React - Hooks

useEffect

O useEffect é um Hook no React que permite realizar efeitos colaterais em componentes, que não devem ocorrer durante a renderização normal do componente, como chamadas de API, manipulação do DOM (parte visual da página), assinaturas de eventos ou lidar com eventos, entre outras coisas. Usando esse Hook, você diz ao React que o componente precisa fazer algo apenas depois da renderização.

Você o usa quando precisa executar algum código em momentos específicos do ciclo de vida do seu componente. Pode ser quando o componente é exibido pela primeira vez, sempre que algo no seu componente muda ou quando o componente está prestes a ser removido.

Você o coloca dentro do seu componente e passa uma função para ele. Essa função será executada em momentos específicos, dependendo do que você especificar.

Vale mencionar o Hook useLayoutEffect que tem um funcionamento similar ao useEffect (usamos praticamente do mesmo jeito), mas é executado depois da renderização do browser. Ou seja, ele é executado após as mudanças no DOM (https://react.dev/reference/react/useLayoutEffect)

useEffect

Vamos usar o useEffect paga pegar o tamanho da tela do navegador, junto com o useState para conseguirmos ter esse valor atualizado mesmo quando a tela mudar de tamanho.

1. Vamos criar um novo componente, Exemple3.tsx
2. Criamos uma variavel de estado que vai ter a largura e altura da nossa janela
3. Vamos colocar esse valor para ser mostrado no nosso HTML
4. Vamos usar o componente no nosso App.tsx

hooks > src > components > Exemple3.tsx > ...

```tsx
import { useState } from "react"

const Exemple3Component = () => {
  const [windowSize, setWindowSize] = useState({
    width: window.innerWidth,
    height: window.innerHeight
  })

  return (
    <div>
      <h4>
        Window size
      </h4>
      <p>
        Width: {windowSize.width}<br />
        Height: {windowSize.height}
      </p>
    </div>
  )
}

export default Exemple3Component
```

useEffect

5. No estado atual do nosso componente, não tem nada atualizando nossas variaveis. Então só vamos conseguir ver o tamanho atual da nossa janela, quando atualizarmos a página.

useEffect

6. Vamos usar o useEffect para criar um listener de Javascript para atualizar a nossa variavel toda vez em que a tela mudar de tamanho.

Vamos observar a sintax do useEffect.

- O primeiro parâmetro sempre vai ser uma função anônima que vai ser executada pelo React
- O segundo parâmetro pode ser omitido, mas caso seja fornecido deve ser um Array.
- O retorno da função é executado quando o componente é destruído. Ou seja, quando ele não é mais renderizado pelo React, algo aconteça para o usuário sair da página, como um click ou alguma ação especifica. Essa ação também pode ser chamada de cleanup

hooks > src > components > Exemple3.tsx > ...

```tsx
import { useEffect, useState } from "react"

const Exemple3Component = () => {
  const [windowSize, setWindowSize] = useState({
    width: window.innerWidth,
    height: window.innerHeight
  })

  useEffect(() => {
    const handleResize = () => {
      setWindowSize({
        width: window.innerWidth,
        height: window.innerHeight
      })
    }

    window.addEventListener('resize', handleResize)

    return () => {
      window.removeEventListener('resize', handleResize)
    }
  }, [])

  return (
    <div>
      <h4>
        Window size
      </h4>
      <p>
        Width: {windowSize.width}<br />
        Height: {windowSize.height}
      </p>
    </div>
  )
}

export default Exemple3Component
```

useEffect

O segundo parâmetro do useEffect é um array que controla quando o efeito será executado. Dependendo dos valores dentro deste array, o comportamento do efeito pode variar. Esse parâmetro é chama de Array de dependências

Sem nenhuma depedência (omissão do array)

Nesse cenário a função será executada TODAS as vezes em que houver alguma mudança no componente. Seja de mudança de estado ou de propriedade. Muito cuidado ao usar dessa forma, pode deixar a aplicação pesada. O uso aqui é extremamente situacional

```
useEffect(() => {
  console.log('Efeito executado a cada renderização');
});
```

useEffect

Array Vazio

Quando o segundo parâmetro é um array vazio, o efeito será executado apenas uma vez, após a montagem inicial do componente. Este padrão é comum para inicializações, como buscar dados ou configurar subscrições.

```
useEffect(() => {
  console.log('Efeito executado apenas uma vez');
}, []);
```

useEffect

Com Dependências (Efeito executado com base nas mudanças das dependências)
Quando o segundo parâmetro é um array com uma ou mais dependências, o efeito será executado apenas quando uma dessas dependências mudar. Neste exemplo, o efeito será executado sempre que o valor de count mudar. Count pode ser tanto uma variavel de estado local quando uma propriedade herdada do componente pai.

```
useEffect(() => {
  console.log('Efeito executado quando count muda');
}, [count]);
```

useEffect

Com Dependências (Efeito executado com base nas mudanças das dependências)

Podemos ter mais de uma variavel no nosso array de dependências, com um funcionamento de logica OR. Ou seja, a função será disparada toda vez que count ou name sofrerem algum tipo de alteração. E no final estamos executando um cleanup.

```
useEffect(() => {
  console.log(`Count: ${count}, Name: ${name}`);

  return () => {
    console.log(`Cleanup para count: ${count} e name: ${name}`);
  };
}, [count, name]);
```

useEffect

1. Agora vamos criar outro componente e fazer uma requisição usando o useEffect. Vamos criar o Exemple4.tsx
2. Vamos criar duas variaveis de estado, uma que vai receber informações de endereço e a outra que vai controlar o nosso carregamento, uma variavel de loading.
3. Também criamos a interface do nosso endereço que será retornado por uma API publica de busca de CEP

nooks › src › components › Exemple4.tsx › Exemple4Component

```tsx
import { useEffect, useState } from "react"

interface Address {
  cep: string
  logradouro: string
  complemento: string
  unidade: string
  bairro: string
  localidade: string
  uf: string
  ibge: string
  gia: string
  ddd: string
  siafi: string
}

const Exemple4Component = () => {
  const [loading, setLoading] = useState(false)
  const [zipcode, setZipcode] = useState('')
  const [address, setAddress] = useState<Address>()

  if (loading) {
    return <p>Carregando...</p>
  }
```

useEffect

FIAP

4. Agora vamos criar um formulário HTML para realizar a busca do CEP

```
return (
  <div>
    <h5>Buscar Endereço</h5>
    <form onSubmit={() => setLoading(true)}>
      <label htmlFor='zipcode'>
        Digite o CEP (ex: 01001-000)<br />
        <input type="text" name='zipcode' id='zipcode' value={zipcode} onChange={e => setZipcode(e.target.value)} pattern="\d{5}-\d{3}" />
      </label>
      <br />
      <button type='submit'>
        Buscar
      </button>
    </form>
    {!loading && address?.logradouro && (
      <div>
        <h6>
          Resultado para o CEP: {cep}
        </h6>
        <p>{address?.logradouro}</p>
        <p>{address?.complemento}</p>
        <p>{address?.bairro}</p>
        <p>{address?.localidade}</p>
        <p>{address?.uf}</p>
        <p>{address?.cep}</p>
      </div>
    )}
  </div>
```

useEffect

FIAP

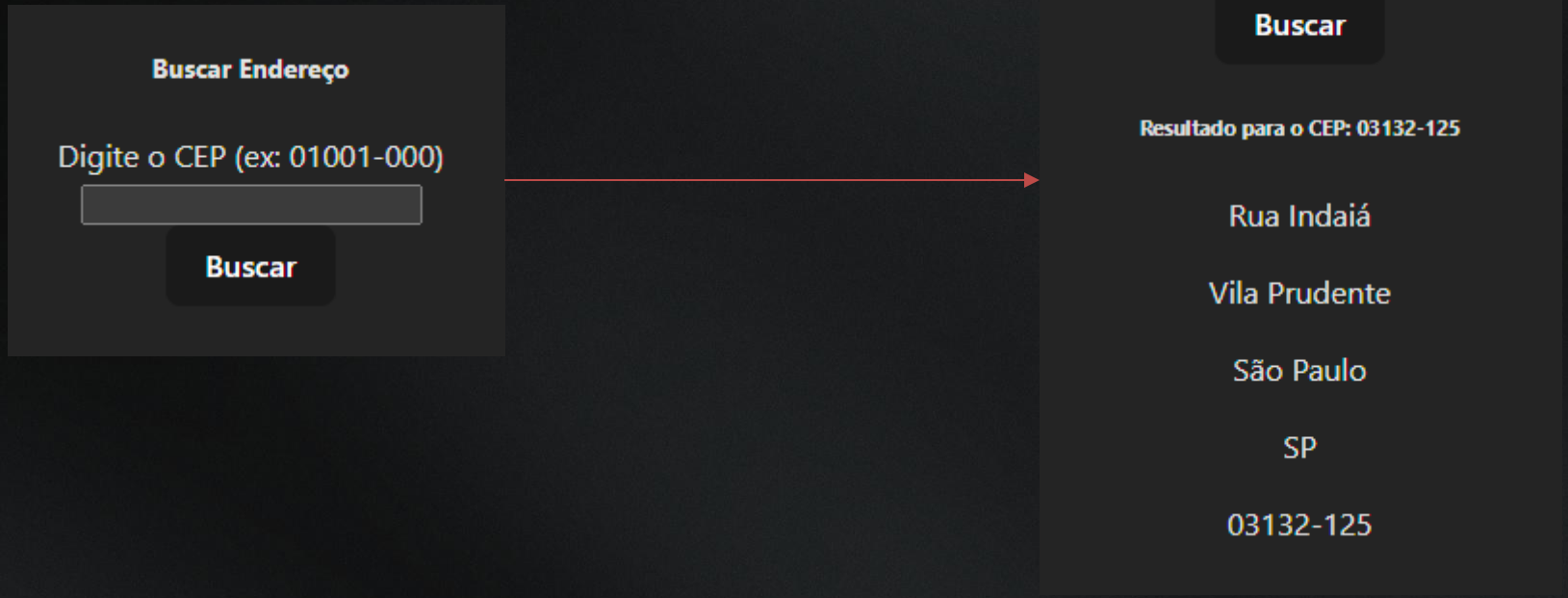
5. Por fim vamos criar um useEffect que vai ser disparado sempre que houver uma mudança na variavel loading e cep mudarem

```
useEffect(() => {
  if (!loading || !zipcode || zipcode.length < 9) {
    return
  }

  fetch(`https://viacep.com.br/ws/${zipcode}/json/`)
    .then(response => response.json())
    .then(data => setAddress(data))
    .catch(() => setAddress(undefined))
    .finally(() => setLoading(false))
}, [loading, zipcode])
```

useEffect

Resultado



useEffect

Fetch

O fetch é uma API moderna disponível no JavaScript para realizar solicitações HTTP. Ele permite que você busque recursos de servidores de maneira assíncrona, facilitando a comunicação entre o frontend e o backend de uma aplicação web.

Vamos ver mais sobre o fetch até o final do curso, mas caso queira saber mais:

- https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API
- https://www.w3schools.com/jsref/api_fetch.asp

useEffect

Exercicios

1. Continue o exercício do contador que fizemos com o useState, porém agora use o useEffect para exibir uma mensagem no console do navegador sempre que o valor do contador mudar.

2. Crie um componente de relógio digital que exibe a hora atual. Utilize o estado (useState) para armazenar a hora e o efeito (useEffect) para atualizar a hora a cada segundo. Adicione um botão que permite ao usuário pausar e retomar o relógio

Dúvidas, críticas ou sugestões?


FIAP