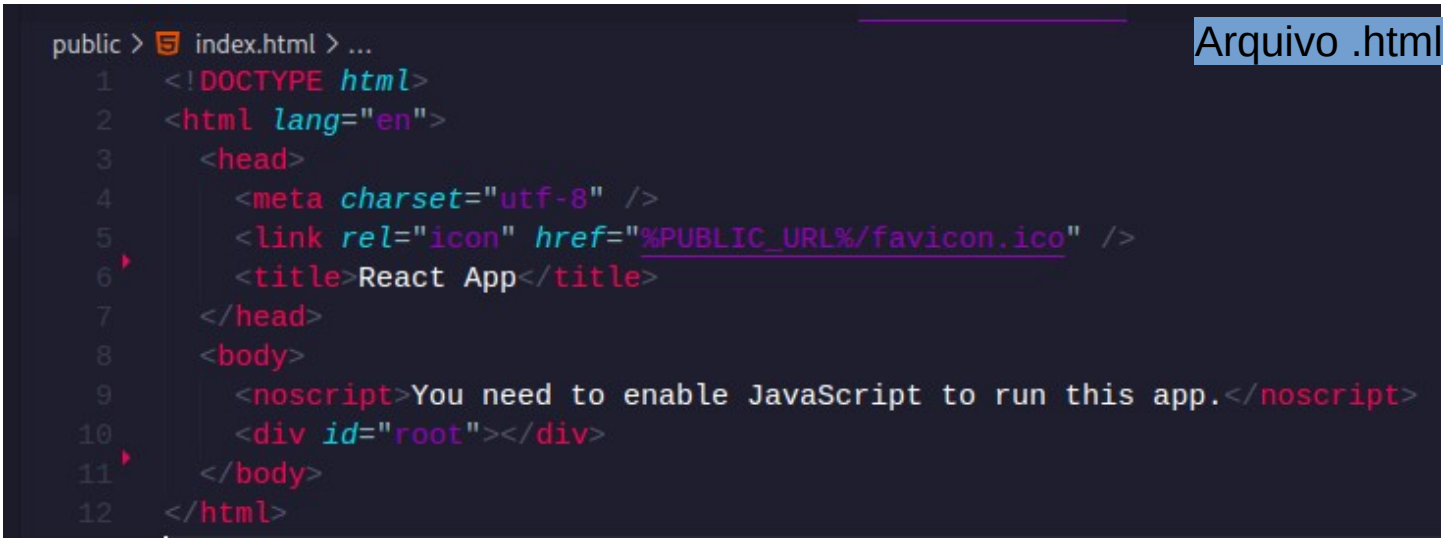
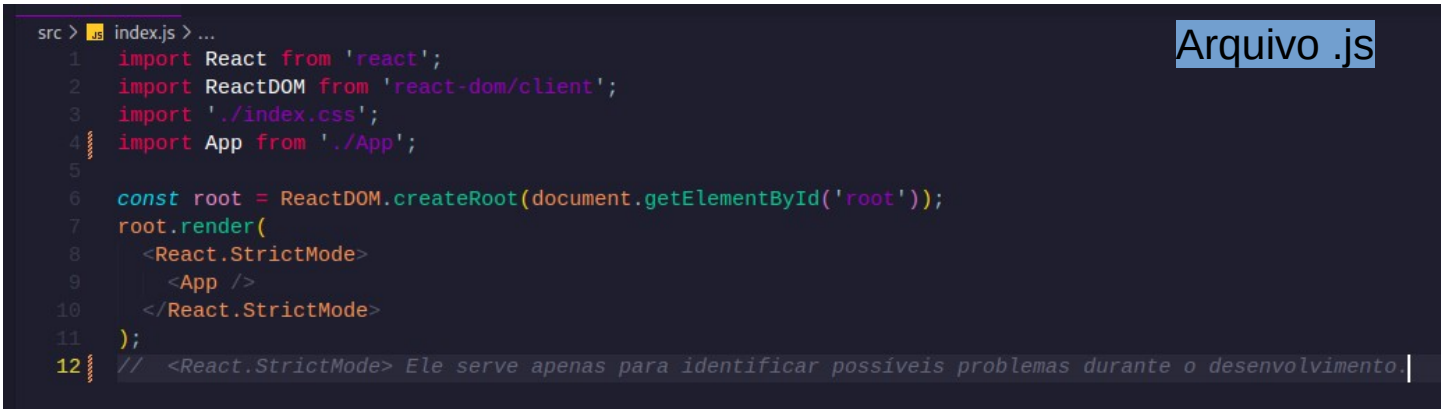
React

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.

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