



Flutter Avançado - Aula 6

Exercícios 16 a ?



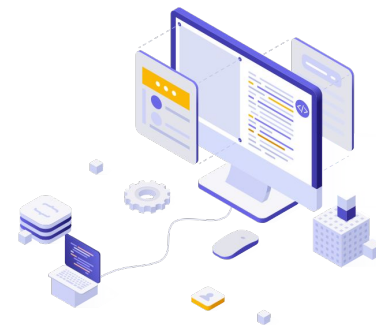
- Flutter Awesome

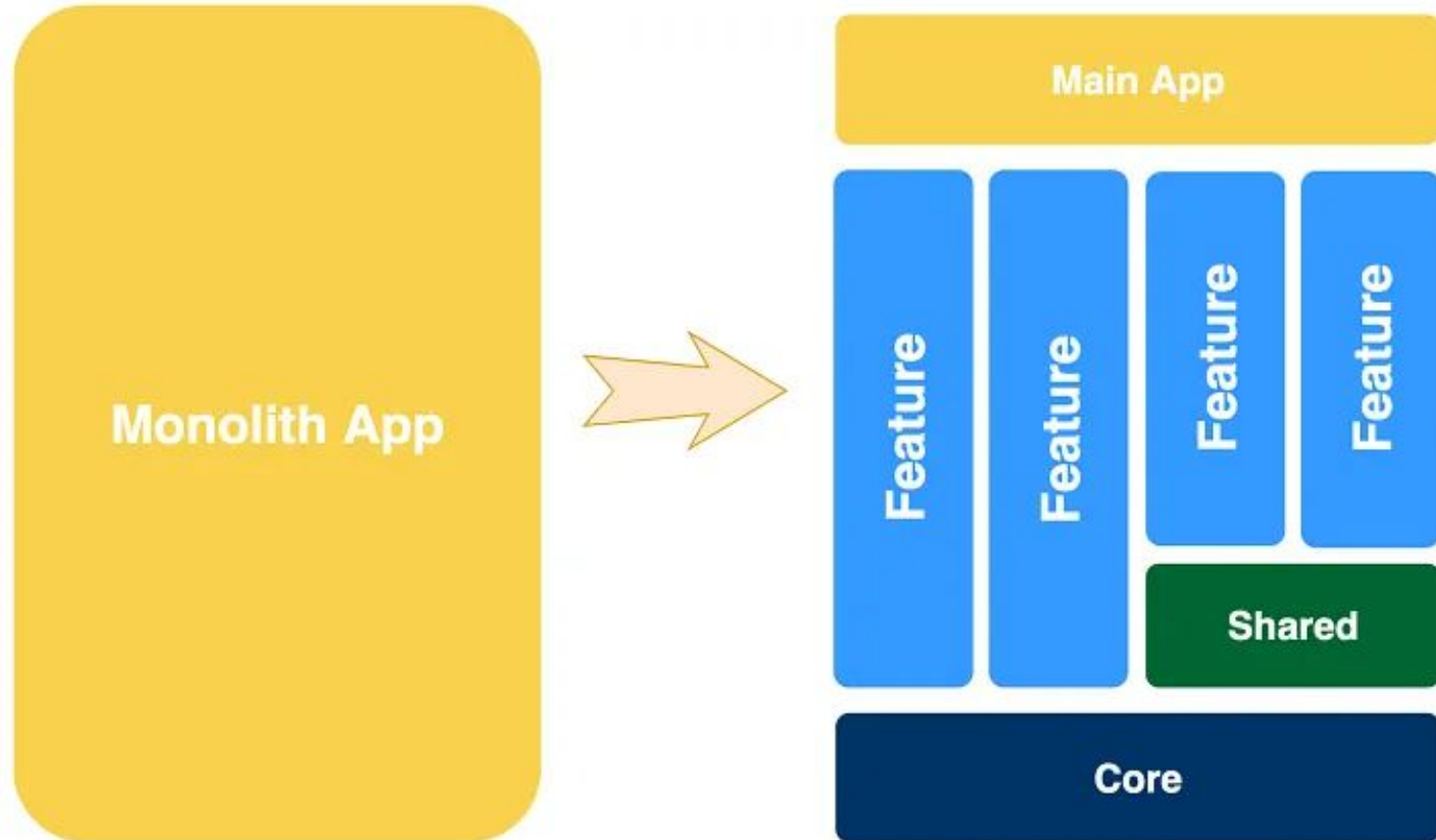
[Ver mais](#)





- Arquitetura orientada a pacotes
- Ideal para projetos de grande porte, os chamados super apps
- Pode diminuir o tempo de build da aplicação de 1 min para 10/20 segs
- Dividindo o app em pacotes nós também facilitamos o trabalho em equipes muito grandes







- Exemplos de inclusão de pacotes

```
carousel_pro:  
  git:  
    url: https://github.com/jlouage/flutter-carousel-pro.git  
    ref: main  
  
my_new_package:  
  path: ./my_new_package
```



- Podemos melhorar ainda mais a performance de nossas aplicações Flutter de acordo com o código que escrevemos
- Entendendo como o Flutter trabalha, podemos pensar em estratégias e formas otimizadas de construirmos nossas aplicações





O método build

- Devemos sempre nos preocuparmos em criar métodos build otimizados. Como os nossos widgets poderão ser redesenhados constantemente pelos seus widgets pais, o impacto de um método de build bem desenvolvido na performance da aplicação será considerável
- Evite utilizar um único *widget*, que pode ficar muito grande dentro do seu *build*
- Pode valer a pena separar esse *widget* em *widgets* menores





O método build

- Sempre que o `setState` é chamado, além do widget atual, todos os seus widgets filhos também serão redesenhados. Evite executar `setStates` em um nível muito alto da árvore de widgets, para evitar o redesenho total de muitos widgets filhos
- Uma boa dica é utilizar a palavra reservada `const` em widgets que são imutáveis, isso ajuda o Flutter a entender que alguns widgets não precisarão ser redesenhados
- O mesmo se aplica a construção de widgets em widgets de animação





Utilizando opacidade e Clips

- Evite, sempre que possível a utilização desnecessária de opacidade nos seus widgets de UI
- Os cálculos relacionados a opacidade podem ser muito custosos
- Os Clips são widgets que possibilitam a manipulação da forma de um widget filho, adicionando bordas arredondadas a um widget filho, por exemplo
- Embora não tão custoso quanto a utilização de opacidade, evite sua utilização
- Muitos widgets que utilizamos já possuem as propriedades necessárias para manipular sua forma





Trabalhando com listas e grids

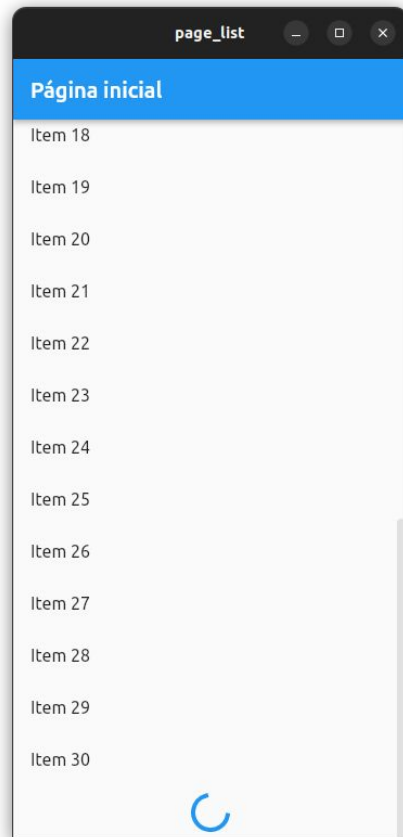
- Utilize o método Lazy: com o operador late nos nossos objetos, além de informarmos ao Dart que uma variável não será nula, também conseguimos otimizar o desempenho da nossa aplicação, pois essa variável somente será instanciada quando for necessária
- Utilize o construtor ListView.builder, ao invés de uma ListView simples. O construtor builder é mais otimizado e renderizará os itens em tela conforme a necessidade de exibi-los
- Utilize a propriedade prototypeItem, para auxiliar o motor de renderização da ListView a entender o conteúdo que será exibido pelos seus filhos





Trabalhando com listas e grids: Utilizando paginação

- Podemos implementar a utilização de paginação na nossa ListView








Utilizando PerformanceView





- Criar o cadastro de produtos
- Armazenar o campo nome e preço
- Utilize um gerenciador de estados, como Redux ou MobX

 Cadastro de produtos 

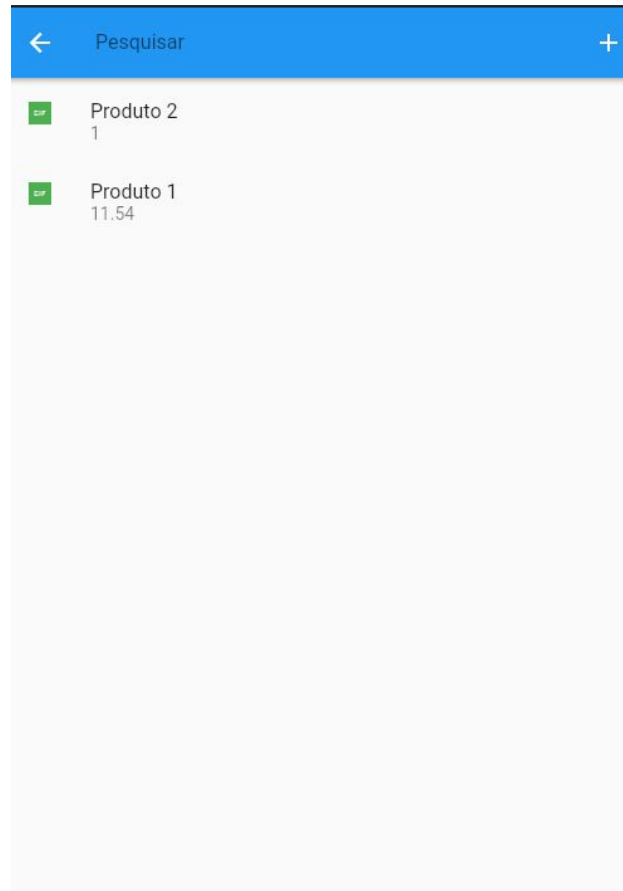
 **Produto**
Cadastro de produtos

Nome do produto

Preço do produto






- Criar a listagem de produtos com pesquisa
- Ao clicar no item, irá para o cadastro do produto
- Utilize um gerenciador de estados, como Redux ou MobX





- Criar a rotina de cadastro de usuário
- Utilizar a rotina de autenticação do Firebase
- Utilize um gerenciador de estados, como Redux ou MobX

 Cadastro de usuários 



Usuário
Cadastro de usuários

Nome do usuário

Senha