Para trabalhar com o react você pode escolher dois tipos de métodos o primeiro é usando-o em um projeto HTML pronto, para isso siga estes passos neste site <https://reactjs.org/docs/add-react-to-a-website.html> , a segunda forma é em um editor de texto instale o node,npm se possível na última versão e no terminal de comando digite:

npx create-react-app <nome da pasta>

Este comando irá um criar um projeto modelo, caso você já por um caso estiver usado um comando npm create-react-app -g é recomendável que você desinstale com o comando npm uninstall create-react-app e use o npx.

Depois de criado o react-app o aplicativo oferece quatro npm scripts que podem ser usados eles são:

npm start: este comando inicia um servidor para trabalhar no projeto, o servidor fica localizado na porta localhost:3000.

npm test: este comando cria um arquivo/projeto para examinar problemas.

npm run build: este comando junta o arquivo todo em um único projeto. Usado para fazer o deploy em sites como Netfly, Heroku, etc...

npm run eject: ele é um comando que funciona só no package.json e no npm modules **essa é uma operação de uma única direção. Uma vez que usou o**eject**, você não pode voltar atrás.**

<https://create-react-app.dev/docs/available-scripts/>

O react funciona como componentes isto é um aglomerado de funções em JSX que são sintaxes de HTML fundidas com javascript exemplo:

function Greeting(){

    return <h4> This is my first component! </h4>;

}

Funções em react só podem ser executadas se o nome inicial dela estiver em maiúsculo.

Quando criado o react-app exclua todos os arquivos da pasta src menos o index.js.

Para criar um componente precisamos importar os arquivos do react.

import React from 'react';

import ReactDom from 'react-dom';

Agora iremos criar um componente JSX

function Greeting(){

    return <h4> This is my first component! </h4>;

}

Para mostrar no servidor o que foi colocado precisamos chamar uma função para renderizar:

ReactDom.render(<*Greeting* />, document.getElementById('root'))

No código acima chamamos o ReactDom e usamos uma função para renderizar, dentro da função retornamos o nome dela abrindo <nome da função\> ou <nome da função><nome da função/> e depois chamamos o arquivo por um document.getElementById(‘nome do id do html principal’)

• CSS em React

Para usarmos o css no react podemos fazer de duas formas uma é importando um arquivo css criado na pasta assim:

import './index.css';

./ no import serve para informar que o arquivo esta na mesma pasta, e diferente do javascript que é só colocar o nome depois da barra no css precisamos do .css.

A outra forma de se inserir css no react e inserindo diretamente nas propriedades do HTML, para inserir uma classe em react precisamos criar assim diferentemente do html tradicional:

<article *className*='book'>

Diferente de só class =’’ usamos className no React.

Para inserir o css diretamente nas propriedades do html usamos o style porém inserimos duas chaves e dentro da segunda chave colocamos o css:

const Author = () => <h4 *style*={{color:'#617d98',fontSize:'0.75rem',marginTop:'0.25rem'}} >Dav Pilkey</h4>

No estilo clássico do html usaríamos assim:

const Author = () => <h4 *style*="color: #617d98; fontsize: 0.75rem;margin-top:0.25rem">Dav Pilkey</h4>

• Javascript em React

Para usarmos javascript em react podemos inserindo uma variável e dando um valor exemplo:

const title = 'Dog Man: Grime and Punishment: From the Creator of Captain Underpants'

A variável pode ser colocada dentro ou de fora da função no react, para chamar o valor dessa variável usamos duas chaves exemplo:

const Book = () => {

    const title = 'Dog Man: Grime and Punishment: From the Creator of Captain Underpants'

    const author = 'Dav Pilkey'

    return (

        <article *className*='book'>

             <img *src*="https://images-na.ssl-images-amazon.com/images/I/71c1LRLBTBL.\_AC\_UL200\_SR200,200\_.jpg" *alt*=""

            />

             <h1>{title}</h1>

            <h4 >{author}</h4>

            {/\*<p>{let x = 6}</p> Assim que se comenta em JSX\*/}

            <p>{6 + 6}</p>

        </article>

    )

}

Observe que criamos a variável title e chamamos ela sobre chaves assim mostrando o que está dentro dela. Tome cuidado para não criar variáveis dentro dos parênteses pois pode ocasionar erros.

Para fazer comentários em JSX usamos chaves e dentro dela colocamos /\* \*/.

• Props

Props são os parâmetros colocados em funções/componentes elas são divididas em diversos tipos. Por exemplo:

const Book = (*props*) => {

    return (

        <article *className*='book'>

             <img *src*={*props*.img} *alt*=""/>

             <h1>{*props*.title}</h1>

             <h4 >{*props*.author}</h4>

        </article>

    )

}

Criamos uma variável que guarda valores de objetos, dentre eles imagem, título e autor. Essas são as propriedades da variável.

const firstbook = {

    img:'https://images-na.ssl-images-amazon.com/images/I/71c1LRLBTBL.\_AC\_UL200\_SR200,200\_.jpg',

    title:'Dog Man: Grime and Punishment: From the Creator of Captain Underpants',

    author:'Dav Pilkey'

}

Depois no componente principal chamamos os valores por javascript informando os valores dentro do objeto.

function BookList(){

    return (

        <section *className*='booklist'>

           <*Book* *img*={firstbook.img} *title*={firstbook.title} *author*={firstbook.author} />

        </section>

    )

}

E lá no componente dos livros criamos a prop e nela informamos onde os valores serão colocados:

const Book = (*props*) => {

    return (

        <article *className*='book'>

             <img *src*={*props*.img} *alt*=""/>

             <h1>{*props*.title}</h1>

             <h4 >{*props*.author}</h4>

        </article>

    )

}

• Props Destruction

Vemos no exemplo anterior como fazer uma pop agora caso você queira retirar o uso de prop. <alguma coisa> toda hora basta destruir exemplo:

const Book = (*props*) => {

    const {img, title, author} = *props*

    return (

        <article *className*='book'>

             <img *src*={img} *alt*=""/>

             <h1>{title}</h1>

             <h4 >{author}</h4>

        </article>

    )

}

Criamos uma variável objeto guardando os valores utilizados e igualamos a props, com isso no return retiramos os props utilizados anteriormente e só informamos os valores. Você também pode desestrutar uma prop dentro do parêntesis inicial exemplo:

const Book = ({*img*, *title*, *author*}) => {

    //const {img, title, author} = props

    return (

        <article *className*='book'>

             <img *src*={*img*} *alt*=""/>

             <h1>{*title*}</h1>

             <h4 >{*author*}</h4>

        </article>

    )

}

• Props Children

Uma propriedade children basicamente é tudo que é colocado entre o início do componente e o final do componente exemplo:

  <*Book* *img*={firstbook.img} *title*={firstbook.title} *author*={firstbook.author}

           >

               <p>

                Excepteur non eu excepteur et quis laboris voluptate aliqua consequat. Mollit voluptate nulla magna laborum exercitation amet sint sunt commodo nulla reprehenderit ullamco quis ea. Duis ipsum cupidatat exercitation exercitation esse ex laboris quis magna. Reprehenderit amet est commodo minim Lorem proident enim.

               </p>

           </*Book*>

Observe que colocamos um paragrafo entre o componente, sozinho esse componente não mostra nada porém se chamarmos no javascript:

const Book = ({*img*, *title*, *author*, *children*}) => {

    //const {img, title, author} = props

    return (

        <article *className*='book'>

             <img *src*={*img*} *alt*=""/>

             <h1>{*title*}</h1>

             <h4 >{*author*}</h4>

            {*children*}

        </article>

    )

}

Para que isso funcione temos que chamar a propriedade de children e depois chama-la usando as 2 chaves. Colocamos o parágrafo no final da box, mas caso quiser colocar antes da imagem, antes do autor também é possível basta colocar ele lá.

• Key Prop

Key props basicamente são valores únicos de uma variável, em react normalmente as key props serão alcançadas pela api.

const books = [

    {

        id: 1,

        img:'https://images-na.ssl-images-amazon.com/images/I/71c1LRLBTBL.\_AC\_UL200\_SR200,200\_.jpg',

        title:'Dog Man: Grime and Punishment: From the Creator of Captain Underpants',

        author:'Dav Pilkey'

    },

    {

        id: 2,

        img:'https://images-na.ssl-images-amazon.com/images/I/91AQs6qv9ML.\_AC\_UL200\_SR200,200\_.jpg',

        title:'Untamed',

        author:'Glennon Doyle'

    },

    {   id: 3,

        img:'https://images-na.ssl-images-amazon.com/images/I/81Kc8OsbDxL.\_AC\_UL200\_SR200,200\_.jpg',

        title:'Greenlights',

        author:'Matthew McConaughey'

    },

]

No caso desse exemplo o valor único dentro da variável e dos objetos é o id.

E para chamar a propriedade chave usamos um comando key = {parâmetro. valor único}

function BookList(){

    return (

        <section *className*='booklist'>

            {books.map((*book*)=>{

                return(

                    <*Book* *key*={*book*.id} *book*={*book*}></*Book*>

                )

            })}

        </section>

    )

}

Advanced React

Para estudar o advanced react baixe ou clone o arquivo: https://github.com/john-smilga/react-advanced-2020

React Hooks

Hooks são funções que permitem a você “ligar-se” aos recursos de state e ciclo de vida do React a partir de componentes funcionais. Hooks não funcionam dentro de classes — eles permitem que você use React sem classes.

O hooks são divididos em useState e useEffect.

useState Básico

O useState é uma function e por isso na hora de importar ela precisamos colocar entre chaves.

import React, { useState } from 'react';

Primeiro criamos uma variável UseStateBasics que é igual a uma arrow function dentro da arrow function iremos colocar os dados para o useState.

const UseStateBasics = () => {

};

No final adicione uma export default para que ele vá para o App.js.

export default UseStateBasics;

Como o useState é uma função podemos evocar ela usando um console.log .

console.log(useState())

Repare que a função useState é um array e que o primeiro valor é um valor indefinido, e o segundo valor é o valor de uma função com isso iremos criar a estrutura do useState. Primeiro criamos uma variável e um array, dentro do array criamos o valor que irá ser chamado e a função que irá chamar esse valor e em seguida chamamos o useState.

const [text,setText] = useState('Random Title')

Depois insira um return dentro de UseStateBasics e crie um React.Fragment dentro ele pode ser feito de duas maneiras a primeira seria usando <React.Fragment></React.Fragment> ou nas versões mais novas simplesmente usando <> </>. Criaremos um h1 e chamaremos o titulo que usamos no useState, depois criaremos um botão Change Title C:\Users\PC\Documents\FrontEnd Roadmap\React\react-advanced-2020\src\tutorial\1-useState\setup\1-error-example.jse um onclick chamando uma função chamada handleClick

return (

    <*React.Fragment*>

      <h1>{text}</h1>

      <button *className*="btn" *onClick*={handleClick}>Change Title</button>

    </*React.Fragment*>

  )

Logo após crie uma função para chamar o handleClick:

const handleClick = () => {

  }

E como estamos fazendo um exemplo básico de só como mudar o titulo no react e só criar um if/else que ela irá mudar o texto do título. E dentos do if/else chame o setText e nomei-o diferente para testar.

 if(text === 'Random Title' ){

      setText('hello world')

    }else{

      setText('Random Title')

    }

Final:

import React, { useState } from 'react';

const UseStateBasics = () => {

  const [text,setText] = useState('Random Title')

  const handleClick = () => {

    if(text === 'Random Title' ){

      setText('hello world')

    }else{

      setText('Random Title')

    }

  }

return (

    <*React.Fragment*>

      <h1>{text}</h1>

<button *className*='btn'*onClick*={handleClick}>Change Title</button>

    </*React.Fragment*>

  )

};

export default UseStateBasics;

useState em Arrays

Nesse caso iremos usar um array armazenado em arquivo data.js e importa-lo para o useState usando o comando:

import { data } from '../../../data';

O array do arquivo data.js possui os seguintes dados e para poder usa-lo precisamos exportar a variável que está guardando os dados:

export const data = [

  { id: 1, name: 'john' },

  { id: 2, name: 'peter' },

  { id: 3, name: 'susan' },

  { id: 4, name: 'anna' },

  { id: 5, name: 'moses' },

  { id: 6, name: 'olivia' },

  { id: 7, name: 'sara' },

];

Agora no arquivo do useState crie uma variável para armazenar um componente e no final exporte este componente.

const UseStateArray = () => {

};

export default UseStateArray;

Agora no caso de array de fora precisamos chamar o useState de uma forma diferente primeiro criamos uma variável que tem array que armazena o valor e a função do valor igual no exemplo passado e em seguida usamos o React importado e chamamos o useState armazenando os dados vindos do data.js.

const [people, setPeople] = *React*.useState(data);

Depois disso criamos um return e dentro desse return adicionamos um react fragmente e nele adicionaremos uma chave para a div suit. Sem a div suit não é possível chamar a variável people dentro return.

 return (

  <>

{}

  </>

 )

Agora chamaremos a variável people e para traçar todo o array usaremos a função do javascript map(), e dentro desse usaremos uma arrow function com um parâmetro de person para identificar cada um. Em seguida queremos encontrar o id e o nome da pessoa por isso precisamos desestruturar e retornar o valor para o parâmetro.

 return (

  <>

{people.map((*person*) => {

const {id, name} = *person*;

return (

)

})}

</>

)

Dentro desse return solto precisamos retornar uma div e dentro dessa div. Precisamos adicionar a keyprop redirecionando pro id que a chave primaria/vulgo valor único, em seguida adicionamos uma className ‘item’ e um h4 chamando o nome e um botão para remover.

return (

  <>

{people.map((*person*) => {

const {id, name} = *person*;

return (

<div *key*={id} *className*='item'>

              <h4>{name}</h4>

    </div>

);

})};

</>

)

Agora fora dessa div suit adicionaremos um botão para limpar todos os nomes da lista adicionamos uma className ’btn’ e um onclick e para chamar precisamos usar uma arrow function e em seguida chamar a função setPeople e invocar um array dentro e como queremos simplesmente apagar todos colocamos só o array vazio.

return (

  <>

{people.map((*person*) => {

const {id, name} = *person*;

return (

<div *key*={id} *className*='item'>

              <h4>{name}</h4>

    </div>

);

})};

<button *className*='btn' *onClick*={() => setPeople([])}>

      clear items

    </button>

    </>

</>

)

Agora caso você queira criar um botão para excluir um item individual precisamos criar ele dentro da div. Para isso usamos o onclick e dentro do onclick utilizamos uma arrow function de nome removeItem e dentro utilizamos por qual parâmetro queremos excluir. Após fazer isso é normal ocorrer um erro porque não criamos uma função removeItem, e esse será o próximo passo.

return (

  <>

{people.map((*person*) => {

const {id, name} = *person*;

return (

<div *key*={id} *className*='item'>

              <h4>{name}</h4>

<button *onClick*={() => removeItem(id)}> remove </button>

    </div>

);

})};

<button *className*='btn' *onClick*={() => setPeople([])}>

      clear items

    </button>

    </>

</>

)

Fora do return criaremos a função removeItem para isso criamos uma arrow function e procuramos pelo id. A primeira é a mais simples é só invocar a função setPeople que irá funcionar a segunda forma e de uma forma mais explicita mostrando passo a passo. Na segunda forma criamos uma variável newPeople ela irá passar pelos valores de people usando a function filter e dentro desse filter você irá passar uma arrow function com os parâmetros person e os parâmetros de person.id é diferente de id ele irá remover a pessoa.

 const removeItem = (*id*) => {

    let newPeople = people.filter((*person*) => *person*.id !== *id*)

    setPeople(newPeople)

//setPeople()

  }

Final.

import React from 'react';

import { data } from '../../../data';

const UseStateArray = () => {

  //

  const [people, setPeople] = *React*.useState(data);

  const removeItem = (*id*) => {

    let newPeople = people.filter((*person*) => *person*.id !== *id*)

    setPeople(newPeople)

    //setPeople()

  }

  return (

    <>

    {people.map((*person*) => {

        const {id, name} = *person*;

        return (

            <div *key*={id} *className*='item'>

              <h4>{name}</h4>

              <button *onClick*={() => removeItem(id)}> remove </button>

            </div>

        );

    })}

    <button *className*='btn' *onClick*={() => setPeople([])}>

      clear items

    </button>

    </>

  )

};

export default UseStateArray;

useState Object

Iremos fazer uma aplicação que muda a mensagem para outra. Para isso primeiro iremos fazer como fizemos na primeira aplicação criar uma variável com um array e dentro desse array colocar o valor e uma função. E no final exportar como default para o App.js.

const UseStateObject = () => {

  const [person, setPerson]

};

export default UseStateObject;

Como nas aplicações passadas colocamos um igual a useState e dentro desse criamos um objeto com três propriedades elas são ‘name’,’age’ e ‘message’ o objetivo é mudar a mensagem quando clicar no botão.

const UseStateObject = () => {

const [person, setPerson] = useState({

    name:'Peter',

    age:'24',

    message:'random message!'

  })

};

export default UseStateObject;

Depois criaremos um return com um react fragment e dentro dele iremos chamar as propriedades com o jsx e depois criar um botão para mudar a mensagem com um onclick ‘changeMessage’.

return <>

    <h3>{person.name}</h3>

    <h3>{person.age}</h3>

    <h3>{person.message}</h3>

 <button *className*="btn" *onClick*={changeMessage}>

      Change Message!

  </button>

</>

Como não criamos uma função para mudar a mensagem a parte do botão não irá funcionar portanto agora iremos criar a função do botão, criaremos uma variável com arrow function e iremos setar o setPerson para uma nova mensagem.

  const changeMessage = () => {

    setPerson('hello world!')

  }

O problema nesse caso é quando você invoca a função ela irá apagar tudo e não irá mudar a mensagem pois o setPerson está invocando uma string e não um objeto.

  const changeMessage = () => {

   setPerson({ message:'hello world!'})

  }

Nesse caso ele irá mudar só a mensagem, porém ele irá excluir tudo acima dessa mensagem. E a solução nesse caso é usar um Spread Operator para mostrar todos os dados e especificar o dado que será mudado.

  const changeMessage = () => {

    setPerson({...person, message: 'hello world!'})

  }

Final.

import React, { useState } from 'react';

const UseStateObject = () => {

  const [person, setPerson] = useState({

    name: 'peter',

    age: 24,

    message: 'random message',

  });

const changeMessage = () => {

    setPerson({ ...person, message: 'hello world' });

}

return (

    <>

      <h3>{person.name}</h3>

      <h3>{person.age}</h3>

      <h4>{person.message}</h4>

      <button *className*='btn' *onClick*={changeMessage}>

        change message

      </button>

    </>

  );

};

export default UseStateObject;

A uma segunda forma, porém ela não é eficiente para isso precisamos criar um useState para cada objeto e mudar a função do botão e os dados do return o código ficaria assim. Ele funciona, porém, ele dá mais trabalho

import React, { useState } from 'react';

const UseStateObject = () => {

  const [person, setPerson] = useState({

    name:'Peter',

    age:'24',

    message:'random message!'

  })

const [name, setName] = useState('peter')

const [age, setAge] = useState(24)

const [message, setMessage] = useState('random message!')

const changeMessage = () => {

 setMessage('hello world!')

 }

 <h3>{name}</h3>

 <h3>{age}</h3>

 <h3>{message}</h3>

<button *className*="btn" *onClick*={changeMessage}>

      Change Message!

  </button>

 </>

};

export default UseStateObject;

.

useEffect Basics

O useEffect Hook permite que você trabalhe com qualquer trabalho de fora do componente por exemplo mudar o titulo do documento, se inscrever no botão de inscrição, buscar dados, colocar um eventListenner e outras coisas do gênero.

Agora iremos mudar no App.js mudar o local do projeto e procurar pela pasta useEffect/setup/1-useEffect-basics

import React from 'react'

import Setup from './tutorial/2-useEffect/setup/1-useEffect-basics';

function App() {

  return (

    <div *className*='container'>

      <*Setup*/>

    </div>

  )

}

export default App

Agora iremos para o arquivo 1-useEffect-basics e vemos que também teremos que importar o useEffect como fizemos no useState. O useEffect funciona passando ele dentro de uma call-back function vai rodar dentro e irá rodar sempre após uma nova renderização.

import React, {useState, useEffect } from 'react';

// by default runs after every re-render

// cleanup function

// second parameter

const UseEffectBasics = () => {

  useEffect(()=>{

    console.log('call useEffect')

  })

  console.log('render component')

return (

    <>

    <h2>use Effect Basics</h2>

  )

};

export default UseEffectBasics;

Criamos um console.log para observar quando o componente é renderizado e quando o useEffect. Se abrir o console observamos que quando rederizamos o componente logo abaixo estará a mensagem chamando o useEffect caso apareça dois render componente é porque no arquivo index.js criamos um React.StrictMode.

Agora vamos testar se podemos mudar o titulo da página primeiro iremos criar um botão que ao clicar irá incrementar o valor que está no título em + 1.

import React, { useState, useEffect } from 'react';

// by default runs after every re-render

// cleanup function

// second parameter

const UseEffectBasics = () => {

  const [value,setValue] = useState(0)

  useEffect(()=>{

    console.log('call useEffect')

  })

  console.log('render component')

return (

    <>

    <h1>{value}</h1>

    <button *className*="btn" *onClick*={()=>{setValue(value+1)}}> click me</button>

    </>

  )

};

export default UseEffectBasics;

Primeiro criamos o botão e colocamos um className “btn” e um onclick = {}, e colocamos um nome click me no botão. Depois iremos criar um valor para o useState const [value, setValue] e informar esse useState o valor inicial é 0. Dentro do return informarmos o valor que queremos mostrar que está guardado no useState <h1>{value}</h1> agora dentro do botão no return especificamente dentro do onClick chamamos o setValue e incrementamos + 1 onClick = {() =>setValue(value+1)}

Agora observando toda a vez que você clica observe no console que você chama o render componente e logo em seguida o call useEffect e isso ocorre por causa do useState ele proporciona 2 coisas, preserva o valor entre as renderizações e sempre gatilho para a re-renderização.

import React, { useState, useEffect } from 'react';

// by default runs after every re-render

// cleanup function

// second parameter

const UseEffectBasics = () => {

  const [value,setValue] = useState(0)

  useEffect(()=>{

    console.log('call useEffect')

document.title = `New Messages ${value}`

  })

  console.log('render component')

return (

    <>

    <h1>{value}</h1>

    <button *className*="btn" *onClick*={()=>{setValue(value+1)}}> click me</button>

    </>

  )

};

export default UseEffectBasics;

Adicionando a funcionalidade de mudar o título document.title = `New Messages ${value}` agora toda vez que clicamos no botão também atualizaremos o título.

useEffect Conditional

Agora vamos supor um exemplo que eu quero implementar caso os valores forem maiores ou iguais a 1 com o useEffect é possível usar condicionais. Como não é possível colocar hooks dentro de condicionais o useEffect permite você colocar condicionais dentro da call-back function.

import React, { useState, useEffect } from 'react';

// by default runs after every re-render

// cleanup function

// second parameter

const UseEffectBasics = () => {

  const [value,setValue] = useState(0)

  useEffect(()=>{

    console.log('call useEffect')

if (value >= 1){

       document.title = `New Messages ${value}`

    }

  })

  console.log('render component')

return (

    <>

    <h1>{value}</h1>

    <button *className*="btn" *onClick*={()=>{setValue(value+1)}}> click me</button>

    </>

  )

};

export default UseEffectBasics;

useEffect Dependency List

Quando usamos o useEffect existe um segundo parâmetro que pode ou não ser passado para usar dele utilizamos um array [] logo depois da função de call-back quando colocamos e apertamos o botão repare que o useEffect será chamado apenas na primeira vez que for iniciado caso você queira chamar ele apenas no começo use um array vazio. Mas por exemplo você quer atualizar a dependência você precisa passar o valor no array exemplo:

import React, { useState, useEffect } from 'react';

// by default runs after every re-render

// cleanup function

// second parameter

const UseEffectBasics = () => {

  const [value,setValue] = useState(0)

  useEffect(()=>{

    console.log('call useEffect')

if (value >= 1){

       document.title = `New Messages ${value}`

     }

},[value])

  console.log('render component')

return (

    <>

    <h1>{value}</h1>

    <button *className*="btn" *onClick*={()=>{setValue(value+1)}}> click me</button>

    </>

  )

};

export default UseEffectBasics;

Nada impede que você possa criar mais de um useEffect por exemplo se eu colocar mais um e quiser colocar ele para mostrar só no início com um ‘hello world’ repare que o log só ira mostrar no inicio e depois que você apertar o botão mais vezes ele não irá mostrar mais.

import React, { useState, useEffect } from 'react';

// by default runs after every re-render

// cleanup function

// second parameter

const UseEffectBasics = () => {

  const [value,setValue] = useState(0)

  useEffect(()=>{

    console.log('call useEffect')

    if (value >= 1){

      document.title = `New Messages ${value}`

    }

  },[value])

  useEffect(()=>{

      console.log('hello world')

  }, [])

  console.log('render component')

  return (

    <>

    <h1>{value}</h1>

    <button *className*="btn" *onClick*={()=>{setValue(value+1)}}> click me</button>

    </>

  )

};

export default UseEffectBasics;

useEffect – Cleanup Function

Agora vamos mexer com outro arquivo. Mude o arquivo basic para o cleanup.

import React from 'react'

import Setup from './tutorial/2-useEffect/setup/2-useEffect-cleanup';

function App() {

  return (

    <div *className*='container'>

      <*Setup*/>

    </div>

  )

}

export default App

Agora no arquivo de cleanup iremos mostrar o tamanho da tela e como ela muda conforme você redefine o tamanho dela. Primeiro vamos criar um useState para o tamanho const[size, setSize] = useState() e dentro colocamos window.innerWidth depois colocamos criamos um console.log(size) e olhamos o console. O console mostrará o tamanho da tela.

import React, { useState, useEffect } from 'react';

// cleanup function

// second argument

const UseEffectCleanup = () => {

  const [size,setSize] = useState(window.innerWidth)

  console.log(size)

return (

<>

  <h2>useEffect cleanup</h2>

  </>

  );

};

export default UseEffectCleanup;

Agora dentro do return com o fragmente criaremos um <h1>window<h1/> informando a tela e um <h2>{size}<h2/> informando o tamanho da tela. Depois iremos criar um eventListenner, aprendemos que para esses eventos precisamos criar um useEffect e adicionar dentro dele window.addEventListener() como evento ‘resize’ e criar uma função para checar o tamanho e daremos o nome de checkSize. Para criar a função criaremos uma variável checkSize com uma call-back function chamando a function setSize com a função window.innerWidth

import React, { useState, useEffect } from 'react';

// cleanup function

// second argument

const UseEffectCleanup = () => {

  const [size,setSize] = useState(window.innerWidth)

  const checkSize = () => {

    setSize(window.innerWidth)

  }

  useEffect(()=>{

    window.addEventListener('resize', checkSize)

  })

  console.log(size)

return (<>

  <h1>window</h1>

  <h2>{size} pX</h2>

  </>

  );

};

export default UseEffectCleanup;

Podemos ver que funciona tudo corretamente não é porem vamos no console vá em Elements e do lado Event Listeners e vemos o evento resize observe que era só para ocorrer um evento porém ocorreu mais de um, o problema desse monte de eventos e que ele pode causar grandes problemas no futuro. E é agora que a cleanup function vem dentro do useEffect temos a opção de retornar uma function. Para isso criamos um return com uma call-back function com um evento para remover o eventListenner window.removeEventListener(‘resize’, checksize). Agora checando no console você consegue enxergar o evento só vai ser ativado uma vez.

import React, { useState, useEffect } from 'react';

// cleanup function

// second argument

const UseEffectCleanup = () => {

  const [size,setSize] = useState(window.innerWidth)

  const checkSize = () => {

    setSize(window.innerWidth)

  }

  useEffect(()=>{

    console.log('hello world')

    window.addEventListener('resize', checkSize)

    return () => {

      console.log('cleanup')

      window.removeEventListener('resize', checkSize)

    }

  }, [])

  console.log(size)

  return (<>

  <h1>window</h1>

  <h2>{size} pX</h2>

  </>

  );

};

export default UseEffectCleanup;

useEffect – Fetch Data

Primeiro vamos mudar o arquivo no App.js para:

import React from 'react'

import Setup from './tutorial/2-useEffect/setup/3-useEffect-fetch-data';

function App() {

  return (

    <div *className*='container'>

      <*Setup*/>

    </div>

  )

}

export default App

Agora no arquivo onde vamos fazer o ligamento dos dados vemos que ele já vem com uma url informando a api. A primeira coisa a se fazer é criar um useState para os usuários const [users, setUsers] = useState([]) esse será o set up inicial depois mexeremos no <h3><h3> e mudaremos o nome para github users. E depois setar o useEffect uma das regaras do useEffect e que não podemos usar async/await na call-back do hooks pois retorna uma promise, porém ele permite usar dentro ou fora, no exemplo iremos criar o fetch de fora. Criaremos uma função getUsers = async() => {} dentro dessa função procuramos por uma resposta então criamos a variável const response = await fetch(url) e também queremos os usuários const users = await response.json() agora criamos um console.log(users) e observamos no console primeiro e vemos que não tem nada porque não chamamos o getUsers coloque dentro do useEffect e veja os dados. Agora caso você chamar setUsers(users) logo após de fazer os passos a página que você está mexendo entrará em um loop infinito e irá crashar porque está gatilhando um re-render para resolver isso basta colocar um [] vazio após o useEffect. Agora para iterar os dados no return crie uma lista com a className=”users” dentro da lista crie {users.map ((user) => {})} dentro da chave destrua os objetos e salve em um parâmetro que no caso é user, const {id, login, avatar\_url, html\_url} = user ,crie um return abaixo da variável. Esse return tem que retornar uma linha para a lista com uma keyprop < li key ={id}></li> dentro dessa linha cria uma imagem retornando src para o avatar e o alt para o login <img src={avatar\_url} alt={login}></img> abaixo crie uma div com um <h4>{login}<h4/> e logo abaixo uma ancora para redirecionar para o perfil do usuário <a href={html\_url}>profile</a>.

import React, { useState, useEffect } from 'react';

const url = 'https://api.github.com/users';

const UseEffectFetchData = () => {

  const [users,setUsers] = useState([])

  const getUsers = async() => {

    const response = await fetch(url)

    const users = await response.json()

    setUsers(users)

    console.log(users)

  }

  useEffect (()=>{

      getUsers()

  },[])

  return(

    <>

      <h3>GitHub users</h3>

      <ul *className*="users">

          {users.map ((*user*) => {

            const {id, login, avatar\_url, html\_url} = *user*;

          return <li *key* ={id} >

              <img *src*={avatar\_url} *alt*={login}></img>

              <div>

                <h4>{login}</h4>

                <a *href*={html\_url}>profile</a>

              </div>

            </li>

          })}

      </ul>

    </>

  );

};

export default UseEffectFetchData;

Multiple Returns

Primeiro vamos alterar o arquivo App.js no import para trabalhar com o multiple returns.

import React from 'react'

import Setup from './tutorial/3-conditional-rendering/setup/1-multiple-returns';

function App() {

  return (

    <div *className*='container'>

      <*Setup*/>

    </div>

  )

}

export default App

Pelas regras do javascript quando temos dois returns o primeiro return é o que será ativado e todo o resto será ignorado. Porém estamos procurando por algumas condições e baseadas nessas condições serão retornados tipos de conteúdo. Primeiro com um exemplo básico vamos criar um useState e informar o valor boolean true: const [loading, setLoading] = useState(true) e agora que temos o nosso useState vamos fazer uma condição if(loading){return <h2>Loading...</h2>}

import React, { useState, useEffect } from 'react';

const url = 'https://api.github.com/users/QuincyLarson';

const MultipleReturns = () => {

  const [loading,setLoading] = useState(true)

  if(loading){

    return(

          <h2>Loading...</h2>

    )

  }

 return <h2>multiple returns</h2>;

};

export default MultipleReturns;

Multiple Returns with fetch data

Vamos manter os mesmos arquivos do multiple returns e iremos criar mais um useState const [isError, setIsError] = useState(false), e mais um useState para o usuário const [user, setUser] = useState (‘default user’). Agora na condição do if no return você pode adicionar mais coisas dentro por exemplo uma <div></div>, como temos um erro vamos criar uma condição para o erro if(isError) {return (<div><h1>Error...</h1></div>)} e também temos o caso default que é quando não tem erro e quando não está carregando no caso mostrando o usuário, em multiple returns apague e adicione <div><h1>{user}</h1></div>. Tudo certo agora vamos setar o fetch dentro do useEffect que irá controlar as condições useEffect (() => {}, []) dentro da call-back do useEffect adicione o fetch para a url fetch(url) como o fetch está retornando uma promise temos que usar o .then() e precisamos de um parâmetro para retornar uma resposta em json e precisamos d outro then para o usuário e um catch para o erro.

fetch(url)

.then((resp) => resp.json())

.then((user) => {

const {login} = user

setUser(login)

setIsLoading(false)

})

.catch((error) => console.log(error))

import React, { useState, useEffect } from 'react';

const url = 'https://api.github.com/users/QuincyLarson';

const MultipleReturns = () => {

  const [isLoading,setIsLoading] = useState(true)

  const [isError,setIsError] = useState(false)

  const [user,setUser] = useState('default user')

  useEffect(()=>{

    fetch(url)

      .then((*res*)=> {

        if(*res*.status >= 200 && *res*.status <=299){

            return *res*.json()

        }else{

            setIsLoading(false)

            setIsError(true)

            throw **new** *Error*(*res*.statusText)

        }

      })

      .then((*user*)=>{

        const {login} = *user*;

          setUser(login)

          setIsLoading(false)

      })

      .catch((*error*)=> console.log(*error*))

  },[])

  if(isLoading){

    return(

      <div>

          <h1>Loading...</h1>

      </div>

    )

  }

  if(isError){

    return(

      <div>

          <h1>Error...</h1>

      </div>

    )

  }

  return(

    <div>

        <h1>{user}</h1>

    </div>

  )

};

export default MultipleReturns;

Short Circuit/Ternary Operator

import React, { useState } from 'react';

// short-circuit evaluation

// ternary operator

const ShortCircuit = () => {

  const [text,setText] = useState('')

   //const firstValue = text || 'hello world';

   //const secondValue = text && 'hello world';

   const [isError, setIsError] = useState(false)

  return (

    <>

     {/\* <h1>{firstValue}</h1>

      <h1>Value: {secondValue}</h1>\*/}

        <h1>{text || 'john doe'}</h1>

        <button *className*="btn" *onClick*={()=>setIsError(!isError)}>toogle error</button>

        {/\*{text && <h1>hello world</h1>}

        {!text && <h1>hello world</h1>}\*/}

        {isError && <h1>Error...</h1>}

        {isError ? (

        <p>there is an error...</p>

        ) : (

          <h2>there is no error</h2>

        ) }

    </>

  );

};

export default ShortCircuit;

Show/Hide Component

import React, { useState, useEffect } from 'react';

const ShowHide = () => {

  const [show,setShow] = useState(false)

  return (

    <>

    <button *className*="btn" *onClick*={()=>setShow(!show)}>show/hide</button>

    {show && <*Item* />}

    </>

  );

};

const Item = () => {

    const [size,setSize]= useState(window.innerWidth)

    const checkSize = () => {

      setSize(window.innerWidth)

    }

    useEffect(()=>{

      window.addEventListener('resize',checkSize)

      return ()=>{

        window.removeEventListener('resize',checkSize)

      }

    },[])

    return (

     <div *style*={{marginTop: '2rem'}}>

       <h1>window</h1>

    <h2>size: {size} px</h2>

     </div>

    )

}

export default ShowHide;

FORMS

Os forms ou formulários são uteis para o cadastro de informações vindos do lado do usuário para o servidor e também para mostrar informações. Agora vamos aprender como usar/fazer formulários com o react, primeiro insira o local destinado no App.js .

import React from 'react'

import Setup from './tutorial/4-forms/setup/1-controlled-inputs';

function App() {

  return (

    <div *className*='container'>

      <*Setup*/>

    </div>

  )

}

export default App

No documento 1-controlled-inputs vemos que ele está vazio e a primeira coisa que iremos fazer é montar a estrutura básica de um formulário.

import React, { useState } from 'react';

// JS

// const input = document.getElementById('myText');

// const inputValue = input.value

// React

// value, onChange

const ControlledInputs = () => {

 return <h2>controlled inputs</h2>

};

export default ControlledInputs;

Agora dentro do return vamos criar um <article></article> e dentro desse article criar um <form></form> com uma className ‘form’ em seguida dentro desse form adicionar uma <div></div> com uma className ‘form-control’ dentro desta div iremos criar um <label><label> com um htmlFor = ‘firstName’ e dentro do label ‘Name :’ o htmlFor e o mesmo o que o For dentro da tag label significa pro html. Depois crie um <input></input> do type=’text’ com um id=’firstName’ e um name=’firstName’. Depois copie e cole a <div></div> criada acima e altere os seguintes valores, em label htmlFor mude para ‘email’ e o ‘Nome :’ para Email : no input mude o id=’email’, name=’email’. No final adicione um <button></button> dentro com a frase ‘add person’ com um type=’submit’.

import React, { useState } from 'react';

// JS

// const input = document.getElementById('myText');

// const inputValue = input.value

// React

// value, onChange

const ControlledInputs = () => {

  return (

    <article>

      <form *className*='form'>

        <div *className*='form-control'>

          <label *htmlFor*='firstName'>Name: </label>

          <input

*type*="text"

*id*="firstName"

*name*="firstName"

          />

        </div>

        <div *className*='form-control'>

          <label *htmlFor*='email'>Email: </label>

          <input

*type*="text"

*id*="email"

*name*="email"

          />

        </div>

        <button *type*="submit">add person</button>

      </form>

    </article>

  )

};

export default ControlledInputs;

No caso de formulários no react podemos usar a JSX tag onSubmit ou utilizar do onClick no botão. Como não utilizamos o onSubmit vamos usa-lo agora no form no início adicione o onSubmit={handleSubmit} e cria essa função acima do return, const handleSubmit = (e) => {console.log (‘hello world’)} agora quando adicionamos um nome e o email e clicamos no botão é para no console estar mostrando o ‘hello world’ porém não está porque o browser tentará submeter o formulário e atualizará a página. Em javascript temos a função preventDefault (); que evita de o navegador atualizar a pagina e agora sim vemos o ‘hello world’.

import React, { useState } from 'react';

// JS

// const input = document.getElementById('myText');

// const inputValue = input.value

// React

// value, onChange

const ControlledInputs = () => {

  const [firstName,setFirstName] = useState('')

  const [email, setEmail] = useState('')

  const [listpeople, setListpeople] = useState([])

  const handleSubmit = (*e*) => {

*e*.preventDefault();

    console.log('hello world')

  }

  return (

    <article>

      <form *className*='form' *onSubmit*={handleSubmit}>

        <div *className*='form-control'>

          <label *htmlFor*='firstName'>Name: </label>

          <input

*type*="text"

*id*="firstName"

*name*="firstName"

          />

        </div>

        <div *className*='form-control'>

          <label *htmlFor*='email'>Email: </label>

          <input

*type*="text"

*id*="email"

*name*="email"

          />

        </div>

        <button *type*="submit">add person</button>

      </form>

    </article>

)

};

export default ControlledInputs;

Se passarmos a função handleSubmit para o botão com um onClick ele ainda irá funcionar.

Agora vamos setar os valores useState acima da função handleSubmit, const [firstName, setFirstName] = useState(‘’) e const[email,s etEmail] = useState(‘’) agora dentro das divs no formulário precisamos adicionar mais dois atributos que são o value e o onChange. Agora adiciono nos inputs o value atribuindo o nome do useState value={firstName}, value={email} agora no console você verá que o react irá reclamar, mas é pelo fato de você não ter adicionado o onChange. Agora iremos setar o onChange nos inputs onChange={(e)=> setFirstName(e.target.value)} e onChange={(e)=> setEmail(e.target.value)} agora adicione um console.log(firstName, email) o handleSubmit e repare que agora obtemos os dados do useState no console.

import React, { useState } from 'react';

// JS

// const input = document.getElementById('myText');

// const inputValue = input.value

// React

// value, onChange

const ControlledInputs = () => {

  const [firstName,setFirstName] = useState('')

  const [email, setEmail] = useState('')

  const handleSubmit = (*e*) => {

*e*.preventDefault();

    console.log(firstName,email)

  }

  return (

    <article>

      <form *className*='form' *onSubmit*={handleSubmit}>

        <div *className*='form-control'>

          <label *htmlFor*='firstName'>Name: </label>

          <input

*type*="text"

*id*="firstName"

*name*="firstName"

*value*={firstName}

*onChange*={(*e*) => setFirstName(*e*.target.value)}

          />

        </div>

        <div *className*='form-control'>

          <label *htmlFor*='email'>Email: </label>

          <input

*type*="text"

*id*="email"

*name*="email"

*value*={email}

*onChange*={(*e*) => setEmail(*e*.target.value)}

          />

        </div>

        <button *type*="submit">add person</button>{/\*no form ou onClick \*/}

      </form>

    </article>

  )

};

export default ControlledInputs;

Aprendemos agora como organizar os dados de um formulário de pequenos inputs, agora iremos aprender como colocar os dados em uma pequena lista em um array dinamicamente. Primeiro vamos adicionar um useState para salvar as pessoas na lista const [listpeople, setPeople] = useState([]) quando clicamos em adicionar observamos que o console retorna 2 valores por causa do nome e do email agora precisamos ter certeza que ele retorne um valor verdadeiro e para isso precisamos criar uma condição porque no caso valores com strings vazias são informações falsas. Então dentro da handleSubmit function adicionamos a condição if(firstName && email){console.log(‘submit form’)} else {console.log(‘empty values’)} , agora da condição verdadeira vamos criar um objeto para os dados que serão inseridos const person= {firstName, email} agora para retornar os valores precisamos chamar a função setPeople((people)=>[...people, person]) e setar os valores para empty string após submeter o form setFirstName(‘’) e setEmail(‘’) agora que tal mostrar as pessoas que adicionamos na lista na página, em baixo do formulário {listpeople.map((person)=>{const {id, firstName, email} = person) return (<div className=’item’ key={id}><h4>{firstName}</h4><p>{email}</p></div>)} agora vamos usar um cheat para não dar um erro de keyprop no objeto person adicione id: new Date().getTime().ToString().

import React, { useState } from 'react';

// JS

// const input = document.getElementById('myText');

// const inputValue = input.value

// React

// value, onChange

const ControlledInputs = () => {

  const [firstName,setFirstName] = useState('')

  const [email, setEmail] = useState('')

  const [listpeople, setListpeople] = useState([])

  const handleSubmit = (*e*) => {

*e*.preventDefault();

    //console.log(firstName,email)

    if(firstName && email){

      console.log('submit the form')

      const person = {

        //firstName,

        //email

        //Both work if the variable of the useState matches

        firstName :firstName,

        email: email,

        id: **new** *Date*().getTime().toString()

      }

      setListpeople((*people*)=>[...*people*,person])

      setFirstName('')

      setEmail('')

    }else {

      console.log('empty values')

    }

  }

  return (

    <article>

      <form *className*='form' *onSubmit*={handleSubmit}>

        <div *className*='form-control'>

          <label *htmlFor*='firstName'>Name: </label>

          <input

*type*="text"

*id*="firstName"

*name*="firstName"

*value*={firstName}

*onChange*={(*e*) => setFirstName(*e*.target.value)}

          />

        </div>

        <div *className*='form-control'>

          <label *htmlFor*='email'>Email: </label>

          <input

*type*="text"

*id*="email"

*name*="email"

*value*={email}

*onChange*={(*e*) => setEmail(*e*.target.value)}

          />

        </div>

        <button *type*="submit">add person</button>{/\*no form ou onClick \*/}

      </form>

      {listpeople.map((*person*)=>{

        const {id,firstName,email} = *person*

        return (

          <div *className*='item' *key*={id}>

              <h4>{firstName}</h4>

              <p>{email}</p>

          </div>

        )

      })}

    </article>

  )

};

export default ControlledInputs;

Multiple inputs

O formulário acima funciona para de 1 a 4 inputs agora vamos otimizar isso de uma forma que não interessa a quantidade de inputs que você o adicionar irá atualizar devidamente. Ao invés de criar um useState para cada input podemos criar um input universal colocando um objeto dentro do useState. Aqui colocamos os elementos firstName, email, age para informar a pessoa dentro do useState. No exemplo anterior utilizamos somente o handleSubmit para confirmar os valores informados nesse exemplo separamos em duas funções a handleSubmit e a handleChange. Primeiro mudaremos os values dos inputs que estavam referenciadas no exemplo anterior por exemplo ao invés de value={firstName} altere para value={person. firstName} mude para os outros respectivamente. Agora percebemos que o onChange está referenciando as functions antigas então iremos criar uma function handleChange para referenciar todas. Para mudar cada valor respectivamente precisamos no handleChange pegar os valores únicos dos inputs que são o nome e o value. Agora dentro do handleChange chamamos a function setPerson e nela precisamos chamar os valores dinamicamente para isso usamos primeiro o spread operator para person {...person} informa todos valores dentro de person e informamos dinamicamente os valores que devem ser iguais [name]:value . No handleSubmit iremos criar uma condição caso os nomes sejam inválidos primeiros vamos criar uma variável newPerson e desestruturar a person e criar um id. E também chamamos setPeople

salvamos people e newPerson em array [...people,newPerson] e também chamaos setPerson para quando enviarmos o formulário ele retorna os valores vazios novamente {firstName:’’, email:’’, age:’’}.

import React, { useState } from 'react';

// JS

// const input = document.getElementById('myText');

// const inputValue = input.value

// React

// value, onChange

// dynamic object keys

const ControlledInputs = () => {

  //const [firstName, setFirstName] = useState('');

  //const [email, setEmail] = useState('');

  //const [age,setAge] = useState('');

  const [person,setPerson] = useState({firstName: '',email: '', age: ''})

  const [people, setPeople] = useState([]);

  const handleChange = (*e*)=>{

    const name = *e*.target.name;

    const value = *e*.target.value;

    setPerson({...person, [name]:value })

  }

  const handleSubmit = (*e*)=>{

*e*.preventDefault();

    if({...person}/\*or person.firstName && person.email && person.age\*/){

      const newPerson = {...person,id: **new** *Date*().getTime().toString()}

      setPeople([...people,newPerson])

      setPerson({firstName: '',email: '',age: ''})

    }

  }

  return (

    <>

      <article>

        <form *className*='form' /\*onSubmit={handleSubmit}\*/ >

          <div *className*='form-control'>

            <label *htmlFor*='firstName'>Name : </label>

            <input

*type*='text'

*id*='firstName'

*name*='firstName'

*value*={person.firstName}

              //onChange={(e) => setFirstName(e.target.value)}

*onChange*={handleChange}

            />

          </div>

          <div *className*='form-control'>

            <label *htmlFor*='email'>Email : </label>

            <input

*type*='email'

*id*='email'

*name*='email'

*value*={person.email}

              //onChange={(e) => setEmail(e.target.value)}

*onChange*={handleChange}

            />

          </div>

          <div *className*='form-control'>

            <label *htmlFor*='age'>Age : </label>

            <input

*type*='text'

*id*='age'

*name*='age'

*value*={person.age}

              //onChange={(e) => setAge(e.target.value)}

*onChange*={handleChange}

            />

          </div>

          <button *type*='submit' *onClick*={handleSubmit}>add person</button>

        </form>

        {people.map((*person*, *index*) => {

          const { id, firstName, email, age } = *person*;

          return (

            <div *className*='item' *key*={id}>

              <h4>{firstName}</h4>

              <p>{email}</p>

              <p>{age}</p>

            </div>

          );

        })}

      </article>

    </>

  );

};

export default ControlledInputs;

Use Ref Basics

O useRef é bem similar ao useState, porém diferente do useState ele não aplica o re-renderização da página e também seleciona DOM node/elements. Primeiro iremos criar um formulário com a className ‘form’ e dentro iremos criar um input do type ‘text’ e um button de type ’submit’ com uma mensagem dentro. Em seguida adicione o onSubmit no formulário adicionando a function handleSubmit. Agora crie a function handleSubmit e faça o mesmo procedimento usado no forms. Do jeito que o useRef funciona temos que criar uma variável refContainer e vamos setar como padrão o useRef(null) e para mostrar o ref precisamos criar um atributo no input e chamar a variável dentro dele. Agora de um console.log() antes do return no refContainer e repare que um ref é um objeto de propriedade current de valor null. Agora dentro do handleSubmit de um console.log () em refContainer.current.value agora quando você digitar algo no input e apertar o botão ele irá mostrar o valor digitado, você não é limitado a usar um useRef por html por exemplo crie depois do for crie uma div com ref divContainer e set o useRef como null, dentro da div digite um Hello World e em seguida de um console.log () no handleSubmit para divContainer,current e depois que você digitar um valor no input e apertar o botão de submit noticie que o divContainer no console é apontado para a div. Crie um useEffect e de um console.log () no refContainer.current e depois de um refContainer.current.focus() que no console quando você passar por cima ele irá apontar a ref.

import React, { useEffect, useRef } from 'react';

// preserves value

// DOES NOT trigger re-render

// target DOM nodes/elements

const UseRefBasics = () => {

  const refContainer = useRef(null)

  const divContainer = useRef(null)

  const handleSubmit = (*e*) =>{

*e*.preventDefault()

    console.log(refContainer.current.value)

    console.log(divContainer.current)

  }

  useEffect(()=>{

    console.log(refContainer.current)

    refContainer.current.focus()

  })

  return (

    <>

      <form *className*='form' *onSubmit*={handleSubmit}>

        <div>

          <input *type*='text' *ref*={refContainer}/>

          <button *type*='submit'>submit</button>

        </div>

      </form>

      <div *ref*={divContainer}>Hello World</div>

    </>

  );

};

export default UseRefBasics;

UseReducer Basics

O useReducer é usado quando temos um programa mais complicado, mas ele definitivamente pode ser usado para exemplos simples também. Neste exemplo iremos usar os dados do primeiro exemplo do arquivo data.js um componente externo modal, e neste arquivo observe que não usamos o App.js nele usamos o index.js em alguns projetos futuros você pode se deparar com isso. Primeiro vamos criar no index.js os useState os das pessoas que irão pegar os dados de data.js const [people, setPeople] = useState(data), um para mostrar o modal const [showModal, setShowModal] = useState(false) e um para o nome que iremos digitar e salvar na lista const [name, setName] = useState (‘’), dentro do return vamos criar um fragment e nele vamos criar uma condição para o modal se for verdadeiro mostrar o modal {showModal && <Modal/>} agora vamos criar um form com um onSubmit={handleSubmit} uma div e um input com type ‘text’ e com um valor igual ao name do useState value={name} e um onChange={(e)=> setName(e.target,value)} depois crie a function handleSubmit com um preventDefault() const handleSubmit = (e) => {e.preventDefault()} depois vamos adicionar um pouco de estilo em form adicione o className =’form’ e abaixo da primeira div crie uma segunda div com um botão de type=’submit’ com a frase add <button type=’submit’>add</button> agora precisamos checar se o nome é uma string vazia então abaixo do e.preventDefault() e se for uma string vazia queremos que mostre o modelo então criamos uma condição if(name){setShowModal(true)}else{setShowModal(true)} queremos adicionar as pessoas digitadas no formulário a lista então precisamos usar o setPeople para salvar na lista e resetar o valor escrito no form if(name){setShowModal(true) setPeople([...people, {id: new Date().getTime().toString(), name} setName(‘’)])}else{setShowModal(true)} e também vamos iterar para mostrar a lista abaixo do form use .map() para ler a lista {people.map((person)=>{return(<div key={person.id}><h4>{person.name}</h4></div>)})} e esse seria o cenário usando o useState.

import React, { useState, useReducer } from 'react';

import Modal from './Modal';

import { data } from '../../../data';

// reducer function

const Index = () => {

  const [name,setName] = useState('')

  const [people,setPeople] = useState(data)

  const [showModal, setShowModal] = useState(false)

  const handleSubmit = (*e*) =>{

*e*.preventDefault()

    if(name){

      setShowModal(true)

      setPeople([...people,{id:**new** *Date*().getTime().toString(),name}])

      setName('')

    }else{

      setShowModal(true)

    }

  }

  return (

    <>

      {showModal && <*Modal* />}

      <form *onSubmit*={handleSubmit} *className*='form'>

        <div>

            <input

*type*='text'

*value*={name}

*onChange*={(*e*) => setName(*e*.target.value)}

            />

        </div>

        <button *type*='submit'>add</button>

      </form>

      {people.map((*person*)=>{

        return (

          <div *key*={*person*.id}>

            <h4>{*person*.name}</h4>

          </div>

        )

      })}

    </>

  );

};

export default Index;

UseReducer – Refractor

Primeiro vamos remover os useState de people e model e iremos criar uma nova variável const [state (pode ser qualquer nome), dispatch] = useReducer(reducer) e acima do index iremos criar a função para o reducer e irá procurar por duas coisas state e action const reducer = (state, action) => {} e ainda dentro do useReducer precisamos setar o state você pode colocar ele dentro do useReducer ou criar uma variável por fora e chamar ela, nesse exemplo iremos criar uma variável por fora abaixo do reducer crie const defaultState = {people: [], isModelOpen: false, modalContent: ‘’} e no useReducer passamos agora o defaultState const [state (pode ser qualquer nome), dispatch] = useReducer(reducer, defaultState) por múltiplos motivos o aplicativo irá quebrar como vemos no exemplo anterior não temos as functions do useState antigo precisa, agora iremos excluir as três functions do if (name) e else e vamos deixar em branco por enquanto, e dentro do return iremos trocar a função da chave inicial por state chamando a propriedade isModalOpen que irá mostrar o modal, e onde temos people logo abaixo trocamos por state.people agora iremos setar algumas coisas no Modal componente nele iremos colocar o atributo modalContent={state.modalContent} agora iremos no arquivo modal e iremos adicionar o parâmetro no parênteses de Modal modalContent e em seguida adicionar no return uma div com um className=’modal’ e dentro da div adicionar {modalContent}.

Index.js

import React, { useState, useReducer } from 'react';

import Modal from './Modal';

import { data } from '../../../data';

// reducer function

const reducer = (*state*,*action*) => {}

const defaultState = {

  people: [],

  isModalOpen: false,

  modalContent: ''

}

const Index = () => {

  const [name,setName] = useState('')

  const [state, dispatch] = useReducer(reducer,defaultState)

  const handleSubmit = (*e*) =>{

*e*.preventDefault()

    if(name){

    }else{

    }

  }

  return (

    <>

      {state.isModalOpen && <*Modal* *modalContent*={state.modalContent} />}

      <form *onSubmit*={handleSubmit} *className*='form'>

        <div>

            <input

*type*='text'

*value*={name}

*onChange*={(*e*) => setName(*e*.target.value)}

            />

        </div>

        <button *type*='submit'>add</button>

      </form>

      {state.people.map((*person*)=>{

        return (

          <div *key*={*person*.id}>

            <h4>{*person*.name}</h4>

          </div>

        )

      })}

    </>

  );

};

export default Index;

Modal.js

import React, { useEffect } from 'react';

const Modal = ({*modalContent*}) => {

  return(

    <div *className*='modal'>

      <p>{*modalContent*}</p>

    </div>

  );

};

export default Modal;

UseReducer - Add items

Primeiro precisamos chamar a function dispatch () que serve para despachar a ação no reducer, em if (name) adicione a function, a function dispatch recebe um objeto do essencial e pode colocar objetos adicionais nele, o objeto essencial é a propriedade type: ‘ADD\_VALUE’ pode ser qualquer nome porem neste exemplo é pra mostrar quando o objeto está adicionando um valor dispatch ({type: ‘ADD\_VALUE’}). Dentro do reducer precisamos sempre retornar um valor pois se não pode ocorrer um erro coloque um return state e depois de um console.log(state)

para observar o que acontece nele você verá seu old state e o dispatch. Agora vamos checar se os itens estão sendo adicionados com uma condição if (action.type === ‘ADD\_VALUE’) {} a action refere-se ao dispatch e o type ao valor. Precisamos retornar um objeto dentro do if assim o return state que fizemos primeiro irá se tornar o default dentro do return do if precisamos retornar um objeto e dentro o objeto precisamos pegar os valores de state, para fazer isso é usar a desestruturação pegando todos os arquivos ...state, e em seguida informar os valores do objeto do defaut state return {...state, people: data, isModalOpen: true, modalContent: ‘item added!’}. Como temos agora um state dentro do if e um fora não podemos usar os dois uma solução básica para isso é colocar um throw newError (‘no matching action type!’), agora no else do if (name) insira um dispatch com um objeto do type:’NO\_VALUE’ dispatch ({type: ‘NO\_VALUE’}). Precisamos passar os dados do antigo array para o novo por isso precisamos criar um novo item dentro do if (name) crie uma variável newItem e passe um objeto para pegar o id e o novo nome adicionado const newItem = {id: new Date().getTime().toString(), name} agora dentro do dispatch passe uma nova propriedade pode ser de qualquer nome e chame o newItem dispatch ({type: ‘ADD\_VALUE’, payload: newItem}). Agora na condição de adicionar no reducer precisamos criar uma nova variável para pessoa e em seguida chamar o valor do state inicial usando o spread operator e o action const newPeople = [...state.people, action.payload] e dentro do return no objeto people altere o valor de people: data para people: newPeople. Para mostrar que não adicionamos nada quando não colocamos nada no input só falta criar a condição dentro do reducer if(action.type === ‘NO\_VALUE’){ return({...state, isModalOpen: true, modalContent: ‘please enter value.’})}

import React, { useState, useReducer } from 'react';

import Modal from './Modal';

import { data } from '../../../data';

// reducer function

const reducer = (*state*,*action*) => {

    console.log(*state*)

  if(*action*.type === 'ADD\_ITEM'){

    const newPeople = [...*state*.people,*action*.payload]

    return(

      {

        ...*state*,

        people: *newPeople*,

        isModalOpen: *true*,

        modalContent: 'item added!'

      }

    )

  }

  if(*action*.type === 'NO\_VALUE'){

      return (

        {

          ...*state*,

          isModalOpen: *true*,

          modalContent:'please enter value.'

        }

      )

  }

  throw **new** *Error*('no matching action type');

}

const defaultState = {

  people: [],

  isModalOpen: false,

  modalContent: ''

}

const Index = () => {

  const [name,setName] = useState('')

  const [state, dispatch] = useReducer(reducer,defaultState)

  const handleSubmit = (*e*) =>{

*e*.preventDefault()

    if(name){

     const newItem = {

       id: **new** *Date*().getTime().toString(),

       name

    }

      dispatch({type: 'ADD\_ITEM', payload: newItem})

      setName('')

    }else{

     dispatch({type: 'NO\_VALUE'})

    }

  }

  return (

    <>

      {state.isModalOpen && <*Modal* *modalContent*={state.modalContent} />}

      <form *onSubmit*={handleSubmit} *className*='form'>

        <div>

            <input

*type*='text'

*value*={name}

*onChange*={(*e*) => setName(*e*.target.value)}

            />

        </div>

        <button *type*='submit'>add</button>

      </form>

      {state.people.map((*person*)=>{

        return (

          <div *key*={*person*.id}>

            <h4>{*person*.name}</h4>

          </div>

        )

      })}

    </>

  );

};

export default Index;

UseReducer: Remove Items

Primeiro vamos fazer uma função para fechar o modal e um botão com uma função para remover o item, agora vá abaixo da function handleSubmit e cria uma nova function com o nome de closeModal e dentro dessa function iremos usar o dispatch com o type:’CLOSE\_MODAL’, const closeModal = () => {dispatch({type:’CLOSE\_MODAL’})}, agora no reducer crie uma condição iguais as anteriores if retornando os mesmos valores do state e mudando apenas o isModalOpen para false, if(action.type === ‘CLOSE\_MODAL’){return({...state, isModalOpen: false})}, o macete aqui é que queremos chamar em forma de prop no componente modelo então iremos adicionar no return com um fragment no componente, closeModal={closeModal}, iremos para o arquivo Modal.js e adicionaremos ele no parênteses onde está modalContent e no model iremos usar um useEffect para setar um display de mensagem de 3segundos, useEffect(()=>{setTimeout(()=>{closeModal()}, 3000)}), agora quando digitamos e apertamos o botão observe que o modelo aparece e 3s depois ele some, logo abaixo do h4 vamos criar um botão com um onClick e dentro dele uma arrow function para um dispatch de type: ’REMOVE\_ITEM’ e também precisamos adicionar o payload de novo porque precisamos ser especifico para qual item remover e passamos person.id, <button onClick={()=>dispatch({type: ’REMOVE\_ITEM’, payload: person.id})} >remove</button>, agora ao clicar em remove após adicionar um nove vemos um grande erro pois não criamos no reducer uma ação para remover o item, para isso vamos ao reducer de novo e adicionamos a condição novamente if(action.type === ’REMOVE\_ITEM’){const newPeople = state.people.filter((person)=>person.id !== action.payload) return({...state, people: newPeople, isModalOpen: true, modalContent: ‘Item removed!’})}, agora para deixar o projeto mais organizado criamos um arquivo reducer.js e passamos para ele o reducer e no início adicionamos o export.

Reducer.js

export const reducer = (*state*,*action*) => {

    console.log(*state*)

  if(*action*.type === 'ADD\_ITEM'){

    const newPeople = [...*state*.people,*action*.payload]

    return(

      {

        ...*state*,

        people: *newPeople*,

        isModalOpen: *true*,

        modalContent: 'item added!'

      }

    )

  }

  if(*action*.type === 'NO\_VALUE'){

      return (

        {

          ...*state*,

          isModalOpen: *true*,

          modalContent:'please enter value.'

        }

      )

  }

  if(*action*.type === 'CLOSE\_MODAL'){

    return (

        {

         ...*state*,

         isModalOpen: *false*

        }

    )

  }

  if(*action*.type === 'REMOVE\_ITEM'){

    const newPeople = *state*.people.filter((*person*)=>*person*.id !== *action*.payload)

    return (

      {

        ...*state*,

        people: *newPeople*,

        isModalOpen: *true*,

        modalContent: 'Item removed!'

      }

    )

  }

  throw **new** *Error*('no matching action type');

}

Agora dentro do index precisamos importar o arquivo reducer.js import {reducer} from ‘./reducer’

Index.js

import React, { useState, useReducer } from 'react';

import Modal from './Modal';

import { data } from '../../../data';

// reducer function

import {reducer} from './reducer'

const defaultState = {

  people: [],

  isModalOpen: false,

  modalContent: ''

}

const Index = () => {

  const [name,setName] = useState('')

  const [state, dispatch] = useReducer(reducer,defaultState)

  const handleSubmit = (*e*) =>{

*e*.preventDefault()

    if(name){

     const newItem = {

       id: **new** *Date*().getTime().toString(),

       name

    }

      dispatch({type: 'ADD\_ITEM', payload: newItem})

      setName('')

    }else{

     dispatch({type: 'NO\_VALUE'})

    }

  }

  const closeModal = () => {

    dispatch({type: 'CLOSE\_MODAL'})

  }

  return (

    <>

      {state.isModalOpen && <*Modal* *closeModal*={closeModal} *modalContent*={state.modalContent} />}

      <form *onSubmit*={handleSubmit} *className*='form'>

        <div>

            <input

*type*='text'

*value*={name}

*onChange*={(*e*) => setName(*e*.target.value)}

            />

        </div>

        <button *type*='submit'>add</button>

      </form>

      {state.people.map((*person*)=>{

        return (

          <div *key*={*person*.id} *className*='item'>

            <h4>{*person*.name}</h4>

            <button *onClick*={()=>dispatch({type:'REMOVE\_ITEM', payload:*person*.id})}>

              remove

            </button>

          </div>

        )

      })}

    </>

  );

};

export default Index;

Modal.js

import React, { useEffect } from 'react';

const Modal = ({*modalContent*,*closeModal*}) => {

  useEffect(()=>{

    setTimeout(()=>{

      closeModal()

    }, 3000)

  })

  return(

    <div *className*='modal'>

      <p>{*modalContent*}</p>

    </div>

  );

};

export default Modal;

Context API/UseContext

Primeiro mude o arquivo do app.js para e reconhecer o arquivo 1-context-api, vemos no arquivo que ele está usando prop drilling para passar uma função para todos os componentes e não ocorrer um erro vamos aprender o useContext para não usar a prop drilling.

import React, { useState, useContext } from 'react';

import { data } from '../../../data';

// more components

// fix - context api, redux (for more complex cases)

const ContextAPI = () => {

  const [people, setPeople] = useState(data);

  const removePerson = (*id*) => {

    setPeople((*people*) => {

      return *people*.filter((*person*) => *person*.id !== *id*);

    });

  };

  return (

    <>

      <h3>prop drilling</h3>

      <*List* *people*={people} *removePerson*={removePerson} />

    </>

  );

};

const List = ({ *people*, *removePerson* }) => {

  return (

    <>

      {*people*.map((*person*) => {

        return (

          <*SinglePerson*

*key*={*person*.id}

            {...person}

*removePerson*={*removePerson*}

          />

        );

      })}

    </>

  );

};

const SinglePerson = ({ *id*, *name*, *removePerson* }) => {

  return (

    <div *className*='item'>

      <h4>{*name*}</h4>

      <button *onClick*={() => removePerson(*id*)}>remove</button>

    </div>

  );

};

export default ContextAPI;

Primeiro crie uma variável para salvar o useContext const PersonContext = React.createContext(), no momento que fazemos isso temos acesso a dois componentes o Provider e o Consumer, o provider funciona como um distribuidor que sai do componente principal e vai para o restante para isso no componente principal no return você deve passar a variável com o componente provedor, no react fragment que é o principal do aplicativo altere para o nome da variavel.provider, <PersonContext.Provider><h3>Prop Drilling</h3><List people={people} removePerson={removePerson}/></PersonContext.Provider> e com o privider podemos passar um value vamos começar simples com um ‘hello’ <PersonContext.Provider value=‘hello’> agora vamos em SinglePerson e iremos criar uma variável para salvar o contexto iremos muda-la mais para frente, const data = useContext(PersonContext) agora console.log(data) e repare que no console ele irá lhe mostrar a quantidade de dados em forma de ‘hello’, perceba-se que estamos usando JSX então podemos passar um objeto em forma de javascript para o valor em value no provider mude para {{removePerson}} depois vai em data e desconstrua a variável const {removePerson} = useContext(PersonContext) e agora note que irá ocorrer um erro porque o nome das instancias são iguais então retire os removePerson de List e ContextAPI, você também pode passar o people de List para o useContext basta adicionar no provider <PersonContext.Provider><h3>Prop Drilling</h3><List removePerson={removePerson, people}/></PersonContext.Provider> retirar do componente List {people} e criar uma variável chamando os dados com o useContext const DadosLista = useContext(PersonContext) e dentro do fragmente adicionar DadosLista antes de people.

import React, { useState, useContext } from 'react';

import { data } from '../../../data';

// more components

// fix - context api, redux (for more complex cases)

const PersonContext = *React*.createContext();

//dois componentes - Provider - Consumer

const ContextAPI = () => {

  const [people, setPeople] = useState(data);

  const removePerson = (*id*) => {

    setPeople((*people*) => {

      return *people*.filter((*person*) => *person*.id !== *id*);

    });

  };

  return (

    <*PersonContext.Provider* *value*={{removePerson, people}}>

      <h3>Context API / useConntext</h3>

      <*List*/>

    </*PersonContext.Provider*>

  );

};

const List = () => {

  const DadosLista = useContext(PersonContext)

  return (

    <>

      {DadosLista.people.map((*person*) => {

        return (

          <*SinglePerson*

*key*={*person*.id}

            {...person}

          />

        );

      })}

    </>

  );

};

const SinglePerson = ({ *id*, *name*}) => {

  const {removePerson} = useContext(PersonContext)

  return (

    <div *className*='item'>

      <h4>{*name*}</h4>

      <button *onClick*={() => removePerson(*id*)}>remove</button>

    </div>

  );

};

export default ContextAPI;

Custom Hooks – useFetch

Custom Hooks permitem que podemos reusar funções do javascript para diferentes componentes dentro do react o exemplo proposto serve para entender como funciona um custom hooks e poder usa-lo para criar seus próprios ou usar outros de outras pessoas.

No exemplo que usamos temos esse código:

import React, { useState, useEffect } from 'react'

import { useFetch } from './2-useFetch'

// ATTENTION!!!!!!!!!!

// I SWITCHED TO PERMANENT DOMAIN

const url = 'https://course-api.com/javascript-store-products'

const Example = () => {

  const [loading, setLoading] = useState(true)

    const [products, setProducts] = useState([])

    const getProducts = async () => {

      const response = await fetch(url)

      const products = await response.json()

      setProducts(products)

      setLoading(false)

    }

    useEffect(() => {

      getProducts()

    }, [url])

  console.log(products)

  return (

    <div>

      <h2>{loading ? 'loading...' : 'data'}</h2>

    </div>

  )

}

export default Example

Neste código temos dois useState um para o carregamento do pagina e um para os produtos e uma função assíncrona para buscar os dados da url e informa-los em array. Agora iremos para o outro arquivo da pasta ele deve estar assim:

import { useState, useEffect } from 'react';

export const useFetch = () => {

};

Agora no primeiro arquivo copie os códigos de const [loading, setLoading] = useState(true) até o final do useEffect e cole os arquivos no sua useFetch function agora dentro do custom Hook precisamos retornar alguma coisa para o arquivo principal, nele precisamos retornar um objeto inserindo os dois valores state return {loading, products} também precisamos passar um argumento para a função e chamar o useEffect quando o argumento mudar.

Custom Hook arquivo:

import { useState, useEffect } from 'react';

export const useFetch = (*url*) => {

    const [loading, setLoading] = useState(true)

    const [products, setProducts] = useState([])

    const getProducts = async () => {

      const response = await fetch(*url*)

      const products = await response.json()

      setProducts(products)

      setLoading(false)

    }

    useEffect(() => {

      getProducts()

    }, [*url*])

    return {loading,products}

};

Agora no arquivo principal precisamos importar o custom Hook import {useFetch} from ‘. /2-useFetch’ agora precisamos invocar o nosso custom Hook então precisamos desestruturar o objeto retornado do custom Hook então adicione const {loading, products} = useFetch(url) uma observação é sempre necessária na hora da criação de um custom Hook na function adicionar o use na frente do nome pois o react não lê como um Hook sem o use.

Arquivo principal:

import React, { useState, useEffect } from 'react'

import { useFetch } from './2-useFetch'

// ATTENTION!!!!!!!!!!

// I SWITCHED TO PERMANENT DOMAIN

const url = 'https://course-api.com/javascript-store-products'

const Example = () => {

  const {loading,products} = useFetch(url)

  console.log(products)

  return (

    <div>

      <h2>{loading ? 'loading...' : 'data'}</h2>

    </div>

  )

}

export default Example

Prop-Types – Setup

Prop-types permitem que validamos propriedades que passamos para o componente. Agora importe para o App.js import Setup from './tutorial/10-prop-types/setup'; e em seguida abra o arquivo index.js na pasta setup repare os seguintes imports nele importamos os Produtos onde iremos trabalhar na maior parte do tempo, o useFetch do exercício anterior e também criamos uma variável para salvar a api do useFetch.

Index.js

import React from 'react'

import Product from './Product'

import { useFetch } from '../../9-custom-hooks/final/2-useFetch'

// ATTENTION!!!!!!!!!!

// I SWITCHED TO PERMANENT DOMAIN

const url = 'https://course-api.com/react-prop-types-example'

const Index = () => {

  const { products } = useFetch(url)

  return (

    <div>

      <h2>products</h2>

      <section *className*='products'>

        {products.map((*product*) => {

          return <*Product* *key*={*product*.id} {...product} />

        })}

      </section>

    </div>

  )

}

export default Index

Agora vá no sidebar em Products.js e observe que nele não retornamos nada além de um artigo com uma mensagem de single product. Precisamos acessar a API para mostrar os itens então dentro do parêntesis da arrow function informamos os parâmetros da api que são nome, price e image. Em seguida adicionamos um <h4>{name}</h4> isso irá mostrar todos os nomes da api em seguida iremos adicionar o preço <p>${price}</p> repare que só ultimo não mostrou um valor pois na api ele não possui a propriedade price mas repare que em javascript quando acessamos um objeto que no caso é de image e temos conteúdo que retorna para ela no final do projeto no ultimo container não temos a propriedade image isso irá gerar um grande erro na aplicação.

import React from 'react';

const Product = ({*image*,*name*,*price*}) => {

  return <article *className*='product'>

   <img *src*={*image*.url} *alt*={*name*}/>

    <h4>{*name*}</h4>

    <p>${*price*}</p>

  </article>;

};

export default Product;

Agora caso uma situação dessa ocorra novamente em um futuro não se preocupe, primeiro coloque as tags em comentários e em seguida cheque os valores com um console.log () observe que alguns valores no console mostram que estão indefinidos e para isso precisamos criar valores default para eles.

import React from 'react';

const Product = ({*image*,*name*,*price*}) => {

  console.log(*image*,*name*,*price*)

  return <article *className*='product'>

    <h4>single product</h4>

   {/\* <img src={image.url} alt={name}/>

    <h4>{name}</h4>

    <p>${price}</p>\*/}

  </article>;

};

export default Product;

Prop-Types Image

Primeiro para resolver este pequeno erro temos que importar do próprio react as propTypes então import PropTypes from ‘prop-types’ este é pacote grátis vem junto quando você importa o react no arquivo. Depois precisamos criar essas propriedades em forma de objeto e informar chame a variável e em seguida chame a propriedade propTypes Product.propTypes = {} dentro do objeto use as propriedades que foram usadas na api Product.propTypes = {image:PropTypes.object.isRequired,name:PropTypes.string.isRequired,price:PropTypes.number.isRequired} repare que o que muda de cada propriedade e o tipo da variável, em seguida precisamos setar uma propriedade inicial então chame novamente Product e crie um nome da variável inicial no caso defaultProps e sete para um objeto Product.defaultProps = {} agora dentro do objeto e só colocar os valores para as variáveis por exemplo em name vamos colocar como ‘default name’ porem vamos precisar importar a imagem default para isso criamos um import defaultImage from ‘origem da pasta/nome completo + tipo do dado’ agora tire os comentários antes colocados no jsx e observe que tudo mostrou corretamente exceto a ultima imagem pois temos que mudar para short terms operators.

import React from 'react';

import PropTypes from 'prop-types'

import defaultImage from '../../../assets/default-image.jpeg'

const Product = ({*image*,*name*,*price*}) => {

  console.log(*image*,*name*,*price*)

  return <article *className*='product'>

    <img *src*={*image*.url} *alt*={*name*}/>

    <h4>{*name*}</h4>

    <p>${*price*}</p>

  </article>;

};

Product.propTypes = {

  image: PropTypes.object.isRequired,

  name: PropTypes.string.isRequired,

  price: PropTypes.number.isRequired

}

Product.defaultProps = {

  name: 'default name',

  price: 3.99,

  image:  defaultImage

}

export default Product;

Prop-Types Default Value Short Term Operator

Para fazer isso com short terms precisamos deixar em comentários Product.defaultProps e em seguida criar uma variável para checar a imagem no caso const url = image && image.url assim a imagem será checada podemos usar isso em name e price também.

import React from 'react';

import PropTypes from 'prop-types'

import defaultImage from '../../../assets/default-image.jpeg'

const Product = ({*image*,*name*,*price*}) => {

 const url = *image* && *image*.url

 return <article *className*='product'>

    <img *src*={url || defaultImage} *alt*={*name* || 'default name'}/>

    <h4>{*name*}</h4>

    <p>${*price* || 3.99}</p>

  </article>;

};

Product.propTypes = {

  image: PropTypes.object.isRequired,

  name: PropTypes.string.isRequired,

  price: PropTypes.number.isRequired

}

//Product.defaultProps = {

//

//name: 'default name',

//price: 3.99,

//image:  defaultImage

//

//}

export default Product;

React Router

Primeiro em todo o projeto que você for fazer com o react caso precisar fazer vários componentes e poder migrar entre eles usando o url e na parte do front-end precisamos usar o react-router ele permite fazer isso de uma forma mais fácil. Para olhar a documentação acesse: <https://reactrouter.com/web/guides/quick-start>. Para o projeto precisamos instalar o router para isso insira o comando. npm react-router-dom. Agora no precisamos importar os valores no index para chamar o pacote import {BrowserRouter as Router, Route, Switch} from 'react-router-dom' neles terão três alias o Router, o Route e o Switch. Precisamos passar o Router para o setup no index então no return mude para <Router>react router</Router> normalmente quando trabalhamos com react router nos cobre todo o projeto com o router no caso do App.js ou o index.js, mas como temos um index nesse projeto não precisamos encobri-lo. Para setar os componentes em forma de páginas precisamos setar a rota então usamos a propriedade path para indicar o caminho da rota coloque dentro do Router <Route exact path="/"><Home /></Route> normalmente a ‘/’ serve como caminho para a página inicial agora repita esse Route para os 2 componentes restantes About e People mudando apenas o path. Brinque acessando as rotas no navegador e repare que logo após colocar uma rota e adicionar outra irá mostrar as duas rotas, mas para isso basta usar na rota principal a propriedade exact o exact não é limitado você pode usar ele mais de uma vez, agora na rota principal no caso Home adicione o exact <Route exact path="/"><Home /></Route>.

import React from 'react';

// react router

import { *BrowserRouter* as Router, Route, Switch } from 'react-router-dom';

// pages

import Home from './Home';

import About from './About';

import People from './People';

import Error from './Error';

import Person from './Person';

// navbar

import Navbar from './Navbar';

const ReactRouterSetup = () => {

  return (

    <*Router*>

        <*Route* *exact* *path*="/">

            <*Home* />

        </*Route*>

        <*Route* *path*="/about">

            <*About* />

        </*Route*>

        <*Route* *path*="/people">

            <*People* />

        </*Route*>

    </*Router*>

  );

};

export default ReactRouterSetup;

React Router Error and Switch

Agora caso alguém ir na url na barra superior e digitar uma rota que não existe por exemplo /hello sua página por enquanto irá mostrar uma página vazia, será uma melhor pratica fazermos uma página de erro redirecionando a pessoa para alguma outra página existente. Agora dentro do index crie uma rota para a página de erro, no path insira um \*, <Route path="\*"><Error /> </Route>. Observe que agora quando mudamos a página para uma rota que não existe ela irá retornar para o erro, porém quando voltamos para as outras páginas elas também estarão mostrando a página de erro para resolver esse problema precisamos embrulhar todas as rotas no alias Switch ele que fará o trabalho de deixar a página ser única.

import React from 'react';

// react router

import { *BrowserRouter* as Router, Route, Switch } from 'react-router-dom';

// pages

import Home from './Home';

import About from './About';

import People from './People';

import Error from './Error';

import Person from './Person';

// navbar

import Navbar from './Navbar';

const ReactRouterSetup = () => {

  return (

    <*Router*>

        <*Switch*>

          <*Route* *exact* *path*="/">

              <*Home* />

          </*Route*>

          <*Route* *path*="/about">

              <*About* />

          </*Route*>

          <*Route* *path*="/people">

              <*People* />

          </*Route*>

          <*Route* *path*="\*">

                <*Error* />

          </*Route*>

        </*Switch*>

    </*Router*>

  );

};

export default ReactRouterSetup;

React Router Links

Agora vamos falar um pouco sobre links, no projeto isso será tipo um navbar para que cliquemos e possamos ir para a rota desejada de uma forma simples. Primeiro abaixo do Router e antes do Switch insira a navbar ela irá aparecer em todas as páginas pois não está no switch, <Navbar />, agora vamos ao arquivo navbar.js e iremos criar o Navbar dentro do return crie um nav com uma lista sem ordem com 3 linhas dentro <nav><ul> <li></li> <li></li> <li></li> </ul></nav> observe que estamos importando do react-router um Link ele que irá fazer o trabalho de direção dentro das linhas insira o componente Link com o parâmetro to ele tem que ter o mesmo nome da path para poder redirecionar você <li><Link to =”/”>Home</Link></li>, vá também no Error.js e insira o Link <Link to =”/” className=”btn”> Back to Home</Link>.

import React from 'react';

import { Link } from 'react-router-dom';

const Navbar = () => {

  return (

    <nav>

      <ul>

        <li>

          <*Link* *to*="/">Home</*Link*>

        </li>

        <li>

          <*Link* *to*="/about">About</*Link*>

        </li>

        <li>

          <*Link* *to*="/people">People</*Link*>

        </li>

      </ul>

    </nav>

  )

};

export default Navbar;

React Router URL Params and Placeholder

Agora por ultimo do tópico vamos ver como lidamos com uma lista no caso do componente em People.js nota-se que temos uma lista de pessoas que vem dos dados importados de data, mas o que a gente quer fazer é colocar um placeholder imaginando que seja uma lista de itens, que quando clicamos em um único item especifico iremos pra sua página especifica e será mostrado, essencialmente você não tem que criar o número diferente de páginas para cada item da lista, nesse caso a gente precisa criar um placeholder que simplesmente quando você clica em um item ele será direcionado por um único dado especifico sendo ele o id ou qualquer outro dado único.

Navegando o index.js vemos que temos um componente para person que ainda não usamos entre em Person.js, nesse arquivo temos apenas um texto para person igual nos outros componentes quando iniciamos. Agora vá em index.js precisamos setar a rota para isso abaixo da rota de people adicione <Route><Route/> no path será um pouco diferente precisamos adicionar um parâmetro, a primeira parte pode ser a sua escolha mas a segunda tem que ser colocado como o dado único a ser pesquisado na lista por exemplo: nome, id, etc..., no meu irei colocar <Route path=” /person/:id”><Route/>, ainda dentro de Route colocamos um children e adicionamos qual o componente que queremos mostrar ficará assim <Route path=” /person/:id” children={<Person/>}><Route/>, então toda vez que navegamos por person e clicamos nesse id irá mostrar essa pessoa. A diferença é que dentro do componente Person iremos pegar esse id e fetch os dados para mostrar essa pessoa especifica.

Index.js

import React from 'react';

// react router

import { *BrowserRouter* as Router, Route, Switch } from 'react-router-dom';

// pages

import Home from './Home';

import About from './About';

import People from './People';

import Error from './Error';

import Person from './Person';

// navbar

import Navbar from './Navbar';

const ReactRouterSetup = () => {

  return (

    <*Router*>

      <*Navbar* />

        <*Switch*>

          <*Route* *exact* *path*="/">

              <*Home* />

          </*Route*>

          <*Route* *path*="/about">

              <*About* />

          </*Route*>

          <*Route* *path*="/people">

              <*People* />

          </*Route*>

          <*Route* *path*="/person/:id" *children*={<*Person* />}>

          </*Route*>

          <*Route* *path*="\*">

                <*Error* />

          </*Route*>

        </*Switch*>

    </*Router*>

  );

};

export default ReactRouterSetup;

Agora em People.js logo abaixo do <h4> precisamos do link componente agora diferente dos outros links precisamos colocar dinamicamente usando o prefixo to e dentro das chaves adicionar um template string e adicionar a rota no caso person e dentro adicionar o id ficará assim. <Link to ={`/person/${person.id}`}>Learn More</Link> isso fará que quando você clique em um dos itens ele abrirá todos na mesma página a diferença será na URL onde os ids irão mudar, observe que sempre clicamos em sempre iremos mostrar a página Person. O problema agora é como iremos acessar aquele dado.

import React, { useState } from 'react';

import { data } from '../../../data';

import { Link } from 'react-router-dom';

const People = () => {

  const [people, setPeople] = useState(data);

  return (

    <div>

      <h1>People Page</h1>

      {people.map((*person*) => {

        return (

          <div *key*={*person*.id} *className*='item'>

            <h4>{*person*.name}</h4>

          <*Link* *to*= {`/person/${*person*.id}`}>Learn More</*Link*>

          </div>

        );

      })}

    </div>

  );

};

export default People;

Agora vá em Person.js observe que ele só tem um texto informando a pessoa, dentro precisamos usar o Hook do react-router useParams para testar se invocamos os parâmetros adicione um console.log e dentro dele chame useParams observe que ele retorna um objeto e dentro do objeto temos a propriedade id ela irá retornar um number o problema e que a propriedade sempre será uma String se os dados forem strings sem problema mas se forem numbers podem ocorrer uns pequenos erros que serão corrigidos a seguir.

Como estamos recebendo um objeto do useParams iremos desestruturá-lo usando const {id}= useParams(); precisamos também inserir o State value, const [name,setName] = useState(‘default name’) , agora dentro do <h1> precisamos mostrar <h1>{name}<h1/> também iremos adicionar um link para navegar de volta para a página de pessoa <Link to='/people' className='btn'>Back to People</Link> agora quando clicamos na pessoa repare que aprece default name e back to people porém as boas noticias e que eu tenho o meu id e podemos usar o useEffect para selecionar a pessoa e mudar o nome, agora crie um useEffect(()=>{},[]) ,dentro da função crie uma variável para procurar o dado usando o find método , const newPerson = data.find() se adicionarmos a simples função dentro do find (person)=> person.id === id, não iremos encontrar nada pois o dado que usamos e number o retornado e String para isso adicione no id um parseInt(id) e agora setName(newPerson.name) e assim que lidamos com uma lista ao invés de criar sempre uma nova página nos usamos a mesma página.

Person.js

import React, { useState, useEffect } from 'react';

import { data } from '../../../data';

import { Link, useParams } from 'react-router-dom';

const Person = () => {

  const [name, setName] = useState('default name');

  const { id } = useParams();

  useEffect(() => {

    const newPerson = data.find((*person*) => *person*.id === parseInt(id));

    setName(newPerson.name);

  }, []);

  return (

    <div>

      <h1>{name}</h1>

      <*Link* *to*='/people' *className*='btn'>

        Back to People

      </*Link*>

    </div>

  );

};

export default Person;

React.Memo

Agora para o próximo exemplo insira no App.js o caminho local para o tutorial que será executado mude para import Setup from './tutorial/12-memo-useMemo-useCallback/setup'; o que você verá na tela será um contador e uma lista de produtos. Agora vá para o arquivo index.js dentro da pasta 12-memo-useMemo-useCallback e observe que temos alguns imports como o useState, useEffect, useCallback, useMemo e o useFetch (custom hook), o ponto primeiro é se adicionarmos um useEffect nos componentes BigList e SingleProduct toda vez que você clicar no botão de adicionar o contador ele está re-renderizando tudo novamente, adicione o useEffect em BigList: useEffect(()=>{console.log(‘big list called’)}), você pode adicionar o array ao final do useEffect que também não tem problema e em SingleProduct: useEffect(()=>{console.log(‘single item called’)}) agora abra o console e note uma vez que chamamos o componente toda vez que clicamos os valores continuam aumentando e a lista e chamada isso acontece porque temos o valor do contador usando o useState ele mantém os valores entre os renders e ativa um re-render, dentro do componente Index estamos chamando o componente BigList e dentro componente BigList temos o componente SingleProduct e por isso todos eles serão re-renderizados, para resolver esse erro usamos o React.memo não confunda com o useMemo eles são diferentes, agora na frente do igual de BigList adicione o React.Memo e englobe todo o componente, agora que salvamos observe o render inicial foi mostrado porém quando clicamos no botão ele não ativa mais o re-render, pois ele checa ou memoriza qual é o valor e se a propriedade de valor não muda não ativamos mais o re-render e temos que lembrar que toda vez que as props ou state muda, o componente re-renderiza.

Index.js

import React, { useState, useEffect, useCallback, useMemo } from 'react'

import { useFetch } from '../../9-custom-hooks/final/2-useFetch'

// ATTENTION!!!!!!!!!!

// I SWITCHED TO PERMANENT DOMAIN

const url = 'https://course-api.com/javascript-store-products'

// every time props or state changes, component re-renders

const Index = () => {

  const { products } = useFetch(url)

  const [count, setCount] = useState(0)

  return (

    <>

      <h1>Count : {count}</h1>

      <button *className*='btn' *onClick*={() => setCount(count + 1)}>

        click me

      </button>

      <*BigList* *products*={products} />

    </>

  )

}

const BigList = *React*.memo(({ *products* }) => {

  useEffect(() =>{

    console.log('big list called');

  })

  return (

    <section *className*='products'>

      {*products*.map((*product*) => {

        return <*SingleProduct* *key*={*product*.id} {...product}></*SingleProduct*>

      })}

    </section>

  )

})

const SingleProduct = ({ *fields* }) => {

  useEffect(() =>{

    console.count('single item called');

  })

  let { name, price } = *fields*

  price = price / 100

  const image = *fields*.image[0].url

  return (

    <article *className*='product'>

      <img *src*={image} *alt*={name} />

      <h4>{name}</h4>

      <p>${price}</p>

    </article>

  )

}

export default Index

useCallback

Agora vamos similar um carrinho de compras no exercício adicionando o botão abaixo de cada produto e em cima um contador de quantas vezes for adicionado no carrinho. Primeiro crie no componente Index um useState, const [cart, setCart] = useState (0), e depois iremos criar uma função para adicionar no carrinho const addToCart = () => {setCart(cart+1)}, agora temos que passar essa função para cada item, logo após o botão do contador adicione um <h1 style={{marginTop:'3rem'}}> cart: {cart}</h1>, agora vamos passar a função para os componentes fazendo um prop drilling no BigList componente em Index adicione a propriedade addToCart={addToCart}, no componente BigList destrua a propriedade adicionando addToCart junto de products e dentro de BigList em SingleProduct adicione addToCart={addToCart} e finalmente no componente SingleProduct destrua addToCart e crie um botão logo abaixo de <p> que irá receber a função de adicionar no carrinho <button onClick={addToCart}>add to cart</button> e agora temos um problema e nota-se que estamos re-renderizando todos os itens novamente, a solução é usar o useCallback que faz uma coisa similar ao React.memo porém ele faz com funções na função addToCart adicione o useCallback e englobe-a e também precisamos adicionar o array de dependência chamando a variável do useState, const addToCart = useCallback(() => {setCart(cart + 1)

}, [cart]), agora observe que quando clicamos no botão de click me para o contador ele não re-renderiza, antes do useCallback ele ativava sempre re-rendizando e agora ele só re-renderiza quando adicionamos no carrinho e isso está correto.

Index.js

import React, { useState, useEffect, useCallback, useMemo } from 'react'

import { useFetch } from '../../9-custom-hooks/final/2-useFetch'

// ATTENTION!!!!!!!!!!

// I SWITCHED TO PERMANENT DOMAIN

const url = 'https://course-api.com/javascript-store-products'

// every time props or state changes, component re-renders

const Index = () => {

  const { products } = useFetch(url)

  const [count, setCount] = useState(0)

  const [cart,setCart] = useState(0)

  const addToCart = useCallback(() => {

    setCart(cart + 1)

  },[cart])

  return (

    <>

      <h1>Count : {count}</h1>

      <button *className*='btn' *onClick*={() => setCount(count + 1)}>

        click me

      </button>

      <h1 *style*={{marginTop:'3rem'}}>cart : {cart}</h1>

      <*BigList* *products*={products} *addToCart*={addToCart} />

    </>

  )

}

const BigList = *React*.memo(({ *products*, *addToCart* }) => {

  useEffect(() =>{

    console.log('big list called');

  })

  return (

    <section *className*='products'>

      {*products*.map((*product*) => {

        return <*SingleProduct* *key*={*product*.id} *addToCart*={*addToCart*} {...product}></*SingleProduct*>

      })}

    </section>

  )

})

const SingleProduct = ({ *fields*, *addToCart* }) => {

  useEffect(() =>{

    console.count('single item called');

  })

  let { name, price } = *fields*

  price = price / 100

  const image = *fields*.image[0].url

  return (

    <article *className*='product'>

      <img *src*={image} *alt*={name} />

      <h4>{name}</h4>

      <p>${price}</p>

      <button *onClick*={*addToCart*}>add to cart</button>

    </article>

  )

}

export default Index

useMemo

O useMemo hook diferente do useCallback que memoriza a função, então no caso do exemplo anterior se o valor do carrinho não muda isso significa que não precisamos criar a função do começo, o useMemo lida com o valor. Primeiro crie uma função que retorna algum valor const calculateMostExpensive = (data) =>{console.log('hello') return data.reduce((total,item)=>{const price = item.fields.price; if(price >= total){total = price};return total;},0)/100} essa função irá calcular o valor do item mais caro da loja, depois insira um <h1> abaixo do carrinho com os seguintes dados <h1>Most Expensive: ${calculateMostExpensive(products)}</h1>, nós precisamos ficar na cabeça que estamos falando de funções que precisam de um longo tempo para calcular, então a função retorna um valor imagine que ela leva um longo tempo para calcular então toda vez que clicamos no botão click me no contador nos estamos chamando o re-render, agora é onde o useMemo é utilizado criaremos uma função const mostExpensive = useMemo(()=>{calculateMostExpensive(products)},[products]), e agora no <h1> mude para mostExpensive, agora você nota que o valor continua do mesmo jeito então toda vez que você clica no botão a gente não vê esse ‘hello’ isso significa que não estamos chamando essa função, nós lembramos o valor com o useMemo e somente quando a propriedade de produtos muda a função rodará novamente e assim irá chamar o ‘hello’ no caso.

Index.js

import React, { useState, useEffect, useCallback, useMemo } from 'react'

import { useFetch } from '../../9-custom-hooks/final/2-useFetch'

// ATTENTION!!!!!!!!!!

// I SWITCHED TO PERMANENT DOMAIN

const url = 'https://course-api.com/javascript-store-products'

// every time props or state changes, component re-renders

const calculateMostExpensive = (*data*) =>{

  console.log('hello')

  return *data*.reduce((*total*,*item*)=>{

    const price = *item*.fields.price

    if(price >= *total*){

*total* = price

    }

    return *total*

  },0)/100

}

const Index = () => {

  const { products } = useFetch(url)

  const [count, setCount] = useState(0)

  const [cart,setCart] = useState(0)

  const addToCart = useCallback(() => {

    setCart(cart + 1)

  },[cart])

  const mostExpensive = useMemo(()=>{

    calculateMostExpensive(products)

  },[products])

  return (

    <>

      <h1>Count : {count}</h1>

      <button *className*='btn' *onClick*={() => setCount(count + 1)}>

        click me

      </button>

      <h1 *style*={{marginTop:'3rem'}}>cart : {cart}</h1>

      <h1> Most Expensive: ${mostExpensive}</h1>

      <*BigList* *products*={products} *addToCart*={addToCart} />

    </>

  )

}

const BigList = *React*.memo(({ *products*, *addToCart* }) => {

  useEffect(() =>{

    console.log('big list called');

  })

  return (

    <section *className*='products'>

      {*products*.map((*product*) => {

        return <*SingleProduct* *key*={*product*.id} *addToCart*={addToCart} {...product}></*SingleProduct*>

      })}

    </section>

  )

})

const SingleProduct = ({ *fields*, *addToCart* }) => {

  useEffect(() =>{

    console.count('single item called');

  })

  let { name, price } = *fields*

  price = price / 100

  const image = *fields*.image[0].url

  return (

    <article *className*='product'>

      <img *src*={image} *alt*={name} />

      <h4>{name}</h4>

      <p>${price}</p>

      <button *onClick*={*addToCart*}>add to cart</button>

    </article>

  )

}

export default Index

useCallback-Fetch Example

E a última coisa que iremos abordar e um outro uso para o useCallback, vá na pasta de arquivos 9-custom-hooks em useFetch.js, observe que o terminal está chamando um erro para as dependências do useEffect no caso adicionar o getProducts ao array de dependência, porém se você fazer isso de bate e pronto você irá cair em um loop infinito de re-renderização, a solução para isso é adicionar no import o useCallback e englobar toda a função getProducts e no final adicionar um array de dependência quando os dados mudarem no caso url, e quando temos isso setado adicione o getProducts no array de dependência do useEffect.

useFetch.js

import { useState, useEffect, useCallback } from 'react';

export const useFetch = (*url*) => {

  const [loading, setLoading] = useState(true);

  const [products, setProducts] = useState([]);

  const getProducts = useCallback(async () => {

    const response = await fetch(*url*);

    const products = await response.json();

    setProducts(products);

    setLoading(false);

  }, [*url*]);

  useEffect(() => {

    getProducts();

  }, [*url*, getProducts]);

  return { loading, products };

};