meio de transformers, vamos começar definindo os validadores, no arquivo validators.dart. Observe como utilizamos expressões regulares para fazer as validações. Se quiser saber mais sobre elas, visite

https://api.flutter.dev/flutter/dart-core/RegExp-class.html

Estamos no arquivo **validators.dart**.

```
import 'dart:async';
mixin Validators{
 //começa com maiúscula, tem, pelos menos, duas letras minúsculas antes de um espaço em branco obrigatório e um sobrenome a seguir (com pelo menos uma letra)
 static final regExpNome = RegExp('[A-Z][a-z]{2,} [A-Za-z]+');
 //começa com parênteses, tem dois dígitos, fecha parênteses, tem um espaço em branco, tem 5 dígitos, tem um traço, tem 4 dígitos
 static final regExpNumero = RegExp('^\\([0-9]{2}\\) [0-9]{5}-[0-9]{4}\$');
 final validarNome = StreamTransformer<String, String>.fromHandlers(
   handleData: (nome, sink){
     //busca por todas as ocorrências da expressão regular no nome e verifica se tem só uma
     if (regExpNome.allMatches(nome).length == 1){
       sink.add(nome);
     }
     else {
       sink.addError('Nome deve começar com letra maiúscula, ter pelo menos 3 letras e um sobrenome');
     }
   }
 );
 final validarNumero = StreamTransformer<String, String>.fromHandlers(
   handleData: (numero, sink){
     //busca por todas as ocorrências da expressão regular no número e verifica se tem só uma
     if (regExpNumero.allMatches(numero).length == 1){
       sink.add(numero);
     }
     else {
       sink.addError('Número deve estar no formato (xx)xxxxx-xxxx');
     }
   }
 );
}
```

Depois disso, a definição do Bloc, no arquivo **bloc.dart**.

```dart
import 'validators.dart';
import 'dart:async';
import '../models/contato_model.dart';
class Bloc with Validators {
 //streams de validação
 final _nomeController = StreamController<String>();
 final _numeroController = StreamController<String>();
  Stream<String> get nome =>
_nomeController.stream.transform(validarNome);

 Stream<String> get numero =>
_numeroController.stream.transform(validarNumero);

 Function(String) get mudarNome => _nomeController.sink.add;

 Function(String) get mudarNumero => _numeroController.sink.add;

 //stream que lida com uma lista de contatos
 final _contatosController = StreamController<List<Contato>>();

 Stream<List<Contato>> get contatos => _contatosController.stream;
 //a lista de contatos
 List<Contato> _contatos = [];
 //adicionamos na lista e o sink a absorve a seguir
 void adicionarContato(Contato contato) {
   _contatos.add(contato);
   _contatosController.sink.add(_contatos);
 }

 void dispose(){
   _nomeController.close();
   _numeroController.close();
   _contatosController.close();
 }
}

//instância global do bloc
final bloc = Bloc();
```

Depois, implementamos a exibição de um Contato. Estamos no arquivo **contato_widget.dart**.

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

//mostrar um contato com nome e numero
//cada contato deve ter uma borda clara entre nome e numero
//cada par nome/numero deve ser englobado num card com sombra
class ContatoWidget extends StatelessWidget{
 final String nome;
 final String numero;
 ContatoWidget(this.nome, this.numero);
 @override
 Widget build(BuildContext context) {
   return Card(
     child: Container(
       padding: EdgeInsets.all(20.0),
       child: Column(
         children: [
           Text(
             nome,
             style: TextStyle(
               fontSize: 20.0,
               fontWeight: FontWeight.bold
             ),
           ),
           Container(
             margin: EdgeInsets.only(top: 10.0),
             child:Text(
             numero,
             style: TextStyle(
               fontSize: 20.0,
               fontWeight: FontWeight.bold
             ),
           )
           )
         ],
       ),
     ),
   );
 }
}
```

A seguir, escrevemos a classe de exibição de uma lista de contatos. Ela fica no arquivo **contatos_widget.dart**. Observe que os métodos que produzem os Widgets que constituem a tela tiveram a sua implementação omitida. Elas aparecem a seguir.

```
import 'package:flutter/material.dart';
import '../blocs/bloc.dart';
import '../models/contato_model.dart';
import '../widgets/contato_widget.dart';
class ContatosWidget extends StatelessWidget{
 String nomeAtual = '';
 String numeroAtual = '';
 List<Contato> contatos = [Contato('Ana', '1')];
 @override
 Widget build(BuildContext context) {
   return Container(
     margin: EdgeInsets.all(20.0),
     child: Column(
       children: <Widget>[
         nomeField(),
         numeroField(),
         Container(
           margin: EdgeInsets.only(top: 12.0),
           child: Row(
             children: <Widget>[
               Expanded(
                 child: submitButton(),
               ),
             ],
           ),
         ),
         contatosList()
       ],
     )
   );
 }
 Widget nomeField(){

   ...

 }
 Widget numeroField(){
    ...
 }

 Widget submitButton(){
   ...
 }

 Widget contatosList(){
   ...
 }
}
```

Veja a implementação de cada método.

```
...
Widget nomeField(){

    return StreamBuilder(
      stream: bloc.nome,
      builder: (context, snapshot){
        return TextField(
          onChanged: (valor){
            bloc.mudarNome(valor);
            nomeAtual = valor;
          },
          keyboardType: TextInputType.name,
          decoration: InputDecoration(
            labelText: 'Nome',
            hintText: 'Digite seu nome',
            errorText: snapshot.hasError ? snapshot.error.toString() : null,
          ),
        );
      },
    );

 }


...
```

```
...
Widget numeroField(){
    return StreamBuilder(
      stream: bloc.numero,
      builder: (context, snapshot){
        return TextField(
          onChanged: (valor){
            bloc.mudarNumero(valor);
            numeroAtual = valor;
          },
          keyboardType: TextInputType.number,
          decoration: InputDecoration(
            labelText: 'Número',
            hintText: 'Digite seu número',
            errorText: snapshot.hasError ? snapshot.error.toString() : null,
          ),
        );
      },
    );
 }


...
```

```
...
Widget submitButton(){
   return ElevatedButton(
     child: Text('Salvar'),
     onPressed: () => bloc.adicionarContato(Contato(nomeAtual, numeroAtual)),
   );
 }
...
```

```
...
Widget contatosList(){
   return Container(
     margin: EdgeInsets.only(top: 12.0),
     child: StreamBuilder(
       stream: bloc.contatos,
       builder: (context, snapshot){
         if (snapshot.hasData){
           return Column(
             children: snapshot.data!.map<Widget>(
               (contato) => ContatoWidget(contato.nome, contato.numero)).toList(),
           );
         }
         else {
           return Text('Nenhum contato adicionado');
         }
       },
     ),
   );
 }
...
```

No arquivo **contatos_tela.dart**, definimos a tela principal. Ela apenas exibe um ContatosWidget e controla as margens para que o conteúdo não fique colado nas bordas da tela do dispositivo.

```
import 'package:flutter/material.dart';
import '../widgets/contatos_widget.dart';
class ContatosTela extends StatelessWidget{
 @override
 Widget build(BuildContext context) {
   return Container(
     margin: EdgeInsets.all(20.0),
     child: Column(
       children: [
         ContatosWidget(),
       ],
     )
   );
 }
}
```

O Widget App, definido no arquivo **app.dart**, se encarrega da estrutura clássica, com um MaterialApp e um Scaffold.

```
import 'package:contatos_com_bloc/src/telas/contatos_tela.dart';
import 'package:flutter/material.dart';

class App extends StatelessWidget{
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Contatos',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: Scaffold(
       body: ContatosTela(),
     ),
   );
 }
}
```

Finalmente, no arquivo **main.dart**, o método main coloca a aplicação em execução, começando por uma instância de App.

```
import 'package:flutter/material.dart';
import 'src/app.dart';
void main() {
 runApp(App());
}
```

Referências

Dart programming language | Dart. Google, 2023. Disponível em <https://dart.dev/>. Acesso em Novembro de 2023.

Flutter - Build apps for any screen. Google, 2023. Disponível em <https://flutter.dev/>. Acesso em Novembro de 2023.