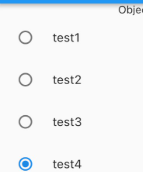
Checkbox nada mais nada menos que um quadrado onde o usuario vai selecionar.

Imagina um aplicativo de refeição

O usuário vai usar um RadioList para escolher uma opção

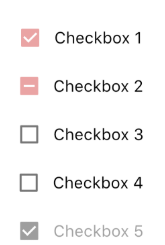
Lembrar que o radioList o usuário só deve escolher uma opção de refeição



* Frango
* Carne
* Churrasco

E ai vamos logo após exibir um checkBox onde o usuário vai escolher

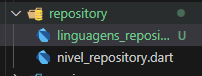
Já o checkbox o usuário vai escolher varias opções.



* Arroz
* Bata-fritas
* Feijão
* Macarrão

Mão na massa.

Primeiro vamos criar uma classe, essa classe vai simular como se essas listas de linguagens estivessem vindo de um banco de dados.



class LinguagensRepository {

  List<String> retornaLinguagens() {

    return ["Dart", "C#", "Java"];

  }

}

Criando o widget

            TextLabel(texto: "Linaguagens preferidas"),

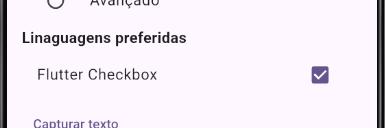
            CheckboxListTile(

              title: Text("Flutter Checkbox"),

              value: true,

              onChanged: (value) {},

            ),



Agora vamos fazer para pegar aquela lista de linguagens que “esta vindo do banco de dados”

  var linguagensRepository = LinguagensRepository();

  var linguagens = [];

  @override

  void initState() {

    niveis = nivelRepository.retornaNiveis();

    linguagens = linguagensRepository.retornaLinguagens();

    super.initState();

  }

# initState

O que é para que serve e como usar.

🚀 O método initState() é uma das peças-chave do ciclo de vida de um widget *stateful* no Flutter. Ele é chamado **uma única vez**, assim que o widget é inserido na árvore e antes de ser renderizado pela primeira vez.

🧠 O que ele faz?

* Serve para **inicializar variáveis**, carregar dados iniciais, configurar listeners, ou qualquer tarefa que só precisa acontecer uma vez.
* É o lugar certo para preparar o estado que seu widget vai usar.

🔍 No seu exemplo:

@override void initState() { niveis = nivelRepository.retornaNiveis(); linguagens = linguagensRepository.retornaLinguagens(); super.initState(); }

💡 Aqui você está:

* Buscando os dados de **níveis** e **linguagens** assim que o widget é criado.
* A chamada ao super.initState() garante que a lógica da superclasse também seja executada corretamente — **sempre mantenha ela no final ou início do método** (o mais comum é no final, como você fez).

🧵 Dica rápida

Se você quiser buscar dados assíncronos (como de uma API), o initState() precisa chamar uma função assíncrona separada, porque ele mesmo não pode ser async. Exemplo:

@override void initState() { super.initState(); \_carregaDados(); } void \_carregaDados() async { niveis = await nivelRepository.retornaNiveis(); linguagens = await linguagensRepository.retornaLinguagens(); setState(() {}); }

Se quiser, posso te ajudar a refatorar seu código para lidar com APIs ou comportamento assíncrono. Quer transformar isso num exemplo completo?

# Terminando o checkbox

Column(

              children: linguagens

                  .map(

                    (checkbox) => CheckboxListTile(

                      title: Text(checkbox.toString()),

                      value: false, // Aqui nao vamos deixar de forma selecionada.

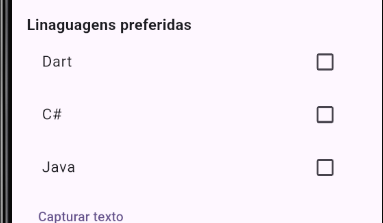
                      onChanged: (value) {},

                    ),

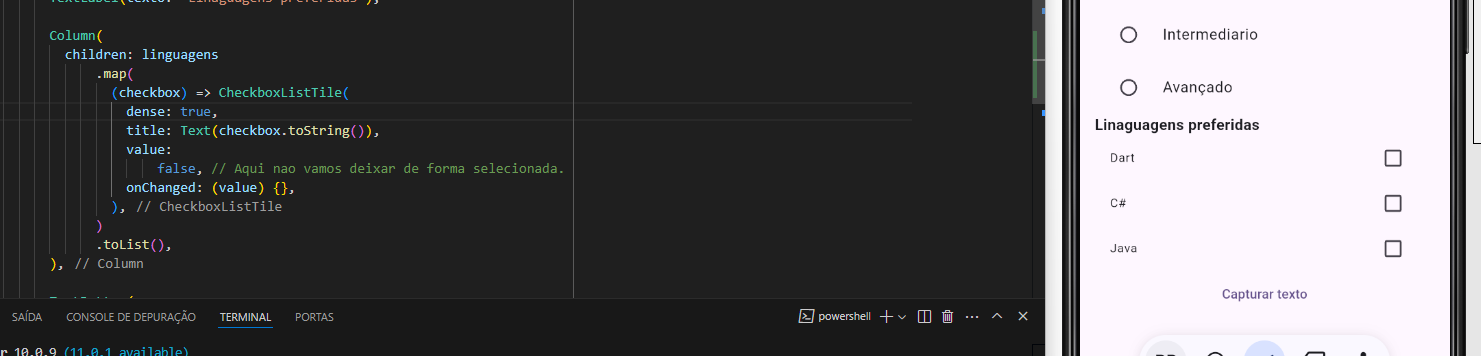
                  )

                  .toList(),

            ),



Usamos também o **DENSE**



Repara como tudo ficou menor.

# Agora vamos armazenar os que foram marcados.

Atenção diferente aqui é que vamos precisar de uma lista, porque vamos ter mais de uma marcação.

Primeiro vamos criar uma lista.

  var linguagensRepository = LinguagensRepository();

  var linguagens = [];

  var linguagensSelecionadas = [];

Criamos uma lista vazia.

Column(

              children: linguagens

                  .map(

                    (item) => CheckboxListTile(

                      dense: true,

                      title: Text(item.toString()),

                      value:

                          false, // Aqui nao vamos deixar de forma selecionada.

                      onChanged: (bool? value) {

                        if (value!) {

                          linguagensSelecionadas.add(item);

                          print(linguagensSelecionadas);

                        } else {

                          linguagensSelecionadas.remove(item);

                        }

                      },

                    ),

                  )

                  .toList(),

            ),

Como vai ser aqui, se o item for verdadeiro, vamos adicionar na lista, se for falso vamos remover da lista.

Agora faltou uma coisa, vamos precisar atualizar sempre quando clicado certo?

Dessa forma agora esta correto.

Column(

              children: linguagens

                  .map(

                    (item) => CheckboxListTile(

                      dense: true,

                      title: Text(item.toString()),

                      value: linguagensSelecionadas.contains(

                        item,

                      ), // Se o item tiver na lista, vamos deixar ele marcado, se nao remove..

                      onChanged: (bool? value) {

                        if (value!) {

                          setState(() {

                            linguagensSelecionadas.add(item);

                            print(linguagensSelecionadas);

                          });

                        } else {

                          setState(() {

                            linguagensSelecionadas.remove(item);

                          });

                        }

                      },

                    ),

                  )

                  .toList(),

            ),

