

Thinking in React

FUTURE4

Visão geral

O que já vimos

- O que é e para que serve React
- Tipos de componente
 - Funcional
 - Classe
- Props e estado
- Ciclo de vida
- Renderização condicional

Dificuldades

- O que deve ser um componente?
- O que deve ser guardado no estado?
- Em qual componente o estado deve ser guardado?
- Como e onde atualizar o estado?
- QQ EU FAÇO AGORA?

Sumário

- Quebrando em componentes
- Lidando com estado e props
- Estratégias para implementação
- Aprendendo a **pensar** em React

Problema para contextualização

Insta4

- Página com lista de posts - mostrar número de curtidas e número de comentários
- Cada post pode ser curtido
- Possui um form para criar um novo post

Passo 0

Criar ou identificar mocks

Criar ou identificar **mocks**


- Mock é um *modelo*, uma versão **estática** do que queremos
- Precisamos de dois mocks para começar:
 - Design
 - Dados

Mock de design

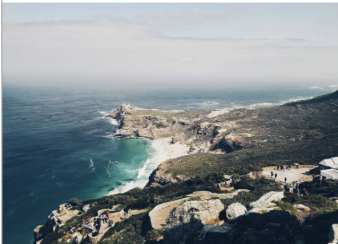
- Normalmente recebemos o mock de design de um designer
- Caso não tenhamos pronto, podemos fazer um rascunho (no papel ou computador) de como imaginamos nosso design



Mock de design


Criar um novo post




future4.br





 0  0



future4.br



 0  0

Mock de dados

- Normalmente recebemos os dados de um servidor
- Responsáveis pelo servidor irão nos passar o formato dos dados
- Se não tiver um servidor - ou se nós mesmos formos responsáveis - devemos definir o formato dos dados previamente
- Modelar os dados necessários para a aplicação
- Criar o mock com dados de exemplo

Mock de dados

```
const listaDePosts = [{  
  id: 1,  
  urlImagem: 'https://picsum.photos/300/200',  
  urlFotoUsuario: 'https://picsum.photos/100/100',  
  nomeUsuario: 'future4.br',  
  curtidoPorMim: false,  
  numeroCurtidas: 0,  
  comentarios: [  
    {  
      texto: 'Que foto incrivel!!',  
      curtidoPorMim: true  
    },  
    {  
      texto: 'Muito daora!!!',  
      curtidoPorMim: false  
    },  
  ],  
}]
```


Passo 1


Quebrar a UI em uma hierarquia de componentes


Quebrar a UI em uma hierarquia de componentes


- Quebrar o mock de design em componentes
- Como saber o que é um componente?
 - Arbitrário
 - Princípio da única responsabilidade - cada componente deve idealmente ser responsável por **só uma** coisa
- Dispor os componentes selecionados em uma hierarquia


Criar um novo post


 future4.br





 0

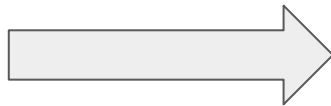
 0

 future4.br



 0

 0



- InstaContainer
 - FormNovoPost
 - Post
 - PostHeader
 - PostFooter
 - PostAction

Passo 2

*Construir uma versão **estática** em React*

Construir uma versão **estática** em React

- Devemos implementar uma versão que somente pega nosso mock de dados e renderiza na tela
- Todos os dados devem ser passados por props
 - **Não usar estado:** Estado é usado para interatividade. Aqui não temos nada de interatividade
- Mock de dados passado por props para componente pai

Estratégias de implementação


- Top-down
 - De cima para baixo: primeiro componentes mais acima da árvore
 - Mais simples para aplicações/componentes pequenos
- Bottom-up
 - De baixo para cima: primeiro componentes mais abaixo na árvore
 - Mais simples para grandes projetos


Considerações


- Fluxo de dados de cima para baixo, nunca de baixo para cima (componentes filhos não alteram dados dos componentes pais)
- Podemos já implementar condicionais, mas **somente baseado em props**


Mãos à massa!


Criar um novo post


 future4.br





 0

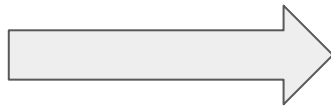
 0

 future4.br



 0

 0



- InstaContainer
 - FormNovoPost
 - Post
 - PostHeader
 - PostFooter
 - PostAction

Passo 3

*Identificar a representação **mínima** do estado da UI*

Identificar a representação **mínima** do estado da UI

- Estado serve para **interatividade**
- Precisamos identificar quais dados devem estar no estado e quais não
- Princípio: *DRY - Don't Repeat Yourself*
 - Dados não devem ser repetidos
 - Valores computados devem ser computados sob demanda - não guardá-los no estado.

Primeiro: identificar todos os dados

- Lista de posts
- Textos digitados pelo usuário

Segundo: para cada dado, se perguntar

1. É passado por um componente pai via props?
2. Não muda com o tempo?
3. Pode ser computado com base em outro valor?

Respostas afirmativas para as perguntas acima são **indícios** de que algo **não deve** ser guardado no estado

Devem estar no estado

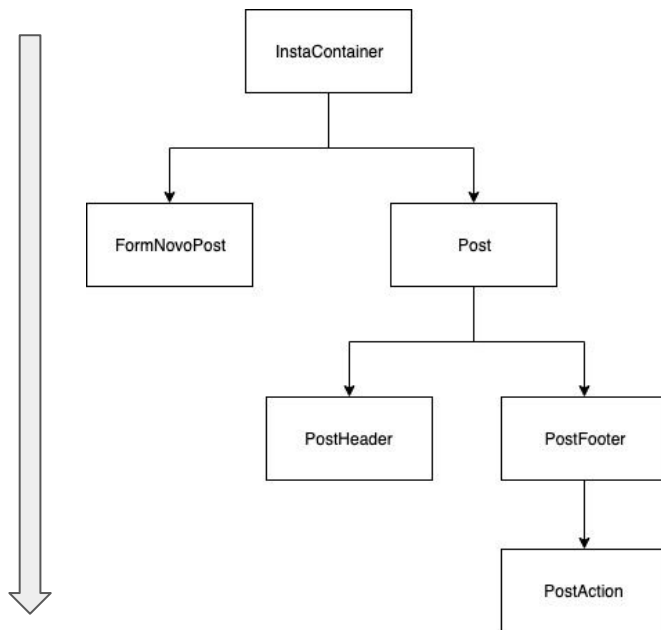
- Lista de posts
- Textos digitados pelo usuário

Passo 4

Identificar onde o estado deve morar

Identificar onde o estado deve morar

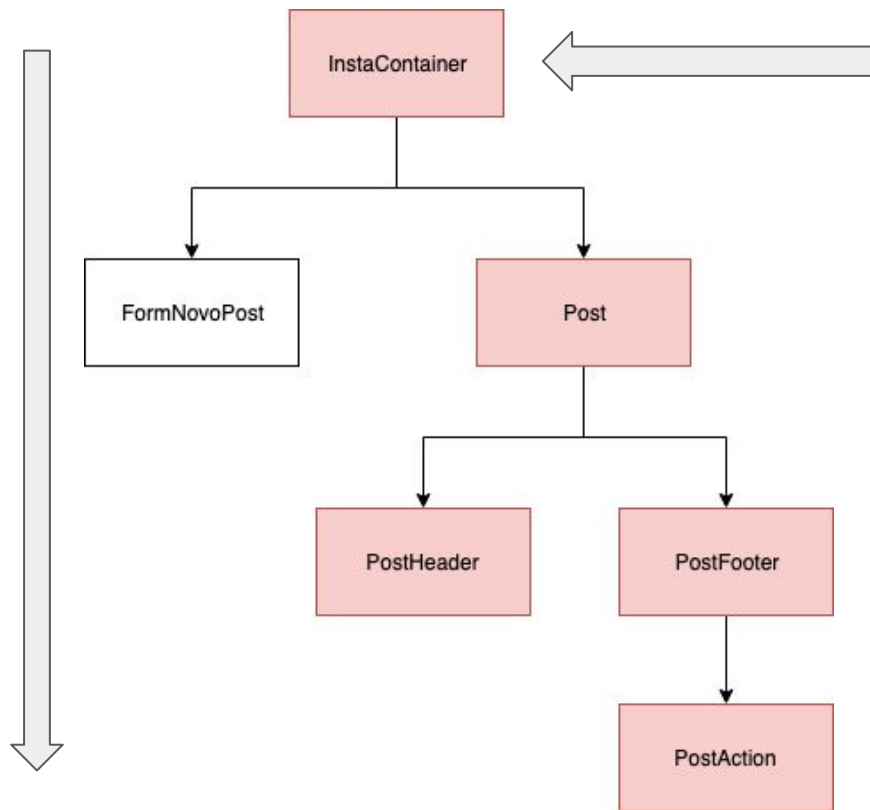
- Importante: o fluxo de dados do React tem um **único sentido** - de cima para baixo



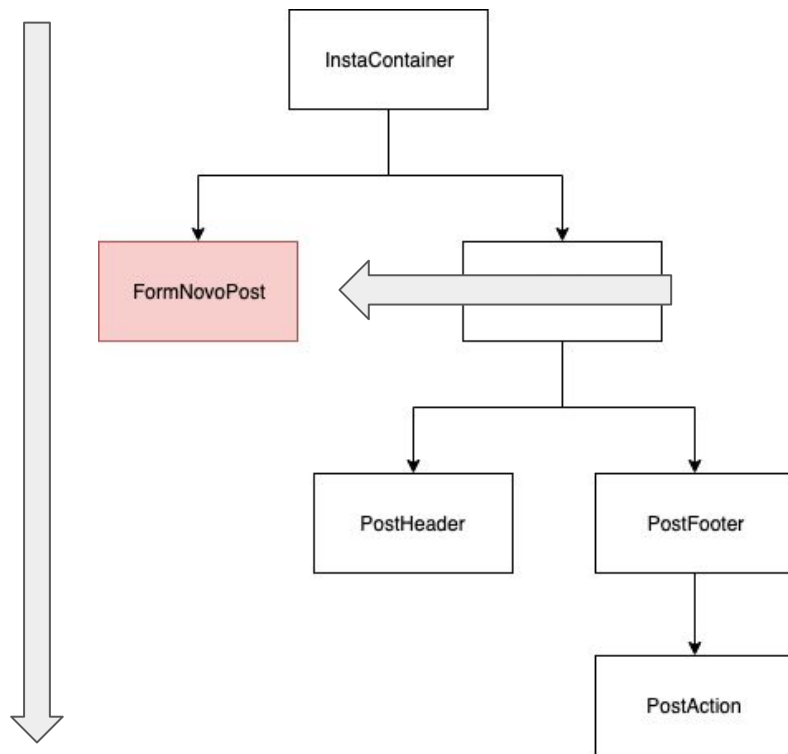
Como saber onde colocar o estado?

- Para cada dado do estado:
 - Identificar todos os componentes que renderizam algo que dependa dele
 - Encontre um componente acima na árvore que seja pai de todos os componentes identificados
 - Esse componente (ou algum pai dele) deve guardar o estado

Lista original de posts



- Textos digitados pelo usuário



Mãos à massa!

Mas e os outros elementos interativos? 🤔

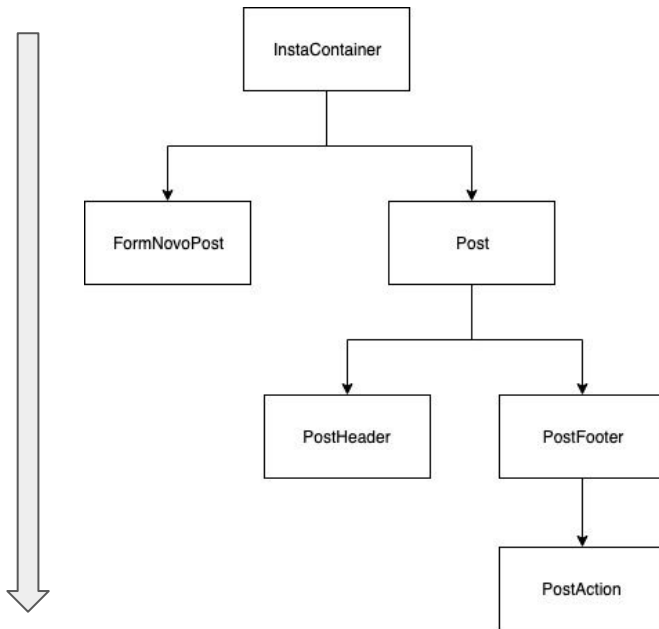
- Como lidar com likes e comentários?
- Normalmente, informações vêm do servidor
- Quando tomamos uma ação, enviamos informação para servidor, pedimos os dados novamente e então temos os dados atualizados
- Podemos simplificar, colocando valores nos estados locais, como fizemos semana passada

Passo 5

Adicionar fluxo de dados invertido

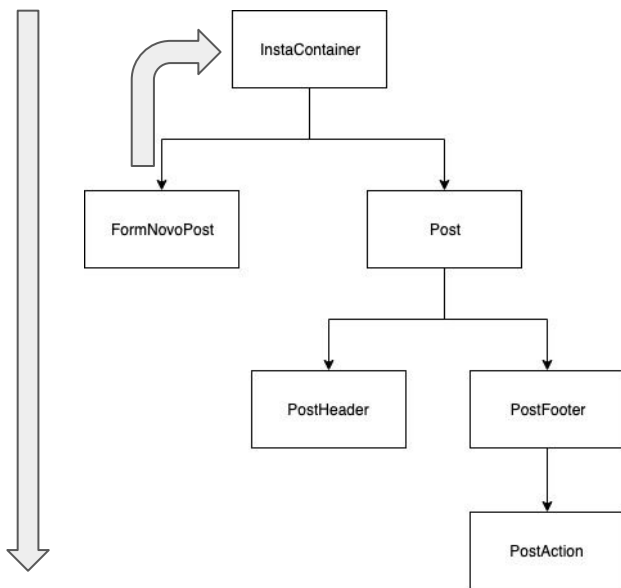
Adicionar fluxo de dados invertido

- Relembrando: o fluxo de dados do React tem um **único sentido** - de cima para baixo



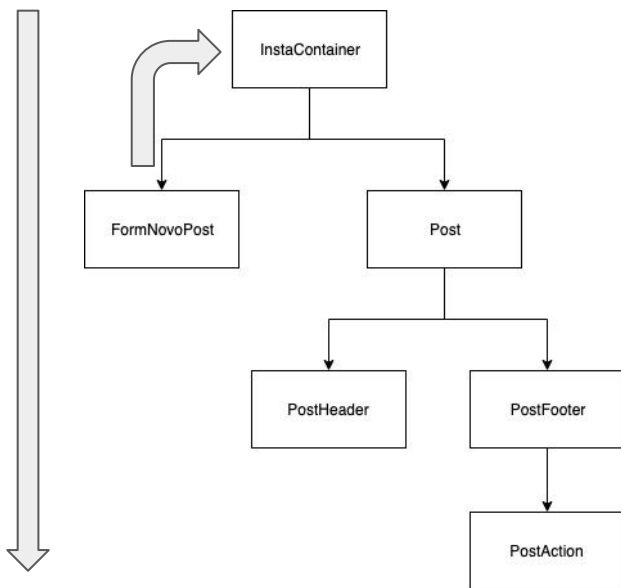
Adicionar fluxo de dados invertido

- As vezes precisamos que componentes mais abaixo da árvore mandem dados e alterem o estado de componentes pais



Adicionar fluxo de dados invertido

- As vezes precisamos que componentes mais abaixo da árvore mandem dados e alterem o estado de componentes pais



Adicionar fluxo de dados invertido

- Fazemos isso passando via props para os componentes filhos **funções** que irão atualizar o estado do componente pai
- Podemos passar essas funções por vários componentes

Estratégia

- Identificar componente que contém o estado e precisa ser mudado
- Criar função que atualiza estado. Dados que virão de componentes filhos devem ser recebidos como parâmetro
- Passar a função por props até o componente filho que possui os dados necessários
- Caso um componente intermediário possua os dados, criar nova função que chama o *callback*

Mãos à massa!

Dúvidas?

Obrigado!