**Semana OmniStack – 10° Edição**

*Front-end*

# Abordagens

## Abordagem Tradicional



Não existe um framework do lado do front-end.

Essa limitação implica que o backend retorna informações que só serão mostradas em ambientes HTML, limitando a chamada de outros serviços, como por exemplo, um app, outras interfaces.... Se fosse realizar uma troca para, por exemplo, o mobile, teria que refazer as rotas adequadas para o mobile, pois elas foram feitas de forma a atender ao HTML e não de forma geral.

O backend fica com TODO o trabalho da aplicação.

O backend fica com TODO o trabalho da aplicação.

O backend fica com TODO o trabalho da aplicação.

## Abordagem de SPA



Essa abordagem **otimiza** a **performance da aplicaçã**o, pois, não há mais a obrigatoriedade de a cada nova requisição ir ao back-end.

A resposta é apenas em JSON, e não a página HTML inteira.

Assim, o frontend fica responsável por receber esse JSON e trata-lo para a apresentação.

SPA: Single Page Application – **Aplicações em uma única página**.

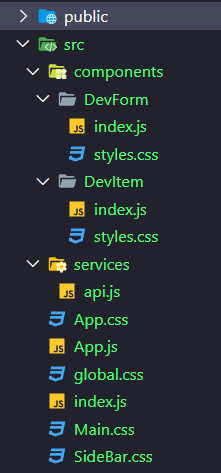
Utilização de frameworks, dessa forma, o backend não tem a responsabilidade de preparar a parte visual.

# Criação do Projeto

Comando: yarn create react-app nome\_projeto OU npx create-react-app nome\_projeto

Inicialização do projeto: yarn start

## Organização de pastas



### Public

Pasta que contém informações públicas da aplicação.

* Favicon: ícone que aparece na aba do navegador.
* Index: arquivo inicial da aplicação.
* PWA – progressive web apps: criação de aplicativos mobile baseado em sites HTML, CSS, JS (logo192.png, logo512.png, manifest.json).
* Robots.txt: informa ao google quais páginas da aplicação desejam ser indexada (rastreada).

Todas as páginas são rastreáveis.

# https://www.robotstxt.org/robotstxt.html

User-agent: \*

# Funcionamento React

Todo HTML e CSS da aplicação (parte visual), é gerada a partir de JavaScript – o HTML está dentro do JS.

É utilizado tags HTML isso facilita o processo de criação da página.

A página da aplicação está em src: index.js (não pode mudar o nome).

Index.js: padrão – não muda – todas as aplicações React seguem esse modelo.

//importação do React

//é necessário importar o React em todo arquivo JS que irá usar JSX

//JSX = JavaScript + XML (sintaxe do HTML)

import React from 'react';

//O ReactDom dá a habilidade do React se comunicar com a árvore de elementos (HTML) da aplicação

import ReactDOM from 'react-dom';

import App from './App';

//pega o reactDom e manda renderizar o App (conteúdo da página) dentro da div 'root'

ReactDOM.render(<*App* />, document.getElementById('root'));

## 3 conceitos principais React

Tudo no React é baseado nos três conceitos: Componente, Estado e Propriedade.

### Componente

É uma função que retorna um JSX.

A aplicação é composta por vários blocos (componentes). Componentes são módulos, que podem ser separados da aplicação sem comprometer o funcionamento de outras partes.

A primeira letra sempre é maiúscula.

No React deve haver apenas 1 componente por arquivo.

Não é possível ter um componente um de baixo do outro sem ter um outro componente que os englobe, por exemplo, uma div, fragment.

Toda função que é própria do componente fica dentro do componente, dentro do function.

import React from 'react'

function Header() {

    return (

      <h1>Header!</h1>

    )

}

export default Header

#### Fragment

É uma tag sem nomenclatura: <>...</>

Ex:

<>

      <*Header* *title*="Dashboard"/>

      <*Header* *title*="Título"/>

      <*Header* *title*="Conteúdo"/>

</>

### Estado

Informação que o componente vai manipular, manter e ler.

Se não utilizar o estado as informações não serão atualizadas.

Conceito de imutabilidade: conceito que sempre que você for manipular uma informação do estado, vc cria uma cópia do mesmo, para dps manipular, e não modificar diretamente. Uso da função setCounter() para mudar o estado e useState() para colocar o valor inicial.

//useState: função utilizada pelo React para criar um estado

import React, { useState } from 'react';

//o app é uma função que retorna o conteúdo HTML (JSX)

function App() {

  //operador destructure para pegar as variáveis. Pode dar o nome que quiser

  const [ counter, setCounter ] = useState(0)

  function incrementCounter() {

    setCounter(counter + 1)

  }

  return (

    <>

      <h1>Contador: {counter}</h1>

  <button *onClick*={incrementCounter}>Incrementar</button>

    </>

  );

}

export default App;

### Propriedade

São os atributos.

São informações que o componente PAI passa para o componente FILHO.

Nome de utilização: props – são todas as propriedades passadas ao componente.

import React from 'react'

//coloca as chaves pois informa que irá introduzir um conteúdo JS (variáveis)

function Header(*props*) {

    return (

      <h1>{props.title}</h1>

    )

}

export default Header

import React from 'react';

import Header from './Header'

//o app é uma função que retorna o conteúdo HTML (JSX)

function App() {

  return (

    <>

      <*Header* *title*="Dashboard"/>

      <*Header* *title*="Título"/>

      <*Header* *title*="Conteúdo"/>

    </>

  );

}

export default App;

# Documentação

Link: <https://reactjs.org/>.

# Curiosidades

Dica: comece sempre pela parte visual, para depois ir para a parte funcional.

Atalho para criar estrutura do div.

div.nome\_da\_classe\_desejada – só digitar e dar enter

## Extensão Chrome

Extensão do chrome para o react: mostra a arvore de elementos: react dev tools

## Aplicar Responsividade

/\*como colocar a responsividade: jeito mais simples\*/

/\*muda a direção quando atingir a largura X\*/

@media (max-width: 1000px) {

*#app* {

        flex-direction: column;

    }

    /\*retira a margin left\*/

*#app* main {

        margin-left: 0;

        margin-top: 30px;

    }

*#app* aside {

        width: 100%;

    }

}

@media (max-width: 650px) {

    main ul {

        grid-template-columns: 1fr;

    }

}

## Programação Imperativa

Oposto do declarativa em que você fala exatamente pro código o que você quer fazer.

Utiliza a criação de um estado e o componente se comporta baseado no estado

## APIs

### API de Contexto

API de contexto do React facilita a transição de dados dentro do componente PAI e o FILHO para evitar utilizar sempre pela propriedade.

### Geolocalização do Navegador

//param1: qual função precisa executar

  //param2: quando a função precisa executar - [] - se o vetor estiver vazio, a função irá executar uma única vez

  //se colocar uma variável, sempre que ela mudar, irá executar o código

  useEffect(() => {

    //retorna primeiro a função de sucesso

    //segundo a função de erro

    //no final passa alguns parâmetros para o getCurrentPosition

    //enableHighAccuracy: habilitar pegar a posição do usuário muito precisamente : alta precisão

    //timeout: 30 segundos

    navigator.geolocation.getCurrentPosition(

      (*position*) => {

        const { latitude, longitude } = position.coords

        setLatitude(latitude)

        setLongitude(longitude)

      },

Utiliza a função useEffect para ser executado uma única vez quando o componente for renderizado.

      (*err*) => {

*console*.log(err)

      },

      {

         timeout: 30000,

      }

    )

  }, [])

const [ latitude, setLatitude ] = useState('')

  const [ longitude, setLongitude ] = useState('')

 <div *className*="input-group">

            <div *className*="input-block">

              <label *htmlFor*="latitude">Latitude</label>

              <input *type*="number" *name*="latitude" *id*="latitude" *required* *value*={latitude}

Atualiza no estado quando modificar o valor a partir do evento HTML passado

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

            </div>

            <div *className*="input-block">

              <label *htmlFor*="longitude">Longitude</label>

              <input *type*="number" *name*="longitude" *id*="longitude" *required* *value*={longitude}

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

            </div>

          </div>

## Acessar a API do backend

Modificação: é necessário instalar uma extensão que permita que o node receba chamadas do mesmo endereço, ele so aceita de endereços distintos. Deve realizar essa modificação, pois, o frontend e o backend estão, ambos, no localhost.

Comando: yarn add cors – cors: cross origin resource sharing.

NO BACKEND

const cors = require(cors)

//se não colocar o parâmetro permite o acesso interno para todo o tipo de aplicação

app.use(cors())

//determina o endereço que irá chamar

//app.use(cors({ origin: 'http://localhost:3000' }))

### Realizar as chamadas

Utilização na biblioteca axios.

Comando: yarn add axios

Criação da pasta services/api.js

import axios from 'axios'

const api = axios.create({

    baseURL: 'http://localhost:3333'

})

export default api

no App.js

//a listagem dos devs acontece uma única vez

  useEffect(() => {

    async function loadDevs() {

      const response = await api.get('/devs')

      setDevs(response.data)

    }

    //chama a execução da função

    loadDevs()

  }, [])

//quando o usuário clicar no submit

  async function handleAddDev(*e*){

    //preveni o comportamento padrão do HTML em formulário

    //que é de enviar o usuário para uma outra página

    e.preventDefault()

    //realiza a chamada a API para adicionar o dev na listagem

    //informa qual a URL e quais parâmetros irá enviar

    const response = await api.post('/devs', {

      github\_username,

      techs,

      latitude,

      longitude,

    })

    setGitHubUsername('')

    setTechs('')

    //copia todos os devs e adiciona o inserido

    setDevs([...devs, response.data])

  }