

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

```
public > index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6     <title>React App</title>
7   </head>
8   <body>
9     <noscript>You need to enable JavaScript to run this app.</noscript>
10    <div id="root"></div>
11  </body>
12 </html>
```

Arquivo .html

```
src > index.js > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5
6 const root = ReactDOM.createRoot(document.getElementById('root'));
7 root.render(
8   <React.StrictMode>
9     <App />
10   </React.StrictMode>
11 );
12 // <React.StrictMode> Ele serve apenas para identificar possíveis problemas durante o desenvolvimento.
```

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 `<div id="root"></div>` 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 > .JS App.js > ...
1 //Componentes / react
2 import First from './components/FirstComponents.js';
3 //Style / CSS
4 import './App.css';
5 import foto from './assets/logo192.png';
6
7
8 function App() {
9   return (
10     <div>
11       <h1>Hello, word!</h1>
12       <First/>
13       <img src={foto} alt="foto"></img>
14     </div>
15   );
16 }
17
18 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 > .JS FirstComponents.js > First
1 const First = () => {
2   const nome = "jean"
3   return (
4     <div>
5       <div>
6         <h2>olá, {nome}</h2>
7         <button>Aperte aqui</button>
8       </div>
9       <div className="teste">
10
11       </div>
12     </div>
13   );
14 };
15 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 implementar o componente, 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/>
    </div>
  );
}
```

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 './App.css';
import foto from './assets/logo192.png';

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

```
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>
  );
}
```

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

const deleteId = () => {
  const randomId = Math.floor(Math.random() * 4);
  setUser(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 > JS showUser.js > ...
1  import { useState, useEffect } from "react";
2
3  function useAccess(url) {
4    const [data, setData] = useState(null);
5
6    useEffect(() => {
7      const fetchData = async () => {
8        try {
9          const res = await fetch(url);
10         console.log(res)
11         const json = await res.json();
12         setData(json);
13       } catch (e) {
14         console.log("Não foi possível obter informação: ", e);
15       }
16     };
17     fetchData();
18   }, [url]);
19
20   return {data};
21 }
22
23 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 `<p>Carregando...</p>`.

OBS: os componentes precisam começar com a letra maiúscula

```
import ShowCarro from './components/carros.js';
```

Condições

No react, as condições podem ser feitas de 3 maneiras:

Condição

```
3  const First = () => {
4    const [x] = useState(1);
5    const [y] = useState(true)
6    return (
7      <div>
8        <div>
9          {x > 6 ? (<p>Maior que 10</p>):(<p>Não é maior que 10</p>)}
10         {y && <p>Valor verdadeiro</p>}
11       </div>
12     </div>
13   );
14 };
15 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 variável 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

```
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


```

//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 reutilizado

OBS: quando usamos um componente repetidamente, 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

```
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.

```
<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

```
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.

```
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>
  );
}
```

```
const handle = (e) => {
  if (e.target.value.trim() === "") {
    setName("Digite 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;

```
src > components > JS msg.js > [⌘] default
1  function msg(props){
2      return(
3          <p>A mensagem é: {props.nota}</p>
4      )
5  };
6  export default msg;
```

```
src > components > JS showMessage.js > [⌘] default
1  function Show({msg}){
2      return(
3          <button onClick={() => msg('Olá')}>Aperta aqui</button>
4      );
5  }
6  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 > JS App.js > ...
1  //Componentes / react
2  import { useState } from 'react';
3  import Msg from './components/msg.js';
4  //Style / CSS
5  import './App.css';
6
7
8  function App() {
9
10     const cor = false;
11
12     return (
13         <div>
14             <h2 style={cor ? ({color: "red"}) : ({color: "blue"})}>Olá, Mundo!</h2>
15             <Msg/>
16             <div>
17             </div>
18         </div>
19     );
20 }
21
22 export default App;
```



```

src > components > msg.js > ...
1  import React, { useState } from "react";
2  import Box from "../FirstComponents.js";
3  import { AnimatePresence } from "framer-motion";
4
5  function Msg(){
6      const mensagem = ["Home", "Serviços", "Contato"];
7      const [index, setIndex] = useState(0);
8      const [mensagemAtual, setMensagem] = useState([]);
9      const showBox = () =>{
10         if(index < mensagem.length){
11             setMensagem(prev => [...prev, mensagem[index]]);
12             setIndex(prev => prev + 1);
13         }else{
14             setIndex(0);
15             setMensagem([])
16         }
17     }
18     console.log(mensagemAtual)
19     return(
20         <div>
21             <button onClick={showBox}>Conhecer</button>
22             <AnimatePresence>
23                 {mensagemAtual.map((msg, i) => (
24                     <Box key={i} mensagem={msg}/>
25                 ))}
26             </AnimatePresence>
27         </div>
28     )
29 };
30 export default Msg;
31
32 //Estamos adicionando novos itens no array, onde será mostrado dentro do .map

```

```

src > components > showMessage.js > default
1  function Show({msg}){
2      return(
3          <button onClick={msg}>Aperta aqui</button>
4      );
5  }
6  export default Show;

```

```

src > components > FirstComponents.js > ...
1  import { motion } from "framer-motion";
2
3  const Box = ({mensagem}) => {
4      return (
5          <motion.div
6              key={mensagem}
7              initial={{ opacity: 0, y: 50 }} // começa invisível e 50px abaixo
8              whileInView={{ opacity: 1, y: 0 }} // quando aparece na tela
9              transition={{ duration: 1 }}
10             viewport={{ once: true }} // anima só uma vez
11             exit={{duration: 0.6}}
12             style={{
13                 width: '200px',
14                 height: '200px',
15                 backgroundColor: '#4f46e5',
16                 margin: '50px auto',
17                 borderRadius: '8px'
18             }}
19         >
20             <p style={{ color: '#fff', textAlign: 'center', paddingTop: '80px' }}>{mensagem}</p>
21         </motion.div>
22     );
23 };
24
25 export default Box;

```

useEffect

Vamos importar uma outra função do react.

Bom, ela funciona da seguinte maneira, ela recebe coisas do servidor e renderiza. Ele recebe uma função de callback

```
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 > JS showUser.js > ...
1  import { useState, useEffect } from "react";
2
3  function useAccess(url) {
4    const [data, setData] = useState(null);
5
6    useEffect(() => {
7      const fetchData = async () => {
8        try {
9          const res = await fetch(url);
10         console.log(res)
11         const json = await res.json();
12         setData(json);
13       } catch (e) {
14         console.log("Não foi possível obter informação: ", e);
15       }
16     };
17     fetchData();
18   }, [url]);
19
20   return {data};
21 }
22
23 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 [data, setData] = useState(false); // vai servir pra  
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 botões 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  
*/}
```


Rotas

Vamos criar rotas para exibir paginas em um determinado ponto da pagina sem recargar-la. Iremos usar 3 funções do React-router-dom

BrowserRouter – Responsável por habilitar o roteamento na sua aplicação React.

Ele envolve toda a aplicação (ou a parte que vai usar rotas).

Usa a história do navegador (history API) para controlar a navegação sem recarregar a página.

Routes – Dentro dela que é definido as rotas

Route – É onde você define as rotas

```
import { useState } from "react";
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'
//componentes
import Navbar from "../components/navbar";
//Pages
import Home from "../pages/Home";
import About from "../pages/About";
import Product from "../pages/produtos";
import NotFound from "../pages/NotFound";
//css
import './App.css';

export default function App () {

  return(
    <div>
      <div className="div-titulo">
        <h2 className="myName"><span></span>Jean Carvalho</h2>
        <h1 className="titulo">Dev web</h1>
      </div>
      <BrowserRouter>
        <Navbar/>
        <Routes>
          <Route path="/" element={<NotFound/>}/>
          <Route path="/" element={<Home/>}/>
          <Route path="/about/:id?" element={<About/>}/>
          <Route path="/product/:nome?" element={<Product/>}/>
          <Route path="/company" element={<Navigate to="/about"/>}/>
        </Routes>
      </BrowserRouter>
      <footer className="footer-box">
        <p>Thanks</p>
      </footer>
    </div>
  )
}
```

Esse * é para identificar quando alguém digitar qualquer coisa, cai pro 404

```
export default function NotFound() {
  return(
    <div>
      <p>Not Found</p>
    </div>
  );
};
```

Componentes fora do BrowserRouter nunca irão mudar, sempre estarão lá, então geralmente você irá colocar o footer que nunca irá mudar, o header e outros...

“E o que é esse navbar ?”

Ele é um componente do react que criamos e que importa o Link e NavLink Componentes importantes no qual fazem o redirecionamento

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

export default function Navbar(){
  return(
    <nav className=''>
      <ul className='ul-nav'>
        {/*<li><Link to="/" className='link-btn'>Home</Link></li>
        <li><Link to="/about" className='link-btn'>Sobre</Link></li>
        <li><Link to="/Product/1" className='link-btn'>Produtos</Link></li>*/}
        <li><NavLink to="/" className='link-btn'>Home</NavLink></li>
        <li><NavLink to="/about" className='link-btn'>Sobre</NavLink></li>
        <li><NavLink to="/Product" className='link-btn'>Produtos</NavLink></li>
      </ul>
    </nav>
  );
}
```

Ele que faz a ancoragem, ou seja, como se fosse o <a>.

A diferença entre o <Link> e o <NavLink> é quando você usa o NavLink, Ele adiciona automaticamente um “active” na classe. Então, a estilização da Página irá ocorrer junto ao clicar .

```
<li>
  <a class="link-btn active" href="/" data-discover="true" aria-current="page">Home</a>
</li>
<li>
  <a class="link-btn" href="/about" data-discover="true">Sobre</a> -- $0
</li>
<li></li>
```

html body div#root div nav ul.ul-nav li a.link-btn

```
.active{
  display: block;
  font-size: 0.8em;
  text-decoration: none;
  border: 2px solid #18b17e;
  border-radius: 50px;
  box-shadow: 0px 0px 8px 3px rgba(8, 206, 157, 0.432);
  color: white;
  padding: 0.6em 0.8em;
  text-transform: uppercase;
}
```

Caixa de busca

Para isso, iremos importar e usar o useParams, Navigate e o useSearchParams. Vamos passo a passo:

```
<Navbar/>
<Routes>
  <Route path="/" element={<NotFound/>}/>
  <Route path="/" element={<Home/>}/>
  <Route path="/about" element={<About/>}/>
  <Route path="/product/:nome?" element={<Product/>}/>
  <Route path="/company" element={<Navigate to="/about"/>}/>
</Routes>
</BrowserRouter>
<Footer className="Footer" />
```

```
src > pages > JS produtos.js > Product
1 import { useParams, useNavigate, useSearchParams } from "react-router-dom";
2 import { motion } from "framer-motion";
3 import useAccess from "../hook/showUser";
4 import { useState } from "react";
5
6 export default function Product() {
7   const [searchParams] = useSearchParams();
8   const valor = searchParams.get('q'); // acessa url
9   const url = valor ? `http://localhost:3000/product?nome=${valor}` : "http://localhost:3000/product"
10  const { data } = useAccess(url)
11  const navigate = useNavigate()
12  const [query, setQuery] = useState()
13  function searchFunction(e){
14    e.preventDefault()
15    navigate("/product?q=" + query)
16  }
17  return (
18    <div>
19      <motion.form
20        initial={{opacity: 0, y: -10}}
21        whileInView={{opacity: 1, y:0}}
22        transition={{duration: 0.5, ease: "easeInOut"}}
23        viewport={{once:true}}
24
25        onSubmit={searchFunction}
26      >
27
28        <input type="text" onChange={(e) => setQuery(e.target.value)}></input>
29        <button type="submit">Buscar</button>
30      </motion.form>
31      <motion.div
32        initial={{ opacity: 0, x: -10 }}
33        whileInView={{ opacity: 1, x: 0 }}
34        transition={{ duration: 0.5, ease: "easeOut" }}
35        viewport={{ once: true }}
36      >
37        {data && data.map(d => (
38          <li key={d.id}>{d.nome} - Preço: {d.preco}</li>
39        ))}
40      </motion.div>
41    </div>
42  )
43 }
```

Aqui inserimos um input que irá fazer o submit com a palavra-chave, exemplo: teclado. Fazendo uma destruturação em useSearchParams, vamos pegar esse valor e usar a função get() dele para obter o valor da pesquisa. Com isso, iremos formatar a string da url caso tenha esse valor, se não tiver, ele busca o url normal contendo todos os itens.

“Mas para que serve o `navigate` ?”

Ele basicamente irá mudar sem recargar a página, entendeu? Então quando fazemos a busca, ele irá retornar com os valores, mas sem recargar a página. E pronto, você já tem a sua página de busca

“E o que é aquele `useParams`?”

Bom ele serve para você pegar o id da página quando é passado de forma direta, e não sendo uma query. Exemplo:

Se fosse algo como `/product/123`, aí sim você usaria:

```
import { useParams } from "react-router-dom";

export default function Produto() {
  const { id } = useParams();

  return <h1>Produto com ID: {id}</h1>;
}
```



The image shows a browser window at `localhost:3001/about/21312` displaying a page titled "About" with the text "O id é: 21312" and "Thanks". Overlaid on the browser is a code editor showing the `About` component implementation. The code imports `motion` from `framer-motion` and `useParams` from `react-router-dom`. It uses `useParams` to get the `id` from the URL and passes it to a `motion.div` component. The `motion.div` has an `initial` state of `{opacity: 0, y: -20}`, a `whileInView` state of `{opacity: 1, y: 0}`, a `transition` of `{duration: 0.5, ease: "easeOut"}`, a `viewport` of `{once: true}`, and an `exit` state of `{opacity: 0, y: -10}`. The component renders an `h2` tag with the text "About" and a `p` tag with the text "O id é: {id}".

```
import { motion } from "framer-motion"
import { useParams } from "react-router-dom"
export default function About(){
  const {id} = useParams();
  return(
    <motion.div
      initial={{opacity: 0, y: -20}}
      whileInView={{opacity: 1, y: 0}}
      transition={{duration: 0.5, ease: "easeOut"}}
      viewport={{once: true}}
      exit={{opacity: 0, y: -10}}
    >
      <h2>About</h2>
      <p>O id é: {id}</p>
    </motion.div>
  )
}
```


Outra forma de usar Navigate

Por mais questão de SEO, quando alguém for buscar a página, por exemplo: Seu site/company, ele irá redirecionar:

```
<Route path="/company" element={<Navigate to="/about"/>}/>
```

Importa o navigate e usa como element do route

useReducer

Semelhante ao useState, porém, ele é o ideal quando você quer atualizar os estados em diferentes condições.

```
6 function reducer(state, action){
7   switch(action.type){
8     case "add":
9       return {count: state.count + 1};
10    default:
11      return state;
12  }
13 }
14
15 export default function About(){
16   const [ state, dispatch ] = useReducer(reducer, {count: 0})
17   return(
18     <motion.div
19       initial={{opacity: 0, y: -20}}
20       whileInView={{opacity: 1, y: 0}}
21       transition={{duration: 0.5, ease: "easeOut"}}
22       viewport={{once: true}}
23       exit={{opacity: 0, y: -10}}
24     >
25       <h2>About</h2>
26       <p onDoubleClick={() => dispatch({type: "add"})}>Valor é: {state.count}</p>
27       <button onClick={() => dispatch({type: "add"})}>alterar</button>
28     </motion.div>
29   )
30 }
```

Criamos uma função reducer que está esperando receber dois argumentos do tipo objetos. Pode parecer confuso, então o hook do useReducer, pode parecer confuso, mas já iremos entender

UseReducer(função, estado inicial) = função pura e o estado inicial que ele state, no caso do exemplo irá começar.

O que acontece: o react passa os argumentos automaticamente para a função pura, no caso, primeiro ele passou o estado inicial – {count: 0} e quando atribuímos os dispatch (nome que geralmente usamos em useReducer) em onClick, ele irá enviar o type, no caso, o action. Isso tudo de forma interna

Extra: quando usamos o onDoubleClick, e você aperta duas vezes em um item, ele executa a função

Contextos

Você pode passar dados para todos os componentes filhos criando contexto

```
src > context > contador.js > CounterProvider
1  import { createContext, useState } from "react";
2  import useAccess from "../hook/showUser";
3
4  export const CounterContext = createContext() //serve para acessar os dados
5
6
7  export function CounterProvider({children}){
8    const [counter, setCounter] = useState(5)
9
10    return(
11      <CounterContext.Provider value={{counter, setCounter}}> /*serve para distribuir os dados para os componentes filhos*/
12        {children}
13      </CounterContext.Provider>
14    )
15  }
```

Você inicia a função do createContext()

Nota-se que criamos um state que irá conter informações que iremos repassar para os componentes filhos

Usamos nossa variavel onde iniciamos o createContext()

Quando você cria um contexto com createContext() no React, esse contexto vem com um componente Provider. Esse .Provider é usado para fornecer dados (valores) para os componentes que estão abaixo dele na árvore de componentes (ou seja, os filhos).

🔗 Sem o .Provider?

Sem o .Provider, nenhum componente filho conseguiria acessar o valor do contexto. O useContext(CounterContext) retornaria undefined ou o valor padrão que você passou ao createContext().

O value={{ counter, setCounter }} é o valor que qualquer componente pode acessar usando useContext(CounterContext).

E o Children são os componentes filhos que irão herdar