# React

Vamos aprender a usar o framework do react
Primeiros usamos o comando: **npm i create-react-app .**
Para criar uma pasta toda estruturada para nosso projeto

Arquivo .html


Arquivo .js


Mas como o js é integrado no html sem a tag <script>?

há uma interação importante entre o React e o arquivo index.html, mesmo sem a tag <script> explícita. Isso acontece porque em projetos React modernos (criados com Create React App, Vite ou similares), o sistema de build cuida dessa integração automaticamente. Vou explicar como isso funciona:

**O que acontece por baixo dos panos**

Ponto de entrada: O ** ** no HTML é o container onde sua aplicação React será renderizada.

**ReactDOM.createRoot**: No seu arquivo JavaScript (index.js),

A linha **ReactDOM.createRoot(document.getElementById('root'))** encontra esse elemento div e prepara-o para receber a aplicação React.

Processo de build: Quando você executa npm start ou npm run build, as ferramentas de build (como webpack no caso do Create React App) fazem o seguinte:
Empacotam todo seu código React em arquivos JavaScript otimizados
Injetam automaticamente as tags <script> necessárias no HTML final
Geram os arquivos finais na pasta build/ (ou similar)

**O que é Build em um Projeto React?**

Build é o processo de transformar seu código React (JSX, JavaScript moderno, CSS, assets) em arquivos otimizados que o navegador pode entender e executar. Isso inclui:

Transpilar código (ex: JSX → JavaScript puro, ES6+ → ES5 para compatibilidade)

Agrupar (bundle) módulos em arquivos menores e eficientes

Minificar código (remover espaços, encurtar nomes de variáveis)

Otimizar imagens e assets

Gerar HTML, CSS e JS prontos para produção

**O que é Webpack?**

Webpack é um module bundler (empacotador de módulos) para JavaScript. Ele pega vários arquivos (JavaScript, CSS, imagens, etc.) e os combina em pacotes otimizados para o navegador.

**Por que o Webpack é usado no React?**

Quando você escreve código React, geralmente:

Usa JSX (que não é entendido pelo navegador).

Divide o código em múltiplos arquivos e dependências.

Usa CSS, imagens, fonts e outros recursos.

O Webpack ajuda a:

✅ Transpilar JSX e ES6+ para JavaScript compatível com navegadores.

✅ Agrupar (bundle) todos os arquivos em um ou mais pacotes otimizados.

✅ Gerenciar dependências (como bibliotecas do node_modules).

✅ Otimizar código para produção (minificação, tree shaking).

✅ Carregar assets (CSS, imagens, fonts) como módulos.

src > App.js > ...

```jsx
//Componentes / react
import First from './components/FirstComponents.js';
//Style / CSS
import './App.css';
import foto from './assets/logo192.png';


function App() {
  return (
    <div>
      <h1>Hello, word!</h1>
      <First/>
      <img src={foto} alt="foto"></img>
    </div>
  );
}

export default App;
```

Essa é o arquivo onde irá funcionar nossa aplicação, onde escrevemos um JSX, no qual o index.js irá renderizar

## Componentes

Componentes são meios que são partes separadas que podem ser inseridos na pagina

src > components > FirstComponents.js > First

```jsx
const First = () => {
    const nome = "jean"
    return (
        <div>
            <div>
                <h2>olá, {nome}</h2>
                <button>Aperte aqui</button>
            </div>
            <div className="teste">

            </div>
        </div>
    );
};
export default First;
```

Para inserir js nessas partes, você usa o {var ou sintaxe js} e sempre dentro uma função return
Ai na hora de implentar o componete, você usa <componente/>.
Exemplo:

```
import First from './components/FirstComponents.js
//Style / CSS
import './App.css';
import foto from './assets/logo192.png';


function App() {
  return (
    <div>
      <h1>Hello, word!</h1>
      <First/>
```

OBS: em JSX, pode mudar um pouco algumas coisas que são em html
Exemplo, na hora de atribuir class, no jsx é className

**<h1 className='pika'>Hello, word!</h1>**

## Importação de fotos

Existe duas formas de importar foto, o tradicional que é src=
Ou assim:

```
import foto from './assets/logo192.png';


function App() {
  return (
    <div>
      <h1 className='pika'>Hello, word!</h1>
      <First/>
      <img src={foto} alt="foto"></img>
      <img src="/logo512.png" alt="pika"></img>
    </div>
  );
}
```

Importando a foto e atribuindo o source, mas a foto tem que ficar em assets por conta do webpack. O metodo tradicional, a foto tá na pasta public por

# Hooks

Os Hooks são funções especiais do React que permitem usar o estado (state) e outras funcionalidades do React sem precisar criar uma classe.

Use Hooks quando precisar armazenar valores dinâmicos e atualizar a UI automaticamente.

```jsx
import { useState } from 'react';

function Exemplo() {
  const [count, setCount] = useState(0); // Hook de estado

  function incrementar() {
    setCount(count + 1); // Atualiza o estado E re-renderiza o componente
  }

  return (
    <div>
      <p>Contador: {count}</p>
      <button onClick={incrementar}>+1</button>
    </div>
  );
}
```

```jsx
const nome = "jean";
const [list] = useState(["jean", "maria", "joão"]);
const [user, setUsers] = useState([{id: 1, nome: "jean"}, {id: 2, nome: "joao"}, {id: 3, nome: "slll"}]);

const deleteId = () => {
  const randomId = Math.floor(Math.random() * 4);
  setUsers(prevUser => { // O argumento é uma função de callback que retorna os valores modificados
    console.log(prevUser);
    return prevUser.filter(users => randomId !== users.id) // 'remove' só que for false - no caso, cria um array novo só com os valores q são true
  })
}
```

[count, setCount]

count → Armazena o valor atual do estado.

setCount(novoValor) → Atualiza count e avisa o React para re-renderizar o componente.

! O React não detecta mudanças em variáveis normais. Ele só re-renderiza o componente quando há uma mudança no estado.

Você pode criar um hook, e ele é diferente de um componente.

Um componente é usado para renderizar algo na tela, ou seja, ele retorna JSX dentro do return() e é chamado como uma tag (ex: <MeuComponente />).

Já o hook é uma função que encapsula lógica de estado ou efeitos colaterais, como o useState, useEffect, ou um hook personalizado (useMeuHook).

Você não usa hooks dentro do return(), mas sim dentro da função do componente, para manipular dados, lidar com estados, buscar informações etc.

Lembrando que todo hook personalizado tem que começar com ‘use’.

Veja um exemplo:

src > hook > showUser.js > ...

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

function useAccess(url) {
  const [data, setData] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const res = await fetch(url);
        console.log(res)
        const json = await res.json();
        setData(json);
      } catch (e) {
        console.log("Não foi possível obter informação: ", e);
      }
    };
    fetchData();
  }, [url]);

  return {data};
}

export default useAccess;
```

```
const url = "http://localhost:3000/product/"
const {data} = useAccess(url);
console.log(data);
```

hooks não podem retornar JSX! O useAccess é um hook, então ele só pode retornar dados ou funções, nunca elementos JSX como Carregando... .

OBS: os componentes precisam começar com a letra maiscula

**import *S*howCarro from './components/carros.js';**

## Condições

No react, as condições podem ser feita de 3 maneiras:
Condição

```jsx
const First = () => {
    const [x] = useState(1);
    const [y] = useState(true)
    return (
        <div>
            <div>
                {x > 6 ? (<p>Maior que 10</p>):(<p>Não é maior que 10</p>)}
                {y && <p>Valor verdadeiro</p>}
            </div>
        </div>
    );
};
export default First;
```

Uma delas, no qual, você pode perceber, é pela condição ternária
A outra é meio que do react mesmo, no caso, a variavel Y
Ela responde só uma condição, se for True, ela roda a condição

A outra é fora do return(), você escreve o if e else normal

```jsx
function MeuComponente({ logado }) {
  let mensagem;

  if (logado) {
    mensagem = <h1>Bem-vindo de volta!</h1>;
  } else {
    mensagem = <h1>Por favor, faça login.</h1>;
  }

  return <div>{mensagem}</div>;
}
```

# Props

Props são como mensagens/identidades que são passados para os componentes, no qual, podem ser tratadas. Props são passados como objetos

```
return (
  <div>
    <NameUser name="jean"/>
    <First />
```

São passados em atribuições com o nome  que você quiser atribuir, mas esse nome vai passar como propriedade

```
const nameUser = (props) =>{
    return(
        <div>
            <h1>Hello, {props.name}</h1>
        </div>
    );
};
export default nameUser;
```

Assim, eles são tratados
Você pode fazer destruturação
Via atribuição

Vamos ver um exemplo de props mais trabalhado

src > components > carros.js > default

```
1   const showCarro = (props) =>{
2       return(
3       <div>
4           <h2>Detalhe do carro</h2>
5           <p>Nome: {props.nome}</p>
6           <p>Modelo: {props.modelo}</p>
7           <p>cor: {props.cor}</p>
8       </div>
9       );
10  };
11  export default showCarro;
```

Componente

```jsx
//Componentes / react
import { useState } from 'react';
import First from './components/FirstComponents.js';
import NameUser from './components/showUser.js';
import ShowCarro from './components/carros.js';
//Style / CSS
import './App.css';
import foto from './assets/logo192.png';


function App() {
  const carros = [{
    nome: "cavalo",
    modelo: 'ferrari',
    cor: "vermelho"
  }, {
    nome: "leo santana",
    modelo: "camaro",
    cor: "amarelo",
  }, {
    nome: "funkeiro",
    modelo: "bwm",
    cor: "azul"
  }];

  return (
    <div>
      <NameUser name="jean"/>
      <First />
      <ul>
        {carros.map(esp => (
          <ShowCarro nome={esp.nome} modelo={esp.modelo} cor={esp.cor}/>
        ))}
      </ul>
    </div>
  ); // é necessário ter o key para o react identificar a chave
}

export default App;
```

Você pode trabalhar com .map(), .filter() nos arrays

Nesse exemplo, estamos fazendo um Reaproveitamento de componente. O mesmo componente sendo reutilzado

OBS: quando usamos um componente repetivamente, ou seja, reaproveitando. Você tem que especificar uma key, essa mesma key não irá aparecer na tag no front. Então, você tem que colocar alguma propriedade ID para especificar o key

```jsx
function App() {
  const carros = [{
    id: 1,
    nome: "cavalo",
    modelo: 'ferrari',
    cor: "vermelho"
  }, {
    id: 2,
    nome: "leo santana",
    modelo: "camaro",
    cor: "amarelo",
  }, {
    id: 3,
    nome: "funkeiro",
    modelo: "bwm",
    cor: "azul"
  }];

  return (
    <div>
      <NameUser name="jean">
        <h2>Seja bem, vindo</h2>
      </NameUser>
      <First />
      <ul>
        {carros.map(esp => (
          <ShowCarro key={esp.id} nome={esp.nome} modelo={esp.modelo} cor={esp.cor}/>
        ))}
      </ul>
    </div>
  ); // é necessário ter o key para o react identificar a chave
}

export default App;
```

Além do props, também tem o children, é quando você passa algo como filho dentro de um componente pai.

```jsx
<Teste>
  <p>Olá mundo</p>
</Teste>
```

Se você não tratar, ele não retorna nada. Então você tem que dar um return em children que é uma propriedade

```jsx
export default function teste({children}){
    return (
        <div>
            {children}
        </div>
    )
};
```

# onChange()

**onChange** é um evento do React (e também do HTML) que é disparado sempre que o valor de um input (ou textarea, select etc.) muda.

Ou seja, quando o usuário digita algo, seleciona uma opção diferente, ou faz qualquer mudança em um campo de formulário, o onChange é chamado.

```jsx
import { useState } from 'react';

function App() {
  const [nome, setNome] = useState('');

  const handleChange = (event) => {
    setNome(event.target.value);
  };

  return (
    <div>
      <input type="text" onChange={handleChange} />
      <p>Você digitou: {nome}</p>
    </div>
  );
}
```

```jsx
const handle = (e) => {
  if (e.target.value.trim() === "") {
    setName("Digte um titulo")
  } else {
    setName(e.target.value)
  }
}
```

**<input type="text" placeholder="titulo" onChange={handle}></input>**

# State lift

**Quando usar "lifting state up"?**

**"Lifting state up" significa mover um estado (useState) de um componente filho para um componente pai, quando dois ou mais componentes precisam acessar ou modificar esse mesmo estado.**

**Situação comum:**

**Imagine dois componentes irmãos (A e B). Eles não conseguem compartilhar diretamente um useState, pois cada um tem o seu próprio.**

**Aí entra o state lift:**

**Você move o estado para o componente pai comum, e passa ele por props.**

Quando dois componentes precisam acessar o mesmo estado.

Quando um componente precisa atualizar um dado e outro precisa mostrar esse dado.

Quando você começa a ter problemas com sincronização de valores entre irmãos.

```
  const [message, setMessage] = useState("");
  const ShowMsg = (msg) =>{
    setMessage(msg)
  }
  return (
    <div>
      <NameUser name="jean">
        <h2>Seja bem, vindo</h2>
      </NameUser>
      <First />
      <ul>
        {carros.map(esp => (
          <ShowCarro key={esp.id} nome={esp.nome} modelo={esp.modelo} cor={esp.cor}/>
        ))}
      </ul>
      <div>
        <Msg nota={message}/>
        <ShowMessage msg={ShowMsg} />
      </div>
    </div>
  );
}

export default App;
```

```
rc > components > msg.js > default
function msg(propos){
    return(
        <p>A messagem é: {propos.nota}</p>
    )
};
export default msg;
```

```
rc > components > showMessage.js > default
function Show({msg}){
    return(
        <button onClick={() => msg('Olá')}>Aperta aqui</button>
    );
}
export default Show;
```

OBS: quando a função possui um argumento, você não pode passar a função direto no onClick, você tem que criar outra função de callback

***<button onClick={msg}>Aperta aqui</button>***

Sem argumento

# Projeto simples

```
src > App.js > ...
//Componentes / react
import { useState } from 'react';
import Msg from './components/msg.js';
//Style / CSS
import './App.css';


function App() {

  const cor = false;

  return (
    <div>
      <h2 style={cor ? ({color: "red"}) : ({color: "blue"})}>Olá, Mundo!</h2>
        <Msg/>
      <div>
      </div>
    </div>
  );
}

export default App;
```

src > components > msg.js > ...

```jsx
import React, { useState } from "react";
import Box from "./FirstComponents.js";
import { AnimatePresence } from "framer-motion";

function Msg(){
    const mensagem = ["Home", "Serviços","Contato"];
    const [index, setIndex] = useState(0);
    const [mensagemAtual, setMensagem] = useState([]);
    const showBox = () =>{
        if(index < mensagem.length){
            setMensagem(prev => [...prev, mensagem[index]]);
            setIndex(prev => prev + 1);
        }else{
            setIndex(0);
            setMensagem([])
        }
    }
    console.log(mensagemAtual)
    return(
        <div>
            <button onClick={showBox}>Conhecer</button>
            <AnimatePresence>
                {mensagemAtual.map((msg, i) => (
                    <Box key={i} mensagem={msg}/>
                ))}
            </AnimatePresence>
        </div>
    )
};
export default Msg;

//Estamos adicionando novos itens no array, onde será mostrado dentro do .map
```

src > components > showMessage.js > default

```jsx
function Show({msg}){
    return(
        <button onClick={msg}>Aperta aqui</button>
    );
}
export default Show;
```

src > components > FirstComponents.js > ...

```jsx
import { motion } from "framer-motion";

const Box = ({mensagem}) => {
  return (
    <motion.div
      key={mensagem}
      initial={{ opacity: 0, y: 50 }} // começa invisível e 50px abaixo
      whileInView={{ opacity: 1, y: 0 }} // quando aparece na tela
      transition={{ duration: 1 }}
      viewport={{ once: true }} // anima só uma vez
      exit={{duration: 0.6}}
      style={{
        width: '200px',
        height: '200px',
        backgroundColor: '#4f46e5',
        margin: '50px auto',
        borderRadius: '8px'
      }}
    >
      <p style={{ color: '#fff', textAlign: 'center', paddingTop: '80px' }}>{mensagem}</p>
    </motion.div>
  );
};

export default Box;
```

# useEffect

Vamos importar uma outra função do react.
Bom, ela funciona da seguinte maneira, ela recebe coisas do servidor e renderiza

```
const url = "http://localhost:3000/product/"
useEffect(() =>{
  async function getter(){
    await fetch(url)
    .then(res => res.json())
    .then(data => setDados(data))
    .catch(e => console.log("erro", e))
  }
  getter()
}, [])

return (
  <div>
    <h2>{name}</h2>
    <input type="text" placeholder="titulo" onChange={handle}></input>
    <div>
      {dados.map(post => (
        <li key={post.id}>{post.nome}<br/>{post.preco}</li>
      ))}
    </div>
    <div>
    </div>
  </div>
);
}
```

O array de dependência (como [ ]) no useEffect serve para:

✅ **Executar o código do useEffect toda vez que aquela entidade (variável, estado, prop...) mudar.**

**Exemplos de uso comum:**

Buscar dados de uma API

Adicionar event listeners (ex: scroll, resize)

Atualizar o título da aba (document.title)

Limpar um timer ou um intervalo

Sincronizar com serviços externos

Vamos ver um exemplo:

```
useEffect(() => {
  async function getter() {
    await fetch(url)
      .then(res => res.json())
      .then(data => setDados(data))
      .catch(e => console.log("erro", e))
  }
  getter()
}, [])
```

Agora vamos criar uma função que vai adicionar novos produtos na nossa db

```
const [nome, setProduto] = useState('');
const [preco, setPreco] = useState('')

const handleSubmit = async (e) => {
  e.preventDefault();
  const product = {
    nome,
    preco
  }
  const res = await fetch(url, {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify(product)
  });
  const data = await res.json()
  setDados(prev => [...prev, data]);
  console.log(product)
}
```

Como estamos fazendo uma requisição via POST, no caso, envio de dados para o servidor. Vamos configurar o header informando o tipo do conteúdo

```
<div>
  {data && data.map(e => (
    <li style={{listStyle: "none"}} key={e.id}><p>{e.nome}</p><p>{e.preco}</p></li>
  ))}
</div>
```

Depois é só trabalhar com as informações.

“Podemos usar essa função toda como hook ?” – Sim, iremos fazer isso agora

src > hook > showUser.js > ...

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

function useAccess(url) {
  const [data, setData] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const res = await fetch(url);
        console.log(res)
        const json = await res.json();
        setData(json);
      } catch (e) {
        console.log("Não foi possivel obter informação: ", e);
      }
    };
    fetchData();
  }, [url]);

  return {data};
}

export default useAccess;
```

```
const url = "http://localhost:3000/product/"
const {data} = useAccess(url);
console.log(data);
```

Vamos adicionar só um tratamento, caso perca a conexão de dados

```
{data ? (data.map(e => (
  <li style={{listStyle: "none"}} key={e.id}><p>{e.nome}</p><p>{e.preco}</p></li>
))):(<p>Carregando...</p>)}
```

Adicionando um loading…

```
const[loading, setLoading] = useState(false);

const httpConfig = (data, method) => {
  if(method === 'POST') {
    setConfig({
      method,
      headers:{
        "Content-Type": "application/json"
      },
      body: JSON.stringify(data)
    });
    setMethod(method);
  }
}

useEffect(() => {
  const fetchData = async () => {
    try {
      setLoading(true);
      const res = await fetch(url);
      const json = await res.json();
      setData(json);
      setLoading(false);
    } catch (e) {
      console.log("Não foi possível obter informação: ", e);
    }
  };
```

**return {data, httpConfig, loading};**

```
{loading && <button disabled >Aguarde</button>}
{!loading && <button type='submit'>Adicionar</button>}
{/* Loading irá fazer que os butões apareçam quando houver troca de valores
    Como existe uma função no hook q chama o fetch novamente quando acontece o POST
    ele muda os valores nesse meio tempo
*/}
```