

26°  
CLASS

DELETE

# ROTEIRO



## PRINCIPAIS ELEMENTOS.

---

1

2

GET

---

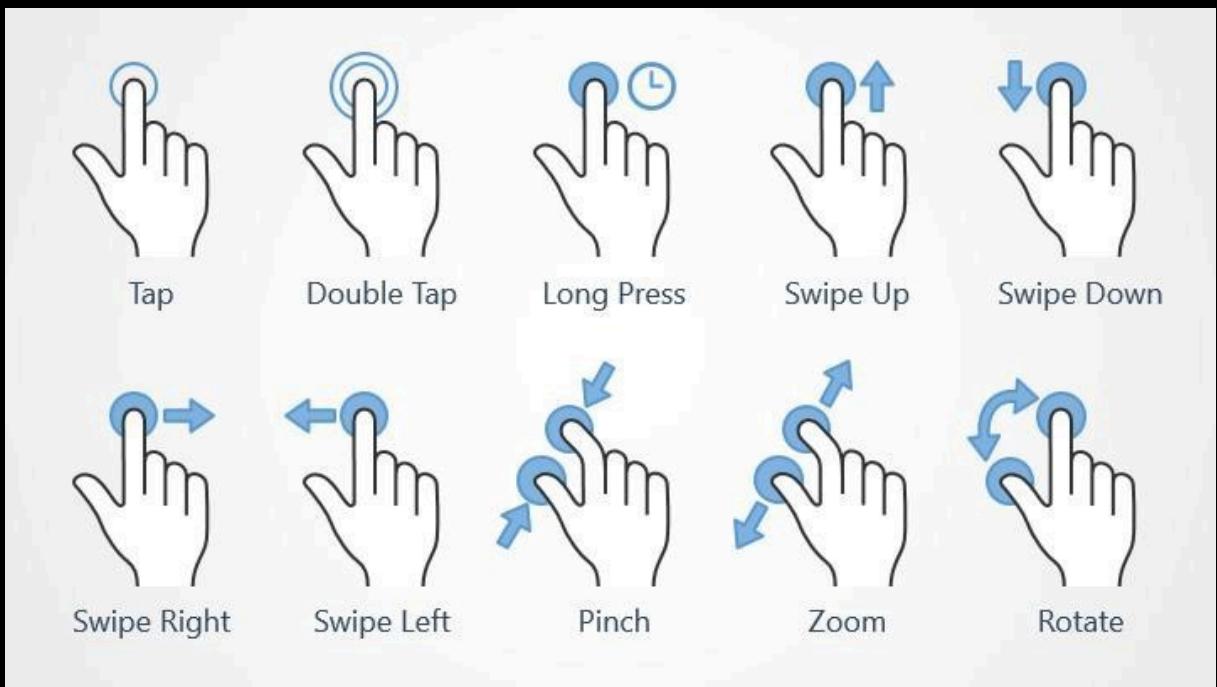
3

DELETE

---

# GESTURE DETECTOR.

NESTA AULA IREMOS APRENDER O  
COMPONENTE GESTURE DETECTOR.



ELE CONSEGUE ATRIBUIR FUNÇÃO  
PARA COMPONENTES QUE NÃO  
POSSUEM AÇÃO NATIVAMENTE.

POR EXEMPLO ICON, IMAGE OU  
TEXT.

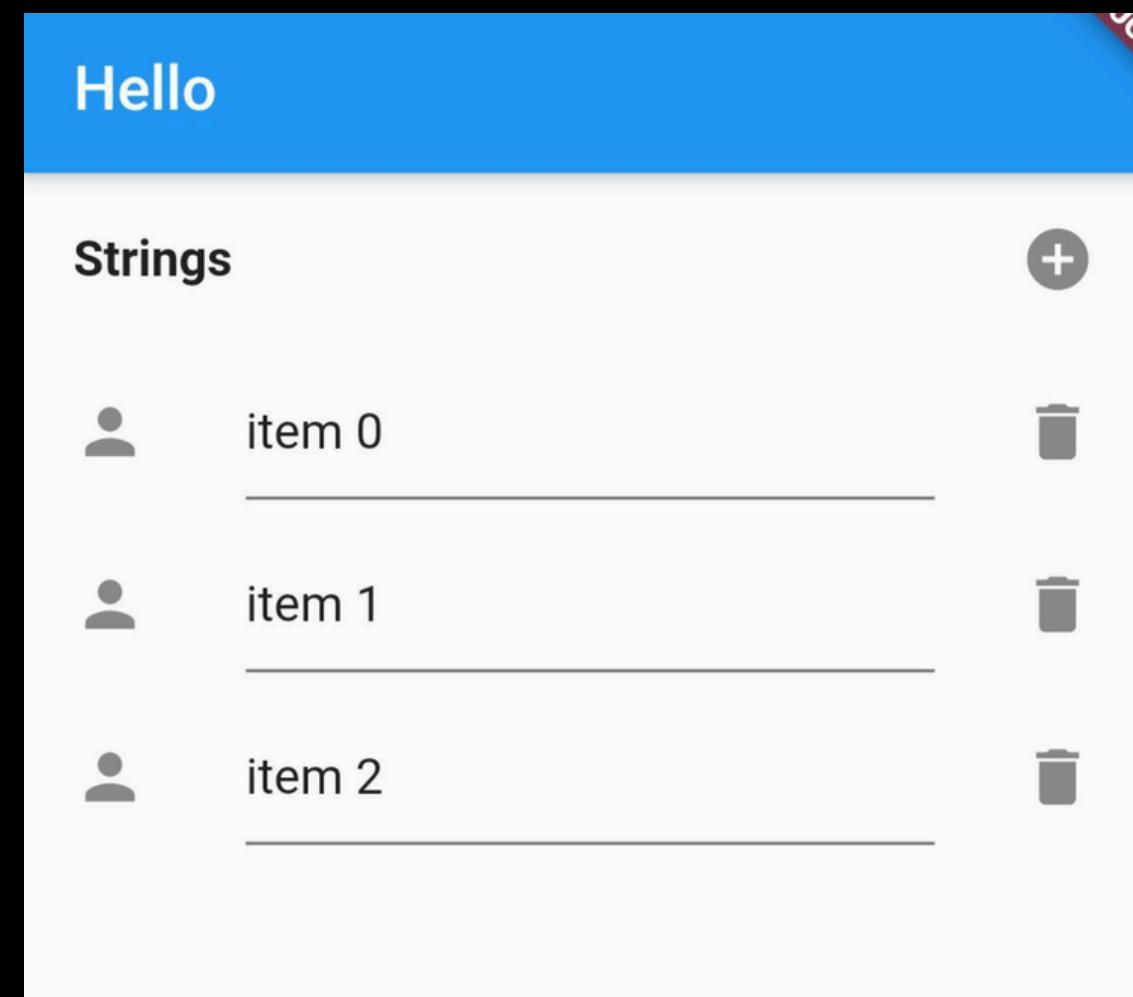


VAMOS  
FAZER  
?

EXEMPLO TELA DELETE

# FAZER A FUNÇÃO GET.

É NECESSÁRIO VER O ITEM PARA DEPOIS DELETAR.  
DESTA FORMA PRECISAMOS CRIAR A FUNÇÃO GET  
TAMBÉM.



# PASSO 1: CRIE SUA CLASSE STATEFUL E DEPOIS,

CRIE UMA LISTA PARA RECEBER OS VALORES DO BANCO;  
CRIE O INITSTATE QUE IRÁ RECEBER A FUNÇÃO GET.

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

  @override
  State<DeletePage> createState() => _DeletePageState();
}

class _DeletePageState extends State<DeletePage> {
  List<dynamic>? values;

  @override
  void initState() {
    super.initState();
    getValues();
  }
}
```

# PASSO 2:

CRIE A FUNÇÃO GET.

```
void getValues() {
    FirebaseFirestore.instance.collection("monitoramento").snapshots().listen(
        snapshots) {
            final data = snapshots.docs;
            setState(() {
                values = data;
            });
        },
    );
}
```

# PASSO 3:

## CRIE A FUNÇÃO DELETE

```
/// Função para deletar um documento pelo ID  
/// Precisa receber o parametro String id ,pois o id de cada item |  
Future<void> deleteValue(String id) async {  
    await FirebaseFirestore.instance  
        .collection("monitoramento")  
        .doc(id)  
        .delete();  
}
```

# PASSO 4:

## CRIE OS COMPONENTES DA TELA

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text("Tela Delete"),
      backgroundColor: Colors.blueAccent,
    ), // AppBar
    body: values == null
      ? const Center(child: CircularProgressIndicator())
      : ListView.builder(
          itemCount: values!.length,
          itemBuilder: (context, index) {
            final item = values![index];
            return ListTile(title: Text("Temperatura"),
              subtitle: Text("${item['temperatura']}"),
              trailing: GestureDetector(
                child: Icon(Icons.remove),
                onTap: () => deleteValue(item.id),
              ), // GestureDetector
            ); // ListTile
          },
        ) // ListView.builder
  );
}
```

# EXPLICANDO PASSO 4:

```
body: values == null  
    ? const Center(child: CircularProgressIndicator())  
    : ListView.builder(
```

NO CONTEÚDO(BODY), SE A LISTA VALUES ESTIVER NULA , NO CENTRO DA TELA FICARÁ UM LOADING.

PORÉM QUANDO OS DADOS DO BANCO CHEGAR, ELA IRÁ EXIBIR O LISTVIEW.BUILDER

# EXPLICANDO PASSO 4:

```
ListView.builder(  
    itemCount: values!.length,  
    itemBuilder: (context, index) {
```

**LISTVIEW.BUILDER** É UM CONSTRUTOR, ELE IRÁ ANALISAR A QUANTIDADE DE DADOS DA LISTA VALUES, E PARA CADA ITEM IRÁ CONSTRUIR UM COMPONENTE.

# EXPLICANDO PASSO 4:

```
itemBuilder: (context, index) {  
    final item = values![index];
```

PARA CADA ITEM CRIAMOS UMA VARIÁVEL ITEM , POIS FACILITA CHAMAR AS PROPRIEDADES.

# EXPLICANDO PASSO 4:

```
return ListTile(title: Text("Temperatura"),  
subtitle: Text("${item['temperatura']}"),  
trailing: GestureDetector(  
    child: Icon(Icons.remove),  
    onTap: () => deleteValue(item.id),  
, // GestureDetector  
); // ListTile
```

LIST TILE É UM CARD, QUE TEM 3 PROPRIEDADES:

- TITLE: TITULO,
- SUBTITLE: SUBTITULO,
- TRAILING: UM COMPONENTE MAIS AFASTADO A DIREITA.

# OBRIGADO!!

FLUTTER WITH VINI;

