04

Para saber mais: Gerenciamento de estado com Provider

O gerenciamento de estado da aplicação consiste em como gerenciar os valores que estão atribuídos às variáveis e constantes no momento em que o aplicativo está sendo utilizado. Existem diversas formas de gerenciar estados, podemos listar algumas delas:

- Provider (https://pub.dev/packages/provider);
- Redux (https://pub.dev/packages/flutter_redux);
- MobX (https://mobx.netlify.app/getting-started/);
- Bloc (https://pub.dev/packages/flutter_bloc).

Não existe um melhor ou pior, a escolha de qual utilizar depende bastante de cada caso. Optamos por iniciar este curso partindo do conteúdo do curso Flutter: Gerenciamento de Estados com Provider (https://www.alura.com.br/curso-online-flutter-gerenciamento-estados-provider) por considerarmos o Provider (https://pub.dev/packages/provider) uma forma mais simples de gerenciar o estado de uma aplicação.

Lembrando que "simples" não quer dizer ruim, de acordo com a própria documentação do Flutter, ele é a forma mais fácil e com o menor grau de burocracias, na visão deles, para gerenciarmos o estado de uma aplicação.

Não é um pré-requisito obrigatório saber utilizar o Provider para fazer este curso, mas é desejável que você tenha alguma noção de como ele funciona. Caso não tenha, sem problemas! Vou te explicar antes de entrarmos na prática:

> O provider parte da premissa do *Single Source of Truth*, ou seja, única fonte da verdade. Por baixo dos panos ele transforma *models* em uma espécie de *Singleton*.
>
> O Singleton é um design pattern que provê uma única instância da classe e um ponto de acesso global para essa mesma instância. Normalmente, em nossos projetos, teremos classes que realmente devem ter apenas uma única instância (conexão com o banco de dados ou gerenciamento de estado, que é o caso do projeto desse curso) e classes que serão instanciadas diversas vezes com valores diferentes (a maioria delas!). Portanto, ele é um padrão que deve ser utilizado com moderação e planejamento para que seja usado apenas onde realmente fizer sentido.
>
> O que o **Provider** faz é aplicar o **Singleton** para o gerenciamento de estado do nosso projeto, fazendo com que esse estado fique mais limpo e consistente.

Se você quiser se aprofundar ainda mais nesse design pattern, recomendo [esse texto sobre Singleton, com exemplos de código em diversas linguagens (https://refactoring.guru/pt-br/design-patterns/singleton)](https://refactoring.guru/pt-br/design-patterns/singleton).

Através de comandos como `Provider.of<Cliente>(context)`, podemos pegar valores ou alterar os valores que estão em model de Cliente (ou em qualquer model, citei apenas um exemplo). Outra forma de obter os dados de model que trabalha com o Provider é através do widget `Consumer`. Com ele, todos os widgets filhos conseguem acessar os dados de model na qual ele está conectado.

A vantagem de fazer desta forma é saber que os dados de cliente sempre estarão em model de cliente, os de transações em model de transações, e assim sucessivamente. Quaisquer dúvidas sobre esse tema, deixa lá no fórum que iremos te ajudar, ok? :)

Caso tenha entendido bem o conceito, que tal exercer o espírito de comunidade e acessar lá para ver se algum(a) outro(a) aluno(a) tem dificuldade? Lembre-se que, quando ensinamos, estamos praticando e aprendendo mais!