Site do professor

https://www.otaviomiranda.com.br/

- Uma vez (cria projeto): npx create-react-app . (ou nome de pasta ao invés do ponto)

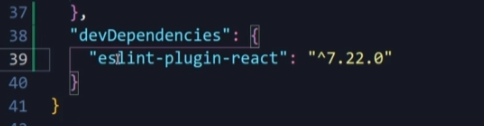
- npm start na pasta do projeto via terminal

- Caso compilar – npm run build

No caso de utilizarmos em um site existente, *podemos incluir react em um site existente facilmente com:* [*https://pt-br.reactjs.org/docs/add-react-to-a-website.html*](https://pt-br.reactjs.org/docs/add-react-to-a-website.html)

Dicas:

Caso precisarmos retirar o plugin ou outra coisa do projeto como o eslint (para evitar conflitos e etc), temos que deletar a linha que não vamos utilizar



deletar o package-lock.json e a node modules para depois usar o npm i para reinstalar tudo novamente.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sempre que o estado muda no react, a função render é chamada novamente.

Link eventos sintéticos – São eventos do React similares aos eventos reais de mesmo nome do JS: [*https://pt-br.reactjs.org/docs/events.html*](https://pt-br.reactjs.org/docs/events.html) Manipuladores de evento (event handlers) serão passados como instâncias do SyntheticEvent, um agregador cross-browser que envolve os eventos nativos do navegador.

Exemplos: onKeyUp, onFocus, onBlur, onSubmit, onLoad, onClick

A diferença no react é que não se precisa usar o addEventListener  exemplo: [*https://www.w3schools.com/jsref/met\_element\_addeventlistener.asp*](https://www.w3schools.com/jsref/met_element_addeventlistener.asp)

exemplos de js com o addEventListener:

*<button id="myBtn">Try it</button>*

*<script>*

*const element = document.getElementById("myBtn");*

*element.addEventListener("mouseover", myFunction);*

*function myFunction() {*

*document.getElementById("demo").innerHTML += "Moused over!<br>"*

*}*

*</script>*

-Duas maneiras de passar argumentos para manipuladores de eventos no React ([*https://pt-br.reactjs.org/docs/handling-events.html*](https://pt-br.reactjs.org/docs/handling-events.html))

(o de baixo poderia ir sem parâmetros, por ex.)

<button onClick={(e) => this.deleteRow(id, e)}>Deletar linha</button>

<button onClick={this.deleteRow.bind(this, id)}>Deletar linha</button>

Exemplo de manipulação no HTML:

<button onclick="activateLasers()">

Ativar lasers

</button>

No react:

<button onClick={activateLasers}>

Ativar lasers

</button>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

O spread “...” ([*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread\_syntax*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax)) distribui elementos de um array:

const array = [1, 2, 3];

const obj = { ...array }; // { 0: 1, 1: 2, 2: 3 }

Outro exemplo:

function sum(x, y, z) {

return x + y + z;

}

const numbers = [1, 2, 3];

console.log(sum(...numbers));

// expected output: 6

console.log(sum.apply(null, numbers));

// expected output: 6

- É a mesma sintaxe e exatamente o oposto da função de Rest parameters que aceita um indefinido número de elementos de um array:

function sum(...theArgs) {

let total = 0;

for (const arg of theArgs) {

total += arg;

}

return total;

}

console.log(sum(1, 2, 3));

// expected output: 6

console.log(sum(1, 2, 3, 4));

// expected output: 10

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Avaliação de curto-circuito ( [*https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions\_and\_Operators*](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators) )

Como expressões lógicas são avaliadas da esquerda para a direita, elas são testadas como possíveis avaliações de "curto-circuito" utilizando as seguintes regras:

false && qualquercoisa é avaliado em curto-circuito como falso.

true || qualquercoisa é avaliado em curto-circuito como verdadeiro.

Exemplo:

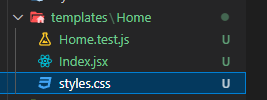
  {filteredPosts.length > 0 && (

          <Posts posts={filteredPosts} />

        )}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sempre o componente vai se chamar index e o css styles dentro da pasta.



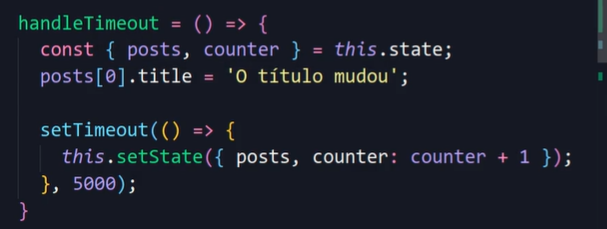
- em uma importação, o ./ significa a pasta que se está (básico)

import './styles.css';

- Se uma função não for construtora e nem componente nem classe, começa com letra minúscula exe.:



-O nome padrão usado para uma função que vai alterar algo na página seria handle, não que seja obrigado.



-Por padrão podemos usar a extensão jsx ao invés de js, por exemplo dentro de um componente um arquivo index.jsx e quando for typescript será tsx

- Um componente react é uma classe que tem um método render e retorna jsx ou uma função que retorna jsx também.

- sempre que usar map, tem que ter uma key única nos elementos exemplo:

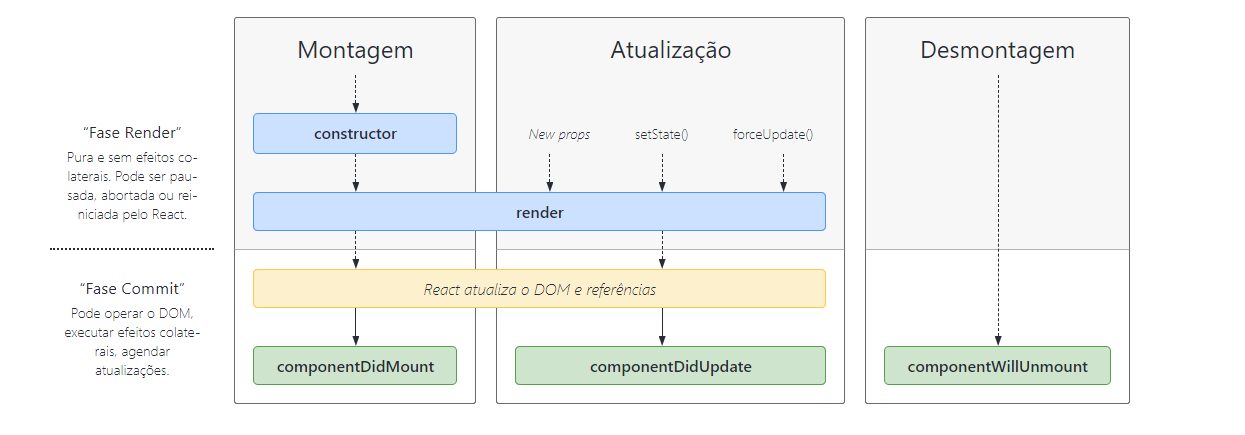
 <PostCard key = {post.id}   title={post.title}  body={post.body}  />

event.preventDefault(); // não quero que o evento faça o que ele ia fazer antes (abrir o site)

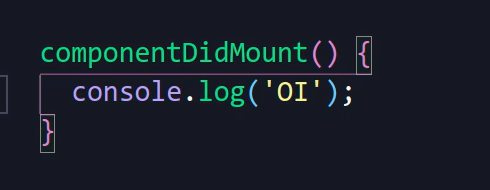
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/map // map do javascript

Ciclo de vida mais comuns utilizados:

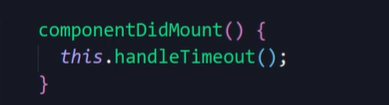
[*https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/*](https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/)



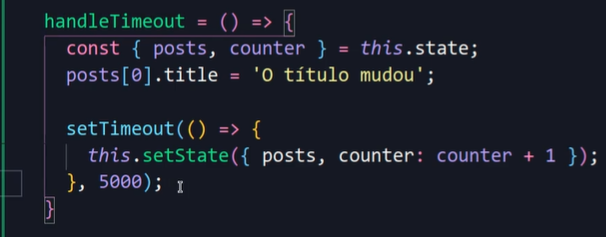
Exemplo de componente que será executado assim que o componente for montado na tela:



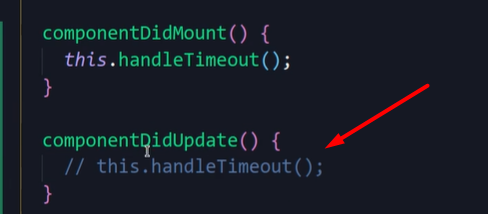
“Quando o componente for montado por inteiro, uma função será acionada”:



Neste exemplo, a função chamada irá atualizar o estado depois de 5 segundos:



Se utilizarmos a função de ciclo de vida componentDidUpdate chamando novamente a handleTimeout, teremos um looping infinito



Outros:

<https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops> // métodos The Component Lifecycle

<https://jsonplaceholder.typicode.com/> // uma api para testes chamada JSON Placeholder

Vamos usar o fetch api para recuperar dados na internet que é Javascript puro (poderíamos usar o axios ou outro pra fetching de dados):

[*https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch\_API*](https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API)

\_\_\_\_\_

Iniciamos o css com

\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

o border-box é necessário para que, ao incluirmos um padding em um elemento, ele entre por dentro do tamanho e não aumente o tamanho de uma div:

<https://www.w3schools.com/css/css3_box-sizing.asp>

Unidede de css vh

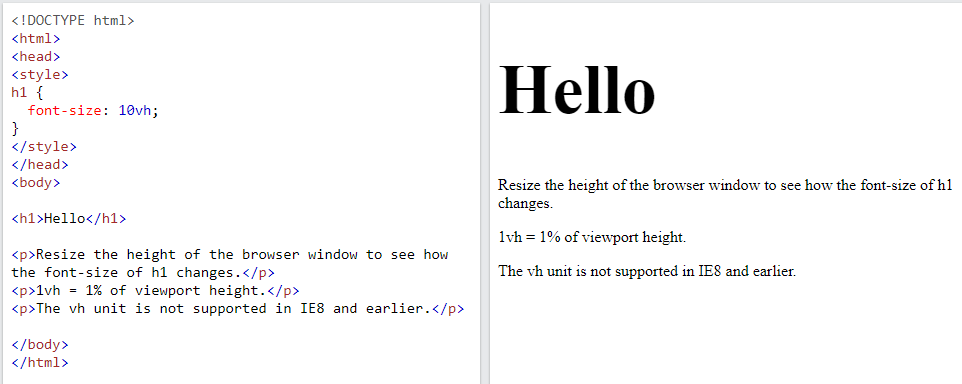
.container {

  min-height: 100vh;

}

vh = 1% of viewport height.

Exemplo



Conforme a tela muda de tamanho, o “Hello” do h1 vai se alterando conforme a percentagem de altura da tela.

Props

App.js

  render() {

    const { posts, counter } = this.state;

    return (

      <section className='container'>

        <div className="posts">

          {posts.map(post => (

              <PostCard

              key = {post.id}

              title={post.title}

              body={post.body}

              cover={post.cover}

              id={post.id}

              />

            ))

          }

        </div>

      </section>

    );

  }

no componente caso não tenha nada entre o inicio e o return:

export const PostCard = ({title,cover, body, id }) =>  (

        <div className="post">

            <img src={cover} alt={title} />

            <div key={id} className="post-content">

                <h1 >{title}</h1>

                <p>{body}</p>

            </div>

        </div>

    );

Caso tenha, temos que fazer com o return:

export const PostCard = ({title,cover, body, id }) => {

    return (

        <div className="post">

            <img src={cover} alt={title} />

            <div key={id} className="post-content">

                <h1 >{title}</h1>

                <p>{body}</p>

            </div>

        </div>

    );

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Dicas JS**

Converter para booleano (igual no php, porem o true se testa com duas exclamações e o false com uma só, fiz uns testes e da pra tratar igual no php):

 { !!searchValue }

Vai testar se existir esse valor será true senão será false

No exemplo abaixo, “se isto for verdadeiro, faça isto que vem depois do &&” .

    {!!searchValue && (

          <>

            <h1>Search value: {searchValue}  </h1>

          </>

       )}

No caso abaixo, “se não houver busca, aparece o botão”:

    {!searchValue && (

          <Button text='Load more posts'

            onClick={this.loadMorePosts}

            disabled={noMorePosts}

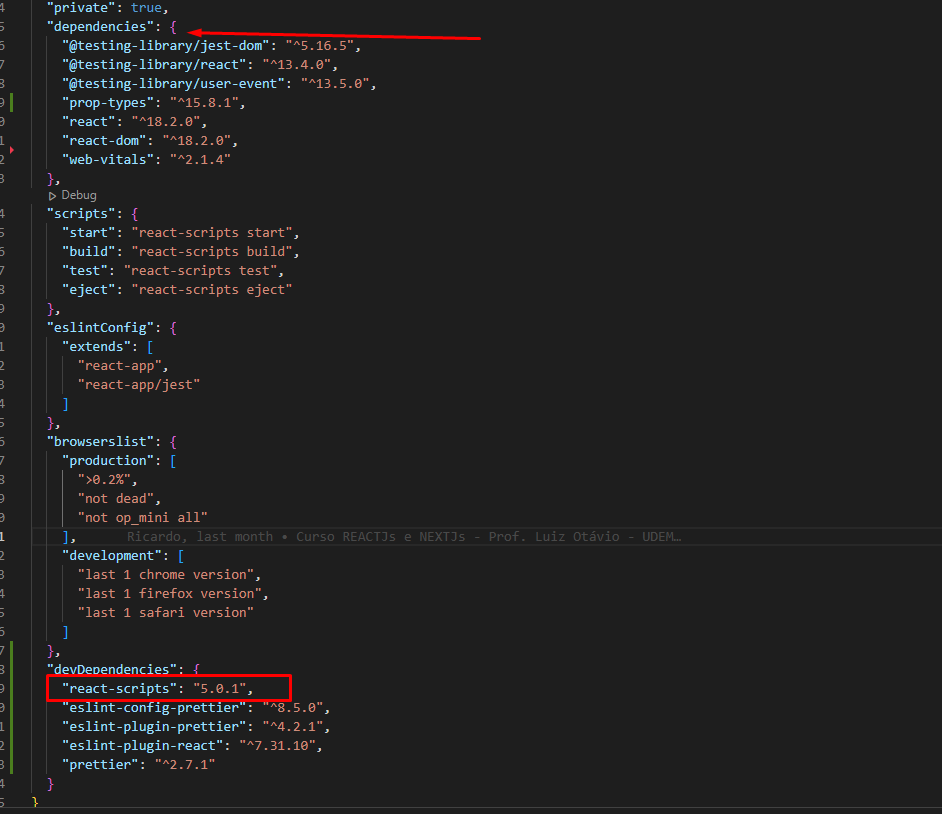
          />

        )}

Bugs:

Corrigi um erro grave *High - Inefficient Regular Expression Complexity in nth-check* (pesquisei e descobri que era apenas um aviso que se portava como erro mas só em desenvolvimento)

Para corrigir, no package.json, trocar esta linha de lugar (de dependencies para dev dependencies)



deletei a package-lock.json e a node-modules e dei um npm i

Comprovei que não tinha nenhum problema com o comando npm audit –production , indicando que em produção o erro não existe.

Podemos fazer os imports de duas maneiras. A primeira com a forma mais longa:



E a segunda mais usada:

