É sempre importante entender os processos que seu aplicativo está fazendo para executar o resultado esperado, mas muitas vezes o que queremos é ter algo em mãos, um produto para testar logo e validar uma ideia. Nesse ponto de vista, seguir o BLoC Pattern de ponta a ponta pode parecer contraintuitivo, muito verboso e até mesmo desnecessário.

BLoC*é um padrão que utiliza Streams para adicionar reatividade aos componentes de uma aplicação.*

Pode ficar tranquilo, com abstrações simples, você vai perceber que vale a pena e vai te ajudar a enxergar de maneira mais clara o ciclo de vida da sua aplicação.

Uma pincelada antes da prática

Com as Streams do Dart (Linguagem de programação utilizada para o Flutter), conseguimos criar canais de comunicação através de eventos, e com esses canais, programar de forma reativa nossa aplicação.

Podemos imaginar a Stream como uma fila, onde podem ser inseridas informações de todos os tipos, inclusive outras Streams, veja uma representação:

O StreamController é a classe responsável por cuidar dessa estrutura, com ela é possível criar uma nova entrada ou consumir um dado presente, respectivamente, utilizando as propriedades *StreamController.sink* (para inserir dados) e *StreamController.stream* (para consumo).

Portanto, o padrão consiste em:

1. Mapear as funcionalidades e criar streams para cada contexto;
2. Criar builders ouvindo essas streams e encapsulando Widgets que serão atualizados dinamicamente;
3. Disparar eventos através da UI do seu APP para estas Streams (*sink.add()*);

4. Criar uma interceptação através de um *StreamTransformer*, para executar os eventos disparados e publicando novos dados, notificando os builders.

Podemos observar que criando canais para disparo de eventos, conseguimos interceptar para fazer requisições e alterar o estado dos componentes sem precisar renderizar toda a árvore de Widgets, gerando uma performance melhor.

Abstraindo para ganhar velocidade

É um padrão sensacional, mas que pode parecer confuso a primeira vista, não se preocupe, existem muitos conceitos de reatividade que podem não ser tão tranquilos assim, mas não abandone.

Você pode ir estudando cada detalhe desses processos mais a fundo enquanto já utiliza em seus projetos, para isso vamos utilizar uma biblioteca, a BLoC Library.

Essa biblioteca cria uma camada de abstração sobre as *streams* do Dart, facilitando a implementação do padrão e dando uma visibilidade absurda sobre o *lifecycle* das suas funcionalidades. Dentre os pacotes contidos na biblioteca, vamos utilizar o bloc e o flutter_bloc.

> Você precisa ter o Flutter instalado e configurado na sua máquina pra seguir os próximos passos. Aqui você descobre como.

Passo 1: Novo App

É só rodar o famoso *flutter create <app>*, adicionar os pacotes bloc e flutter_bloc no seu *pubspec.yaml*, e instalar os pacotes rodando o comando *flutter pub get*.

> Nosso *pubspec.yaml com os pacotes.*

Passo 2: Estruturando nosso BLoC

Vamos criar um BLoC simples para o nosso contador, e matar o setState contido nele por padrão.

Podemos adicionar um arquivo *counter-bloc.dart* dentro da nossa pasta *lib*, que vai conter:

1. Um enum para identificação do tipo de evento que estamos recebendo.
2. A classe principal da nossa estrutura, que ja utiliza o pacote que adicionamos.
3. O método *mapEventToState*, responsável por interceptar nossos eventos e retornar estados para nossa UI.

Nosso arquivo ficou assim:https://medium.com/media/3024eeacb4d7d213ff6b74107fe2064d

Antes de escrever nossa lógica, precisamos adicionar um *BlocProvider* à nossa aplicação, apontando o BLoC que queremos utilizar. Isso será feito no arquivo *main.dart.*

Assim, nossa classe MyApp (*lib/main.dart*), toma a seguinte forma:https://medium.com/media/0f237df754a39d137e1aaf6cf495cb40

Vamos voltar ao nosso BLoC, e adicionar uma condicional para cada evento que chegar ao nosso método *mapEventToState*, se o evento for de entrada, vamos somar 1 ao *state* atual, se for saída é só subtrair.

Por fim, nosso BLoC ficará dessa maneira:https://medium.com/media/bc031284e25cef7394e79bbd6c390198

No arquivo *main.dart*, só precisamos substituir o *counter* local pelo nosso BLoC, encapsulando o Widget que escutará o estado em um *BlocBuilder* (Que recebe nosso BLoC vindo do contexto), e alterando a função *onPressed* do floating button para adicionar um evento ao BLoC ao invés de chamar um *setState.*

Com essas alterações nosso arquivo *main.dart* fica assim:

E como podemos ver, nossa aplicação já está com seu ciclo de vida funcionando através do BLoC de maneira simples, prática e intuitiva.

Por fim

Como deixamos de utilizar o *setState* do Widget, o Flutter não irá renderizar novamente toda a árvore de Widgets cada vez que o estado mudar, pois só os Widgets que estão encapsulados no Builder escutam essas alterações. Isso nos deixa com uma performance melhor e com muito mais controle dos itens dinâmicos na nossa interface do usuário.

Gostou do artigo se sim compartilhe e ajude o Flutter a crescer.