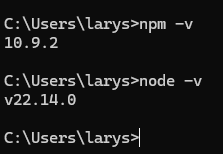
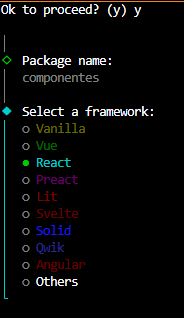
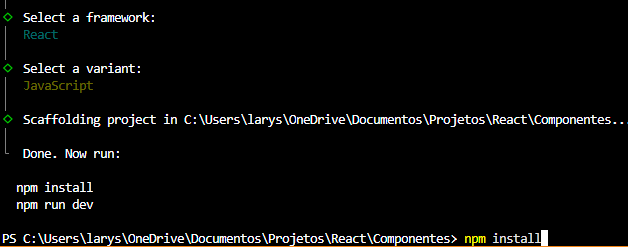
# **React**

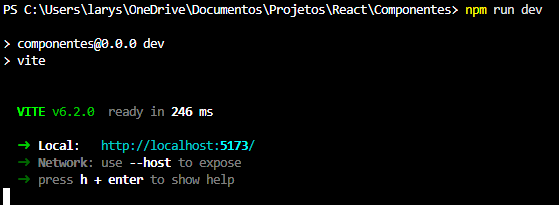
O React é uma biblioteca usada para criar interfaces de usuário interativas.

1. Instalar o Node.Js <https://nodejs.org/en/download>
2. Fazer o teste, no prompt comando digitando ‘npm -v’ depois ‘node -v’
3. Vamos utilizar o VsCode
4. Instalar a extensão: *ES7+ React/Redux/React-Native snippets*
5. Documentação do React <https://react.dev/>
6. Beleza, no terminal do VsCode digita: ‘npx create-vite.’



1. Solicitará em seguida para que vc digite o nome do projeto que deseja criar, em seguida selecione a biblioteca, no caso React e depois escolha a linguagem, no caso JavaScript (é bem intuitivo)
2. E digita ‘npm install’



1. Para mostrar o projeto no chrome digita ‘npm run dev’

**Para fazer comentário**:

Com duas barras 

E dentro do parênteses é Ctrl + :

1. Feito a estrutura de pastas do react:



* *node\_modules*: todos os pacotes para o React rodar, que é um monte de coisas instalados
* *src*: é a pasta que a gente vai mexer, onde estar os arquivos iniciais:
  + - * + *App.jsx*: o que aparece no chrome, o .html
        + *main.jsx*: no HTML, cai nesse arquivo e roda toda a aplicação, chamando o app.jsx
* .*gitignore*: ele ignora o que a gente não quer que salve no repositório, para não deixar pesado.
* *eslint.config.js*: é a configuração do link, que é o organizador de código.
* *package.json*: é onde tem todas as dependências do nosso projeto.

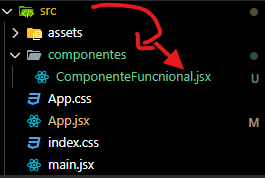
## Vamos começar em componentes.

O que é componentes?

É uma parte essencial do React, ele é basicamente uma parte da aplicação dividia do principal código dela. Praticamente são vários elementos usando um mesmo arquivo.

No React, todas as aplicações basicamente vão ser orientadas a componentes

Fazendo o primeiro componente:

No *src*, criar uma pasta, e um arquivo jsx

Com letra MAIÚSCULA

Esse arquivo vai ser o Componente FILHO

O App.jsx é o componente PAI

ComponenteFuncional.jsx:

//Componente FILHO

//Criar uma função

function ComponenteFuncional(){

    //retorna um objeto com jsx

    return( //parenteses

        <div> <h1>Testando o novo Componente</h1> </div>

    )

}

//Sempre temos que exportar

export default ComponenteFuncional;

App.jsx:

//Componente PAI

import './App.css'

//Importar o componente FILHO

import ComponenteFuncional from './componentes/ComponenteFuncnional';



function App() {

  return (

    <>

    <h1>Heloo Word</h1>

{/\* utilizando o componente importado no jsx \*/}

    <ComponenteFuncional></ComponenteFuncional>

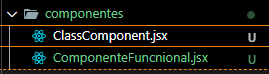
    </>

  )

}

//Sempre temos que exportar

export default App;



Componentes de Classe: Criar mais um arquivo primeiro ->

//Componente de classe FILHO

import React from "react";

//Criar uma classe, estender uma classe PAI

class ClassComponent extends React.Component {

    // renderizar os elementos

    render() {

        return (

            <div>

                <h2>Componente de Classe</h2>

            </div>

        )

    }

}

//Exportar

export default ClassComponent;

Volta no App.jsx para importar e aparecer no Chrome:

//Componente PAI

import './App.css'

//Importar o componente FILHO

import ComponenteFuncional from './componentes/ComponenteFuncnional';

import ClassComponent from './componentes/ClassComponent';

function App() {

  return (

    <>

    <h1>Helloo Word</h1>

    {/\* utilizando o componente importado no jsx \*/}

    <ComponenteFuncional></ComponenteFuncional>

    <ClassComponent></ClassComponent>

    </>

  )

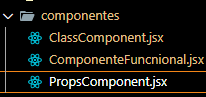
}

//Sempre temos que exportar

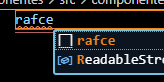
export default App;

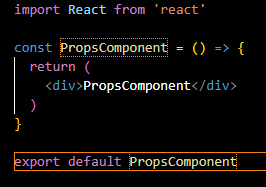
### Props

Props é uma palavra-chave do React que significa "propriedades". É um mecanismo que permite passar dados de um componente pai para um componente filho.



Criar mais um arquivo no Componente ->

Colocando rafce, ele cria um componente pronto com mais facilidade



PropsComponents.jsx:

import React from 'react';

const PropsComponent = ({nome, idade}) => {

  return (

    <div>

        <h3>Ola {nome}</h3>

        <h4>Tenho {idade} anos.</h4>

    </div>

  );

};

export default PropsComponent;

Importando no App.jsx:

import PropsComponent from './componentes/PropsComponent';

<>

{/\* componente com dados \*/}

<PropsComponent nome = 'Laryssa' idade = {26} ></PropsComponent>

</>

## Novo projeto: 2\_JSX

1. Fazer o mesmo processo do inicio
2. No *src*, criar uma pasta, e um arquivo jsx
3. Esse arquivo vai ser o Componente FILHO
4. Coloca ‘rafce’, para ele criar um componente pronto

//Componente FILHO

import React from 'react'

import '../App.css'

const JSXExemplo = () => {

    //Podemos ter Variaveis

    const userName = 'Carlos';

    const user = {

        name: "Ana",

        lastName: "Souza",

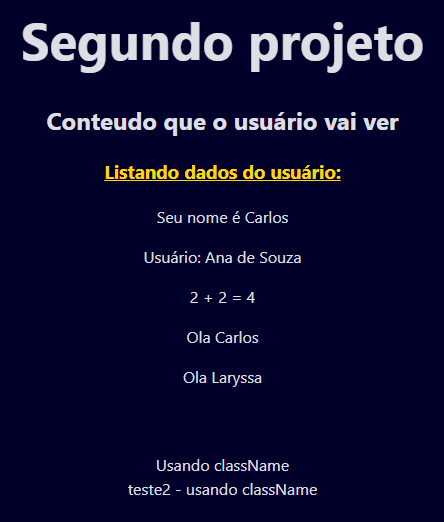
    };

    //Podemos ter tbm Funçoes

    function getGreeting(name2){  //pegar saldação

        return `Ola ${name2}`; //usando crase ` para template literals

    }

    //Renderização condicional

    const userIsLoggedin = true;

    //Expressao lógica

    const userRole = 'admin';

    //Renderização de listas

    const users = [ //fazer um array de usuarios

        { id: 1, name: "Joao" },

        { id: 2, name: "Maria" },

        { id: 3, name: "Pedro" },

    ];

  return (

    <div>

        {/\* BASICO \*/}

        <h2>Conteudo que o usuário vai ver</h2>

        <h3>Listando dados do usuário:</h3>

        <p>Seu nome é {userName}</p>

        <p>Usuário: {user.name} de {user.lastName}</p>

        <p>2 + 2 = {2 + 2}</p>  {/\* Tudo que esta entre chaves, vai executar em JS \*/}

        <p>{getGreeting(userName)}</p>

        <p>{getGreeting("Laryssa")}</p>

        <br /><br />

        {/\* DIFERENÇAS DO HTML \*/}

        <div className='alguma coisa'>Usando className</div>

        <div className='teste'>teste2 - usando className</div>

        {/\* no HTML é apenas o 'class'

            no JSX tem que ser 'classNAme' \*/}

        <br /><br />

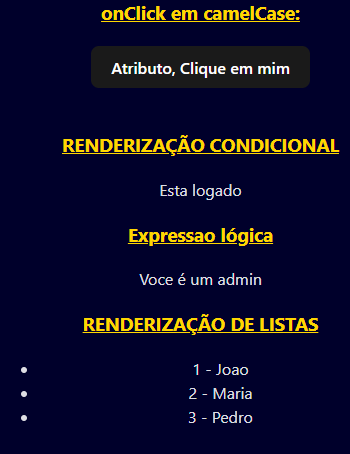
        <h3>onClick em camelCase:</h3>

        <button onClick={() => alert("Teste")}>Atributo, Clique em mim</button>

        <br /><br />

        <h3>RENDERIZAÇÃO CONDICIONAL</h3>

        {userIsLoggedin ? (<p>Esta logado</p>) : (<p>NAO esta logado</p>)}

        <h3>Expressao lógica</h3>

        <p>{userRole === "admin" && "Voce é um admin"}</p>

        <h3>RENDERIZAÇÃO DE LISTAS</h3>

        <div>

            <ul>

                {users.map((user) => (

                    <li key={user.id}> {/\* cada item precisa

de um identificado unico \*/}

                        {user.id} - {user.name}

                    </li>

                ))}

            </ul>

        </div>

    </div>

  )

}

export default JSXExemplo

## Gerenciamento de dados

O useState é um hook do React que permite adicionar variáveis de estado a componentes de função. É uma ferramenta que ajuda a gerenciar o estado de componentes funcionais.

//Componente FILHO

import {useState} from "react";

//useState = os hooks, gerencia os estados

    const Gerenciamento = () => {

        //crio uma variavel de consulta, e uma alteração, inicio o hook

        const [count, setCount] = useState(0); //sintaxe base

        const Increment = () => { //um evento

            setCount((prevCount) => prevCount + 1)

            //setCount(count + 1)

            console.log(count)

        }

        //Sempre que vamos alterar o valor de algo => useState

        //Se é somente leitura =: var, state

        const [user, setUser] = useState({

            name: "Ana",

            age: 25,

            hobbies: ["Leitura", "Programação"]

        })

        const updateUserAge = () => {

            setUser((prevUser) => ({

                ...prevUser,

                age: prevUser.age + 1,

            }))

        }

        return (

            <div>

                <h2>Contador:</h2>

                <p>Voce clicou {count} vezes.</p>

                <button onClick={Increment}>Incrementar</button>

                <br />

                <p>

                    Nome: {user.name} e idade: {user.age}

                </p>

                <br />

                <button onClick={updateUserAge}>Incrementar idade</button>

            </div>

        )

    };

export default Gerenciamento;

## Eventos

import {useState} from 'react'

const EventHandlingExample = () => {

  //fazer uma função com um evento

  const handleClick = () => {

      alert ('Testando o evento handleClick')

  }

const handleGreet = (name) => {

    alert(`Ola ${name}`)

  }

 //pegar os valores do formulário

  const[name, setName] = useState("");

  const handleSubmit = (e) =>{

    e.preventDefault(); //para nao carregar a pagina

    alert (`Enviado ${name}`)

  }

  return (

    <div>

        <h2>EventHandlingExample</h2>

        <button onClick={() => alert("oi")}>Clique aqui</button> {/\*evento inline \*/}

        <button onClick={handleClick}>Clique aqui 2</button>

<br />

        <button onClick={() => handleGreet("Ana")}>Dizer oi para Ana</button>

        <button onClick={() => handleGreet("Pedro")}>Dizer oi para Pedro</button>

<h3>Formulário:</h3>

        <form onSubmit={handleSubmit}>

            <input

              type="text"

              value={name} //state de leitura

              onChange={(e) => setName(e.target.value)} //alteração de conteudo

              placeholder='Digite seu nome'

            />

            <input type="submit" value="Enviar"/>

        </form>

    </div>

  )

}

export default EventHandlingExample

## useEffect

O useEffect é um hook do React que permite lidar com efeitos colaterais em componentes funcionais. É uma ferramenta fundamental para criar interfaces de usuário responsivas e atualizadas.

import {useState, useEffect} from 'react'

//O useEffect é um hook do React que permite lidar com efeitos colaterais em componentes funcionais. É uma ferramenta fundamental para criar interfaces de usuário responsivas e atualizadas.

const UseEffect = () => {

  //sem dependencias

  useEffect(() => { //função anonima

    console.log("Rodou o Effect 1")

  }) //ele roda sempre q algo disperar qnd reenderizar o react

  //com dependencias vazias / ou array dep. vazio

  useEffect(() => { //função anonima

    console.log('Rodou o Effect 2')

  }, []) //só roda uma vez, só qnd carrega a pagina

  //useState: ativa a reenderização

  const [count, setCount] = useState(0)

  //com dependencias

  useEffect(() => { //função anonima

    console.log('Rodou o Effect 3')

  }, [count]) //ele roda só qnd tem uma mudança no estado

  return (

    <div>

        <h2>UseEffect</h2>

        <p>{count}</p>

        <button onClick={() => setCount(count + 1)}>Aumentar contagem</button>

    </div>

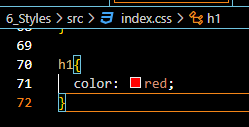
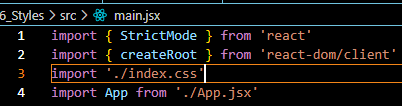
  )

}

export default UseEffect

## Styles

1. Primeiro jeito é pelo index.css (global)

Ele esta lincado no main.jsx (arquivo principal da aplicação)

1. Segundo jeito é o estilo Inline

import React from 'react'

const StylesExemplo = () => {

    //inline

    const inlineStyle = {

        color: 'blue',

        fontsixe: '20px'

    }

  return (

    <div>

        <h2>stylesExemplo</h2>

        <h3 style={inlineStyle}>Estilo Inline</h3>

    </div>

  )

}

export default StylesExemplo

1. Terceira é ter um arquivo em css (com o mesmo nome do jsx)

import React from 'react'

//arquivo css

import './StylesExemplo.css'

const StylesExemplo = () => {

  return (

    <div>

       {/\* arquivo de estilos, com class \*/}

        <p className="text">Estilo de arquivo com class</p>

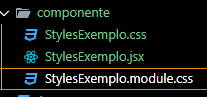
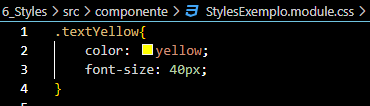
    </div>

  )

}

export default StylesExemplo

O ruim, desse jeito, o estilo pode vazar para outros componentes da pasta, não é eficaz

1. Quarto jeito é estilo em módulos
   1. Criar um arquivo dentro de componente
   2. Com o nome ‘StylesExemplo.module.css’
   3. Vamos usar classes
   4. Importando como objeto no ‘StyleExemplo.jsx’

import React from 'react'

//arquivo css

import './StylesExemplo.css'

//arquivo modulo, importa como objeto

import style from './StylesExemplo.module.css'

const StylesExemplo = () => {

  return (

    <div>

        {/\* arquivo de estilo, com css modules \*/}

        <p className={style.textYellow}>Meu css modules</p>

    </div>

  )

}

export default StylesExemplo

## Você pode adicionar styles dentro do function:

function Footer() {

  const footerStyle = {

    textAlign: "center",

    color: "white",

    backgroundColor: "#181717",

    padding: "14px",

  };

  return <h1 style={footerStyle}>&copy;2025 Laryssa Tortato Gomes da Silva</h1>;

}

export default Footer;

## Esta sendo usado muito o <https://tailwindcss.com/>

Um framework CSS voltado para utilidades, repleto de classes como ‘flex , pt-4 , text-center e rotate-90’ que podem ser compostas para criar qualquer design, diretamente na sua marcação.

Hoje ele vem se tornando cada vez mais o padrão.

No terminal coloque:

* npm i -D tailwindcss@3.4.10 postcss@8.4.41 [autoprefixer@10.4.20](mailto:autoprefixer@10.4.20)
* npm i -D tailwindcss postcss autoprefixer

Gere o arquivo tailwind.config.js com:

* npx tailwindcss init -p

Adicionar no ‘tailwind.config.js’:

* content: ["./index.html", "./src/\*\*/\*.{js,ts,jsx,tsx}"],

No seu arquivo CSS principal (ex: index.css ou globals.css), adicione:

@tailwind base;

@tailwind components;

@tailwind utilities;

Se der algum erro, execute este comando no terminal para garantir que o Tailwind está instalado corretamente:

* npm list tailwindcss

Se não aparecer nada, tente reinstalar:

* npm install -D tailwindcss postcss autoprefixer

## Criar outras páginas no react (9\_ProjetoTarefasYoutube)

Usando a biblioteca [React Router dom](https://www.npmjs.com/package/react-router-dom)

No terminal:

* npm i react-router-dom@(e a versao)

Cria outra pasta em src: routes.jsx

Nele colocar: EX: MyPortifolio (mas foi tirado, pois é só um exemplo)

import { BrowserRouter, Routes, Route } from "react-router-dom";

//BrowserRouter: Cria um roteador baseado em histórico de navegação do navegador.

//Routes: listas de rotas

//Route: cada rota

import Firstmain from "./componentes/Firstmain";

import Presentation from "./componentes/pages/Presentation";

import Skills from "./componentes/pages/Skills";

import Knowledge from "./componentes/pages/Knowledge";

import Projects from "./componentes/pages/Projects";

function AppRoutes() {

  return (

    <BrowserRouter>

      <Routes>

        {/\* path="/" : Rota raiz \*/}

        <Route path="/" element={<Firstmain />}></Route>

        <Route path="/presentation" element={<Presentation />}></Route>

        <Route path="/skills" element={<Skills />}></Route>

        <Route path="/knowledge" element={<Knowledge />}></Route>

        <Route path="/projects" element={<Projects />}></Route>

      </Routes>

    </BrowserRouter>

  );

}

export default AppRoutes;

No app.jsx: importar o AppRoutes

import AppRoutes from "./routes";

function App() {

  return (

    <>

      <div>

        <AppRoutes />

      </div>

    </>

  );

}

export default App;

Agora vai onde é para ser clicável e redirecionado para pagina:

Ex: No Header.jsx:

import { Link } from "react-router-dom";

function Header() {

  return (

    <div className="header">

      <Link to="/"> 🡪 Adiciona o Link e importa la em cima

        <h1 className="h1Header">Dev.Tgs</h1>

      </Link>

      <div className="flex gap-4 cursor-pointer">

        <a href="https://github.com/Larytgs" target="blank">

          <img src="/img/logo/github.png" />

        </a>

        <a

          href="https://www.linkedin.com/in/laryssa-tortato-7612961a9/"

          target="blank"

        >

          <img src="/img/logo/linkedin.png" />

        </a>

        <a href="https://www.instagram.com/larytgs/" target="blank">

          <img className="mr-4" src="/img/logo/instagram.png" />

        </a>

      </div>

    </div>

  );

}

export default Header;

Outro Exemplo: (mas foi tirado, pois é só um exemplo)

<Link to="/knowledge">

   <Button>Conhecimentos</Button>

</Link>

* É só adicionar o <Link to=”*nome colocado no Route*”>*Nome do botão*</Link>

## Mostrando conteúdo em baixo, ao clicar em algum botão

import Button, { Lista } from "./estilização/First";

import Header from "./Header";

import { useState } from "react";

import { Presentation, Skills } from "./pages";

function Firstmain() {

  // null, ou seja, nenhuma seção está visível no início.

// Ou inicializando "presentation" como a seção visível

  // setVisibleSection é a função usada para atualizar o valor de visibleSection

  const [visibleSection, setVisibleSection] = useState("presentation");

  //Essa função toggleSection é chamada sempre que um botão é clicado, e ela recebe o nome de uma seção (como "presentation", "skills", etc.) como parâmetro (section)

  const toggleSection = (section) => {

    setVisibleSection(visibleSection === section ? null : section);

  };

  return (

    <>

      <Header />

      <section className="mb-6 p-5 mt-14">

        <div className="mt-10">

          <ul className="text-center">

            <Lista>

              <Button onClick={() => toggleSection("presentation")}>

                Apresentação

              </Button>

              {visibleSection === "presentation" && <Presentation />}

            </Lista>

            <Lista>

              <Button onClick={() => toggleSection("skills")}>

                Habilidades

              </Button>

              {visibleSection === "skills" && <Skills />}

            </Lista>

            <Lista>

          </ul>

        </div>

{/\* Adiciona um espaçamento para evitar que o conteúdo fique coberto \*/}

        <div className="col-span-12 mt-4 sm:mt-10">

          {visibleSection === "presentation" && <Presentation />}

          {visibleSection === "skills" && <Skills />}

          {visibleSection === "knowledge" && <Knowledge />}

          {visibleSection === "projects" && <Projects />}

        </div>

      </section>

    </>

  );

}

export default Firstmain;

O que acontece aqui:

Se o nome da seção passada (por exemplo, "presentation") for igual ao valor atual de visibleSection, então setVisibleSection(null) é chamado, ou seja, a seção será **ocultada**.

Caso contrário, a seção será **exibida**. Ou seja, se o nome da seção não for igual ao valor atual de visibleSection, ela será definida como o valor de visibleSection, ou seja, **a seção será exibida**.

Como está num Componente Button, vai onde esta importado:

function Button(props) {

  return (

    <button

      onClick={props.onClick} 🡨 Adiciona essa parte

      className="text-3xl pt-[10px] pb-[10px] pl-[70px] pr-[70px]"

    >

      {props.children}

    </button>

  );

}

### Para adicionar uma animação de entrada suave da esquerda para a direita, podemos usar a biblioteca **Framer Motion**.

1. Instalar o Framer Motion (se ainda não tiver instalado):

npm install framer-motion

1. import { motion } from "framer-motion";

//Cards as divs do conhecimentos - Knowledge.jsx

function Cards(props) {

  return (

    <motion.div

      initial={{ opacity: 0, x: -50 }} // Começa invisível e deslocado para a esquerda

      animate={{ opacity: 1, x: 0 }} // Anima para a posição normal

      transition={{ duration: 1.5, ease: "easeOut}} // Duração e suavização da animação

      className="ml-3 bg-[#333] rounded-2xl w-[400px] h-[280px] m-auto align-middle p-4 text-left"

    >

      {props.children}

    </motion.div>

  );}

## Colocando zoom num botao/imagem

Só adicionar isso: transition-transform duration-500 hover:scale-110

<figure>

   <img

    className="w-[35rem] m-auto cursor-zoom-in transition-transform duration-500 hover:scale-110"

     src="/img/Tendencias-de-tecnologia-2023-Capa.jpg"

   />

</figure>

**transition: transform 0.5s;** com :hover **transform: scale(1.2);**

# Criando componentes funcionais input, button e title

Para diminuir a duplicidade do código, principalmente a estilização

Criar o componente Input.jsx ; Buttonjsx ; Title.jsx



Input.jsx:

function Input(props) {

  //props é um objeto contendo as propriedades passadas para o componente.

  return (

    <input

      className="bg-slate-300 border-slate-300 text-gray-950 outline-slate-700 px-4 py-2 rounded-md"

      type={props.type}

      placeholder={props.placeholder}

      value={props.value}

      onChange={props.onChange}

      //   {...props} //todas as props

      // Isso tornaria o componente mais flexível, permitindo que ele aceite qualquer propriedade do <input> padrão do HTML.

    />

  );

}

export default Input;

Usando essa função... importar no arquivo, e colocar o ‘Input’

import Input from "./Input";

function App() {

  return (

    <div>

      <Input

        type="text"placeholder="Digite o titulo da tarefa"

        value={title}onChange={(event) => setTitle(event.target.value)}

      />

      <Input

        type="text" placeholder="Digite a descrição da tarefa"

        value={description} onChange={(event) => setDescription(event.target.value)}

      />

    </div>

  );

}

export default App;

Button.jsx:

function Button(props) {

  //props é um objeto contendo as propriedades passadas para o componente.

  return (

    <button

      {...props}

      className={"bg-slate-400 text-left text-white p-2 rounded-md "}

    >

      {props.children}{" "}

      {/\* O conteúdo do botão (texto, ícones, etc.) será definido pelo props.children \*/}

      {/\* children representa tudo que for passado dentro da tag <Button>...</Button> \*/}

    </button>

  );

}

export default Button;

Usando a função, importando primeiro dps adicionando onde desejar

<Button onClick={() => onSeeDetailClick(tasks)}>

     <SlArrowRight />

</Button>

<Button onClick={() => onDeleteTaskClick(tasks.id)}>

     <FaRegTrashAlt />

</Button>

Title.jsx:

function Title(props) {

  //props é um objeto contendo as propriedades passadas para o componente.

  return (

    <h1 className="text-4xl text-slate-300 font-bold text-center">

      {props.children}

    </h1>

  );

}

export default Title;

Usando a função, importando primeiro dps adicionando onde desejar

<Title>Gerenciador de Tarefas</Title>

## Mudar o backgroud e o color junto com cada section, ao clicar num botão

Exemplo do projeto de Fotografias TGS:



// Cada botão muda, da seção + muda o background + muda a cor do header.

// Essa sectionColors é só um objeto que liga cada seção a uma cor.

  const sectionBackgroundColors = {

    beach: "#d4a14e9a",

    food: "#60210C",

    moments: "#B584AD",

    animals: "#A9CA76",

  };

  // Cada botão muda, da seção + muda a letra do header.

// Essa textColors é só um objeto que liga cada seção a uma cor

  const textColors = {

    beach: "#2E5DAD",

    food: "#8AA33B",

    moments: "#B7011F",

    animals: "#4D392E",

  };

// Aí você usa ela no seu section:

  return (

    <>

      <section

        style={{

          backgroundColor: sectionBackgroundColors[visibleSection],

          color: textColors[visibleSection],

          minHeight: "100vh",

        }}

      >

        <div

          className="flex justify-around m-0"

          style={{

            backgroundColor: sectionBackgroundColors[visibleSection],

            color: textColors[visibleSection],

          }}

        >

          <h1

            style={{

              color: textColors[visibleSection],

            }}

            className="font-black text-left m-5"

          >

            Fotografia Tgs

          </h1>

          <Ul>

            <li>

              <Button onClick={() => toggleSection("beach")}>Beach</Button>

            </li>

            <li>

              <Button onClick={() => toggleSection("food")}>Food</Button>

            </li>

            <li>

              <Button onClick={() => toggleSection("moments")}>Moments</Button>

            </li>

            <li>

              <Button onClick={() => toggleSection("animals")}>Animals</Button>

            </li>

          </Ul>

        </div>

      </section>

## Colocando um cursor personalizado, tirando o padrão

Adicione no index.css:  
Adiciona cursor: none; do body e adicione o CSS para a classe .cursor:

/\*Remove o cursor padrão \*/

body {

  cursor: none;

}

/\* Estilizando o cursor personalizado \*/

.cursor {

  width: 20px;

  height: 20px;

  border-radius: 50%;

  cursor: help;

  background-color: #00ffff80; /\* Cor turquesa com transparência \*/

  position: absolute;

  pointer-events: none; /\* Evita interferência com cliques \*/

  transform: translate(-50%, -50%);

  transition: transform 0.5s linear;

}

Crie o Componente de Cursor Personalizado (CustomCursor.jsx):

import { useEffect, useState } from "react";

const CustomCursor = () => {

  const [position, setPosition] = useState({ x: 0, y: 0 });

  //Aqui, criamos um estado (position) que armazena a posição do cursor no eixo X e Y.

  //Estado inicial: { x: 0, y: 0 } → O cursor começa no canto superior esquerdo da tela.

  //Efeito para capturar o movimento do mouse

  useEffect(() => {

    const moveCursor = (e) => {

      setPosition({ x: e.clientX, y: e.clientY });

    };

    //e.clientX e e.clientY capturam a posição atual do cursor.

    //setPosition({ x: e.clientX, y: e.clientY }) atualiza o estado com a nova posição

//Cria a função moveCursor, que é chamada sempre que o mouse se move (mousemove).

    window.addEventListener("mousemove", moveCursor);

    return () => window.removeEventListener("mousemove", moveCursor);

  }, []); //([] no final indica que ele só executa quando o componente é montado).

  return (

    <div

      className="cursor"

      style={{

        left: `${position.x}px`,

        top: `${position.y}px`,

      }}

      // Retorna um <div> com a classe "cursor" que será estilizada no CSS.

      // A posição do cursor é atualizada dinamicamente usando inline styles:

      // left: ${position.x}px → Posiciona horizontalmente.

      // top: ${position.y}px → Posiciona verticalmente.

    />

  );

};

export default CustomCursor;

Adicione no App.js:

import CustomCursor from "./componentes/estilização/CustomCursor";

import AppRoutes from "./routes";

function App() {

  return (

    <>

      <div>

         <CustomCursor /> {/\* O cursor \*/}

        <AppRoutes />

      </div>

    </>

  );

}

export default App;

### Seta para subir a tela

Instale o pacote de ícones (se ainda não tiver):

* npm install react-icons

Importe e coloque o botão no seu App:

import ScrollToTopButton from './components/ScrollToTopButton';

function App() {

  return (

    <>

      {/\* Seu conteúdo aqui \*/}

      <ScrollToTopButton />

    </>

  );

}

Novo Componente ScrollToTopButton.jsx

### Clicar na imagem → ampliar / desampliar: Zoom na foto

Coloca no img

ZoomableImage src="sua-imagem.jpg" alt="Descrição da imagem" />

Faz outro componente ZoomableImage.jsx:

import React, { useState } from "react";

function ZoomableImage({ src, alt }) {

  const [isOpen, setIsOpen] = useState(false);

  const openImage = () => setIsOpen(true);

  const closeImage = () => setIsOpen(false);

  return (

    <>

      {/\* Imagem pequena \*/}

      <img

        src={src}

        alt={alt}

        onClick={openImage}

        style={{

          width: "300px", // tamanho inicial

          cursor: "zoom-in",

          borderRadius: "8px",

          transition: "transform 0.3s",

        }}

      />

      {/\* Modal da imagem ampliada \*/}

      {isOpen && (

        <div

          onClick={closeImage}

          style={{

            position: "fixed",

            top: 0,

            left: 0,

            width: "100vw",

            height: "100vh",

            backgroundColor: "rgba(0, 0, 0, 0.8)",

            display: "flex",

            alignItems: "center",

            justifyContent: "center",

            zIndex: 9999,

            cursor: "zoom-out",

          }}

        >

          <img

            src={src}

            alt={alt}

            style={{

              maxWidth: "90%",

              maxHeight: "90%",

              borderRadius: "8px",

              objectFit: "contain",

            }}

          />

        </div>

      )}

    </>

  );

}

export default ZoomableImage;

// A imagem pequena fica clicável.

// Quando clica:

// Abre um modal cobrindo a tela com fundo preto semi-transparente.

// A imagem ocupa no máximo 90% da largura e altura da tela.

// Se clicar de novo na tela escura ou na imagem → fecha.

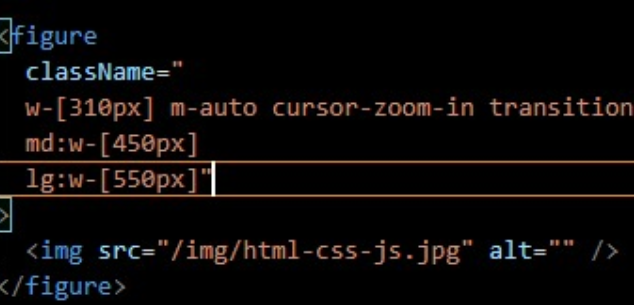
// E no responsivo?

// Como usamos max-width: 90% e max-height: 90%, ele se adapta automaticamente no celular, tablet, etc.

## Responsivo com tailwindcss

[Responsivo com Grids e Breakpoints -- Youtube](https://www.youtube.com/watch?v=1287-96sMuk)

| **Prefixo de ponto**  **de interrupção** | **Largura mínima** | **CSS** |
| --- | --- | --- |
| sm | 40rem *(640px)* | @media (width >= 40rem) { ... } |
| md | 48rem *(768px)* | @media (width >= 48rem) { ... } |
| lg | 64rem *(1024px)* | @media (width >= 64rem) { ... } |
| xl | 80rem *(1280px)* | @media (width >= 80rem) { ... } |
| 2xl | 96rem *(1536px)* | @media (width >= 96rem) { ... } |

Exemplo, a imagem.. começa com o tamanho pequeno (width:310px)

Depois o md(passa pra width:450px)

Depois o lg (passa para width:550px)

Ele vai aumnetando

### Dica nova – adicionar um fundo dentro do titulo

h2 {

  font-size: 3rem;

  color: #fff;

  background: url('Images/gradientbg.jpg') center;

  background-size: cover;

  background-clip: text;

  color: transparent;

  animation: animate 10s linear infinite

}

@keyframes animate {

  to {

    background-position-x: -550px; exemplo: do portifólio

  }

}

### Dica nova – colocar o titulo/texto digitando que nem maquina de escrever

import { Typewriter } from "react-simple-typewriter";

<span className="text-[0.6em] font-thin text-[#797676] bg-none">

  <Typewriter

     words={["Desenvolvedora Web Front-end"]}

    loop={true} // 🔁 Se true, digita/apaga infinitamente; false digita uma vez só

     cursor // ✏️ Mostra o cursor piscando (|) no final do texto

     cursorStyle="|" // 💅 Estilo do cursor (você pode mudar para \_ ou █ se quiser)

     typeSpeed={80} // ⏩ Velocidade de digitação (ms por caractere) – menor = mais rápido

     deleteSpeed={50} // ⏪ Velocidade de apagar (ms por caractere)

     delaySpeed={1500} // ⏸ Tempo de espera (ms) antes de começar a apagar o texto

  />

</span>

exemplo: do portifólio

### Dica nova – Colocar body claro ou escuro - exemplo no portifólio

import { useEffect, useState } from "react"; // Importa os hooks do React

import { Link } from "react-router-dom"; // Link para navegação sem recarregar a página

import "./StyleH.css"; // Importa o estilo personalizado

function Header() {

const [darkMode, setDarkMode] = useState(false); // Estado para controlar se está no modo escuro

useEffect(() => {

// Toda vez que darkMode mudar, essa função executa

document.body.classList.toggle("dark", darkMode); // Adiciona ou remove a classe 'dark' do <body>

}, [darkMode]);

return (

<div className="header text-3xl text-[#6ad0ff] font-bold sm:text-4xl flex justify-between items-center p-4">

{/\* Logo ou título do site, clicável para ir à página inicial \*/}

<Link to="/">

<h1 className="h1Header">Dev.Tgs</h1>

</Link>

{/\* Área à direita do cabeçalho com ícones e toggle \*/}

<div className="flex items-center gap-4">

{/\* Botão de alternância de modo claro/escuro \*/}

<label className="relative inline-flex items-center cursor-pointer">

{/\* input invisível que controla o estado \*/}

<input

type="checkbox"

checked={darkMode} // liga com o estado

onChange={() => setDarkMode(!darkMode)} // muda o estado ao clicar

className="sr-only peer" // escondido visualmente/>

{/\* Trilha do toggle (fundo) \*/}

<div className="w-11 h-6 bg-gray-300 peer-focus:outline-none peer-focus:ring-2 peer-focus:ring-[#6ad0ff] rounded-full peer dark:bg-gray-600 peer-checked:bg-[#6ad0ff] transition-colors"></div>

{/\* Círculo que desliza (a “bolinha”) \*/}

<div

className={`absolute left-1 top-1 w-4 h-4 bg-white rounded-full transition-transform

${ darkMode ? "translate-x-5" : ""}`}

></div>

</label>

export default Header; // Exporta o componente para ser usado em outro lugar

CSS:

body.dark {

  background-color: white;

  color: black;

}

body {

  transition: background-color 0.8s, color 1.8s;

}

Se tiver algum botão que não altera quando clica nesse botão:

<button className="bg-black text-white dark:bg-white dark:text-black px-4 py-2 rounded-md flex items-center gap-2">

  Download CV

  <span>⬇️</span>

</button>

**dark:**bg-white dark:text-black: muda para fundo branco e texto preto quando a classe dark estiver no <body>.

### Nova dica: para os títulos ou conteúdos, por exemplo, deslizando de cima e aparecendo com opacidade) quando ele aparece na tela ao rolar a página (scroll).

Primeiro importa o motion:

import { motion } from "framer-motion";

E adiciona <motion.div> <motion.li> <motion.h3>

<motion.div

     className="flex relative w-full justify-center mt-20 mb-24"

     initial={{ opacity: 0, y: -30 }} // Começa invisível e deslocado para cima

     whileInView={{ opacity: 1, y: 0 }} // Anima para visível e na posição certa

     transition={{

     duration: 1.5,

     delay: 0.2,

     ease: [0.5, 0, 0, 1], // Aproxima da cubic-bezier(0.5, 0, 0, 1)

    }}

     viewport={{ once: false }} // anima toda vez ao entrar na viewport>

     <h3 className="absolute uppercase text-[#414040] text-xl font-bold">Cursos</h3>

</motion.div>

Adicionado:

initial={{ opacity: 0, y: -30 }} // Começa invisível e deslocado para cima

whileInView={{ opacity: 1, y: 0 }} // Anima para visível e na posição certa

transition={{

duration: 1.5,

delay: 0.2,

ease: [0.5, 0, 0, 1], // Aproxima da cubic-bezier(0.5, 0, 0, 1)

}}

viewport={{ once: false }} // anima toda vez ao entrar na viewport>

### Fazendo formulário exemplo no meu portifólio

import Section, { P } from "../styles/Section";

import H2 from "../styles/Subtitulo";

import Form, { Input } from "../styles/Form";

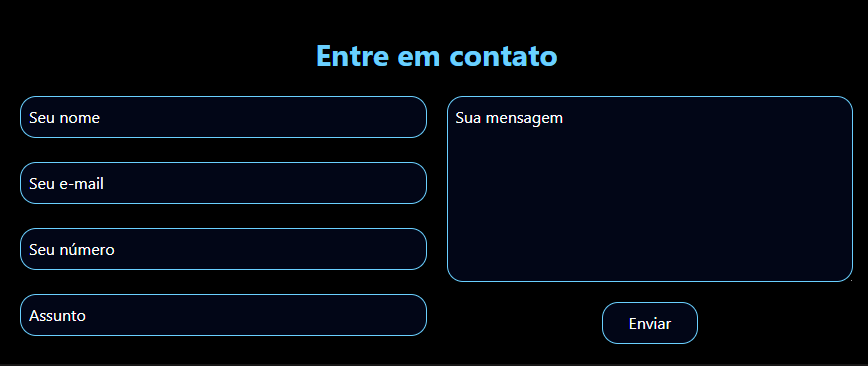
import { useState } from "react";

//Biblioteca AXIOS para coectar ao API

import axios from "axios";

const Formulario = () => {

  // Declarar uma nova variavel de dados com state e atribuir objetos

  // Este useState cria uma variável chamada data que guarda todos os valores dos campos do formulário como um objeto.

  const [data, setData] = useState({

    name: "",

    email: "",

    telefone: "",

    assunto: "",

    msg: "",

  });

  // Declarar a variavel para receber a mensagem

  const [message, setMessage] = useState("");

  //Receber todos os valores dos campos do formulario

  const valorInput = (e) =>

    setData({ ...data, [e.target.name]: e.target.value });

  // e → é o evento do input (por exemplo, digitar no campo).

  // ...data : copia todos os campos anteriores (para não perder o conteúdo dos outros inputs)

  // e.target.name → acessa o nome do campo (como "email" ou "msg").

  // e.target.value → pega o valor digitado.

  // [e.target.name]: e.target.value → define dinamicamente a propriedade do objeto data com base no nome do campo

  const sendMsg = async (e) => {

    e.preventDefault(); //para bloquear o recarregamento da pagina

    // Manipular os dados recebidos, por exemplo, enviar os dados para API

    // Concatenar e imprimir utilizando Template strings

    console.log(`Nome: ${data.name}`);

    console.log(`E-mail: ${data.email}`);

    console.log(`Telefone: ${data.telefone}`);

    console.log(`Assunto: ${data.assunto}`);

    console.log(`Conteudo: ${data.msg}`);

    // Criar a constante com os dados do cabeçalho

    const headers = {

      headers: {

        // Indicar que sera enviado os dados em formato de objeto

        "Content-Type": "application/json",

      },

    };

    // Fazer a requisição para o servidor utilizando axios, indicando o método da requisição, o endereço, enviar os dados do formulário e o cabeçalho

    await axios

      .post(`${process.env.REACT\_APP\_API\_URL}/message`, data, headers)

      .then((response) => {

        // acessa o then qnd a API retornar status 200

        // Atribuir a msg no state mensage

        setMessage(response.data.message);

      })

      .catch((err) => {

        if (err.response) {

          setMessage(err.response.data.message);

        } else {

          setMessage("Erro de conexão com o servidor");

        }

      });

  };

  return (

    <Section>

      <H2>Entre em contato</H2>

      {/\* Imprimir a mensagem retornada da API \*/}

      {message ? <p>{message}</p> : ""}

      {/\* Início do formulário, executar o onSubmit quando usuário clicar no botão submit e chamar a funcao sendMsg \*/}

      <Form onSubmit={sendMsg}>

        {/\* Inputs (coluna esquerda) \*/}

        <div className="flex flex-col gap-4">

          <Input

            type="text"

            name="name"

            placeholder="Seu nome"

            //onChange={(e) => setData({ ...data, name: e.target.value })}

            onChange={valorInput}

            required //campo obrigatório (não permite enviar vazio)

            //Criar o campo, quando o usuário digitar valor no campo, chamar com onChange a funcao valorInput

            // onChange={...} : Esse é um evento quando o valor do input muda (o usuário digita, apaga, cola..)

            // (e) => : Essa é uma função anônima que recebe o evento (geralmente chamado de e ou event)

          />

          <Input

            type="email"

            name="email"

            placeholder="Seu e-mail"

            onChange={valorInput}

            required //campo obrigatório (não permite enviar vazio)

          />

          <Input

            type="text"

            name="telefone"

            placeholder="Seu número"

            onChange={valorInput}

            required //campo obrigatório (não permite enviar vazio)

          />

          <Input

            type="text"

            name="assunto"

            placeholder="Assunto"

            onChange={valorInput}

            required //campo obrigatório (não permite enviar vazio)

          />

        </div>

        {/\* Textarea + botão (coluna direita) \*/}

        <div className="flex flex-col gap-4 justify-between h-full">

          <textarea

            className="p-2 m-1 border rounded-2xl border-[#6ad0ff] bg-slate-950 placeholder-white focus:outline-none focus:ring-2 focus:ring-blue-500 transition"

            type="text"

            name="msg"

            rows="7"

            placeholder="Sua mensagem"

            onChange={valorInput}

          />

          <div className="col-span-1 md:col-span-2 flex justify-center">

            <button

              className="p-2 w-24 text-center border rounded-2xl border-[#6ad0ff] bg-slate-950 hover:bg-[#6ad0ff73] transition "

              type="submit"

            >

              Enviar

            </button>

          </div>

        </div>

      </Form>

    </Section>

  );

};

export default Formulario;

Parte do design:

function Form(props) {

  return (

    <form

      onSubmit={props.onSubmit} // necessário para funcionar corretamente o envio do formulario, pois ele foi customizado

      className="m-5 text-left grid grid-cols-1 md:grid-cols-2 gap-3 max-w-[1000px] mx-auto"

    >

      {props.children}

    </form>

  );

}

function Input(props) {

  return (

    <input

      className="p-2 m-1 border rounded-2xl border-[#6ad0ff] bg-slate-950 placeholder-white focus:outline-none focus:ring-2 focus:ring-blue-500 transition"

      {...props}

    ></input>

  );

}

export default Form;

export { Input };

### Para receber os valores, adiciona:

const Formulario = () => {

  // Declarar uma nova variável de dados com state e atribuir objetos

  // Este useState cria uma variável chamada data que guarda todos os valores dos campos do formulário como um objeto.

  const [data, setData] = useState({

    name: "",

    email: "",

    telefone: "",

    assunto: "",

    msg: "",

  });

  //Receber todos os valores dos campos do formulário

  const valorInput = (e) =>

    setData({ ...data, [e.target.name]: e.target.value });

  // e → é o evento do input (por exemplo, digitar no campo).

  // ...data : copia todos os campos anteriores (para não perder o conteúdo dos outros inputs)

  // e.target.name → acessa o nome do campo (como "email" ou "msg").

  // e.target.value → pega o valor digitado.

  // [e.target.name]: e.target.value → define dinamicamente a propriedade do objeto data com base no nome do campo

  const sendMsg = (e) => {

    e.preventDefault(); //para bloquear o recarregamento da pagina

    // Manipular os dados recebidos, por exemplo, enviar os dados para API

    // Concatenar e imprimir utilizando Template strings

    console.log(`Nome: ${data.name}`);

    console.log(`E-mail: ${data.email}`);

    console.log(`Telefone: ${data.telefone}`);

    console.log(`Assunto: ${data.assunto}`);

    console.log(`Conteudo: ${data.msg}`);

  };

### Colocando no return:

{/\* Início do formulário, executar o onSubmit quando usuário clicar no botão submit e chamar a funcao sendMsg \*/}

      <Form onSubmit={sendMsg}>

### E nos inputs adiciona:

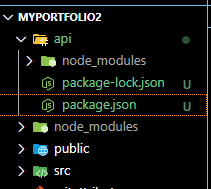
onChange={valorInput}

Criar o campo, quando o usuário digitar valor no campo, chamar com onChange a funcao valorInput

onChange={...} : Esse é um evento quando o valor do input muda (o usuário digita, apaga, cola..)

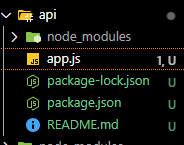
(e) => : Essa é uma função anônima que recebe o evento (geralmente chamado de e ou event)

### Agora vamos criar a API

Cria o novo arquivo api

**no terminal**: cd api 🡪 npm init 🡪 vai dando enter que não alterar em nada

ele vai mostrar a pasta package.json, no terminal: npm i --save express



Criar um novo arquivo app.js 🡪

// Incluir as bibliotecas

// Gerencia as requisições, rotas e URLs, entre outra funcionalidades

const express = require("express");

// Chamar a funcao express

const app = express();

app.listen();

No terminal coloca 🡪 node app.js se não aparecer nada, é que deu certo (que nem npm run dev)

Adiciona a porta 8080

app.listen(8080, () => {

  console.log("Servidor iniciado na porta 8080: http://localhost:8080");

});

No terminal coloca 🡪 node app.js para rodar o projeto novamente

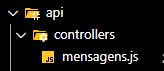
Para eu não ter que rodar esse código toda vez que fizer uma alteração, coloque no terminal: **npm install -- save-dev nodemon** ele vai atualizar sozinho

Se nunca instalou a dependência na máquina, coloque primeiro: **npm install -g nodemon** após instalar, reiniciar o PC

Em seguida coloca no terminal: **nodemon app.js** e ele atualizará sozinho

### O padrão MVC (Model-View-Controller)

é uma arquitetura de software que divide a aplicação em três componentes principais: Model (representa os dados e a lógica da aplicação), View (representa a interface do usuário) e Controller (interage com o Model para alterar a View). Em APIs, a View é implícita, e o foco principal está nos Models e Controllers, que lidam com a comunicação entre o cliente e os dados da API



Criar o diretório controllers 🡪 mensagens.js

// Incluir as bibliotecas

// Gerencia as requisições, rotas e URLs, entre outra funcionalidades

// Importando o modulo express, Ele facilita a criação de servidores HTTP, gerencia rotas, requisições e respostas, etc.

const express = require("express");

// Chamar a função express

const router = express.Router(); // Quero somente importar a parte das rotas

// express.Router() cria um "mini app" do Express, útil para definir rotas separadas em arquivos diferentes.

// Criar a rota cadastrar

router.post("/", async (req, res) => {

  // req: requesição res: resposta

  return res.json({

    error: false,

    mensagem: "Mensagem cadastrada com sucesso!",

  });

});

// Exportar a instrução que esta dentro da constante router

module.exports = router;

Aqui você está definindo uma **rota do tipo POST** no caminho /.

* router.post(...): define uma rota que responde a requisições HTTP do tipo **POST**.
* "/": indica que a rota é o caminho raiz daquele grupo de rotas.
* async (req, res) => { ... }: é a função que será executada quando a rota for acessada.
  + req: é o objeto da requisição (dados enviados pelo cliente).
  + res: é o objeto de resposta (usado para responder ao cliente).
* res.json(...): envia uma resposta no formato JSON.
  + O conteúdo { error: false, mensagem: "Mensagem cadastrada com sucesso!" } indica que não houve erro e exibe uma mensagem.

Volta no arquivo app.js para incluir os Controllers e criar as rotas

// Incluir os Controllers

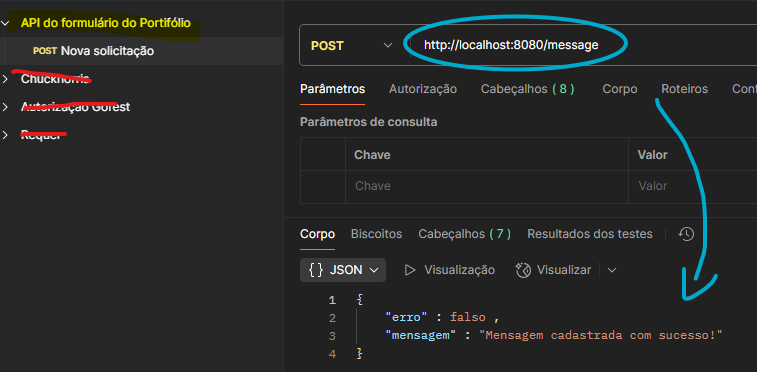
const mensagens = require("./controllers/mensagens");

// Criar as rotas

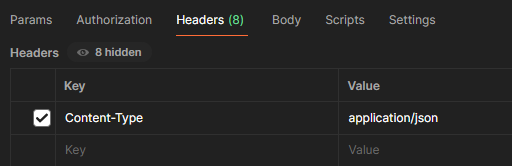
app.use("/message", mensagens); //quando o usuario colocar mensagem, ele quer acessar a rota mensagens

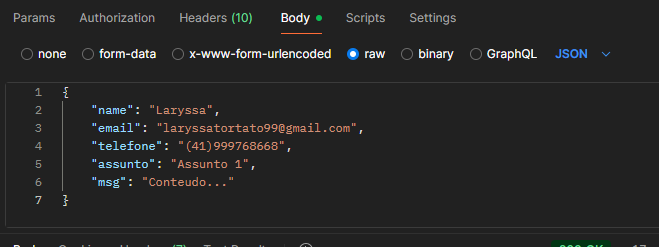
Agora entrar no Postman, criar uma nova Colletion, adicionar o título, e criar uma nova requisição Post

Adiciona o http que criamos /message, aperta para enviar e ve se esta funcionando

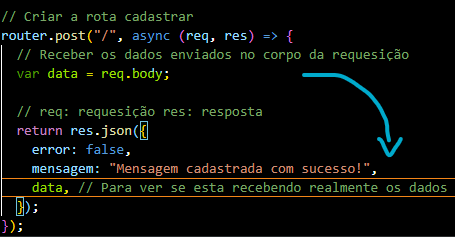


Adiciona no Cabeçalho:



Agora vai no corpo 🡪 raw 🡪 JSON e adiciona os campos colocado no formulário  


Volta no VSCode em mensagens.js e adiciona



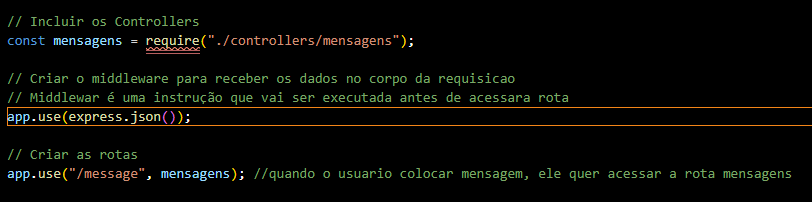
var data = req.body;

E la em baixo 🡪 data

### Para funcionar e aparecer os dados, quando clicar em send no postiman

Vamos voltar para o app,js e adicionar apenas:

app.use(express.json());



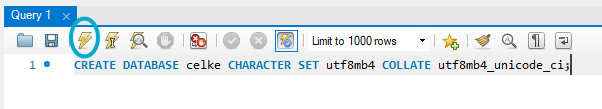
Faz o teste no postman, ele tem que retornar os dados do formulário

### Agora vamos usar o PHPMyAdmin ou MySQL

No MySQL cria um novo

Colocando o nome e a porta 3306

add: CREATE DATABASE celke CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci; 🡪 Executa



Ele vai aparecer no PHPMyAdmin tbm

**Agora temos que conectar o API no MySQL**

No terminal do VSCODE coloque:

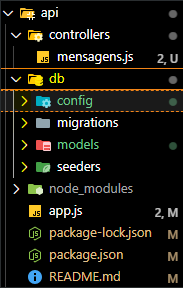
Sequelize é uma biblioteca Javascript que facilita o gerenciamento do banco de dados SQL:

**npm install -- save sequelize**

Instalar o drive do banco de dados: **npm install -- save mysql2**

Sequelize-cli interface de linha de comando usada para criar modelos, configurações e arquivos de migração para bancos de dados: **npm install -- save-dev sequelize-cli**

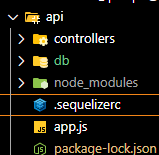
Iniciar o Sequelize-cli e criar o arquivo config: **npx sequelize-cli init**

Ele vai criar uns arquivos, você vai fazer uma pasta

chamada **db**, e coloque esses arquivos dentro dele:

Depois criar novo arquivo:

***.sequelizerc***

Add:

const path = require("path");

module.exports = { add “db”

  config: path.resolve("db", "config", "config.json"),

  "models-path": path.resolve("db", "models"), 🡪

  "seeders-path": path.resolve("db", "seeders"),

  "migrations-path": path.resolve("db", "migrations"),

};

Podendo fechar já  
Isso esta na pagina <https://sequelize.org/docs/v6/other-topics/migrations/>

Manipular variaveia de ambiente, para poder renomear algumas partes: **npm i dotenv --save**

E criar novo arquivo: **.env**

Add:

TZ = 'America/Sao\_Paulo' # Fuso horário utilizado no projeto - Padrão

NODE\_ENV=development # Ambiente do projeto

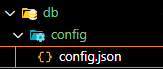
DB\_HOST=127.0.0.1 # Local do banco de dados

DB\_BASE=celke # Base de dados

DB\_USER=root # Usuario do banco de dados

DB\_PASS= # Senha do banco de dados

DB\_DIALECT=mysql # Tipo do banco de dados

Agora vamos acrescentar no config.js dentro do db 🡪 config

Vamos alterar umas coisas, primeiro mudar para **.js** esse arquivo

Vai dar erro, pois precisa exportar agora:

module.exports = { ... }

Em cima coloca:

require("dotenv").config();

Para utilizar, temos que alterar algumas coisas em baixo com os dados do **.env**:

module.exports = {

  development: {

    username: process.env.DB\_USER,

    password: process.env.DB\_PASS,

    database: process.env.DB\_BASE,

    host: process.env.DB\_HOST,

    dialect: process.env.DB\_DIALECT,

  },

  test: {

    username: process.env.DB\_USER,

    password: process.env.DB\_PASS,

    database: process.env.DB\_BASE,

    host: process.env.DB\_HOST,

    dialect: process.env.DB\_DIALECT,

  },

  production: {

    username: process.env.DB\_USER,

    password: process.env.DB\_PASS,

    database: process.env.DB\_BASE,

    host: process.env.DB\_HOST,

    dialect: process.env.DB\_DIALECT,

  },

};

Volta para o arquivo ***.sequelizerc*** e troque para .js se n vai dar erro futuros

### Agora temos que conectar no banco de dados, vai em models 🡪 index.js

Arruma a linha 9, para .js

const config = require(\_\_dirname + "/../config/config.js")[env];

Add antes do fs:

// Verificar a conexão com banco de dados

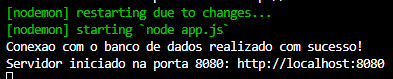
try {

  console.log("Conexao com o banco de dados realizado com sucesso!");

} catch (error) {

  console.log(

    "Erro: Conexão com o banco de dados não realizado com sucesso.",

    Error );

}

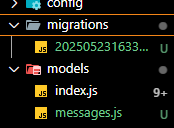
Vai em **app.js add:**

// Testar conexão com banco de dados

const db = require("./db/models");

Criar a Models mensagem, para manipular as informaçoes e os registros junto com o Banco de dados:

**npx sequelize-cli model:generate --name Messages –attributes name:string,email:string,subject:string,content:string**

Criando o migrations e o models > messages.js

Executar a migrations: **npx sequelize-cli db:migrate**

Vai em **app.js e retira:**

// Testar conexão com banco de dados

const db = require("./db/models");

E adiciona ele no controllers > menssages.js Mudando o nome para Incluir..

const express = require("express");

// Incluir conexão com banco de dados

const db = require("./db/models");

mais la p baixo

await db.Messages.create(data)

    .then((dataMessage) => {

      // req: requesição res: resposta

      return res.json({

        error: false,

        mensagem: "Mensagem cadastrada com sucesso!",

        data, // Para ver se esta recebendo realmente os dados

      });

    })

    .catch(() => {

      // req: requesição res: resposta

      return res.json({

        error: false,

        mensagem: "Mensagem NÂO cadastrada com sucesso!",

      });

    });

});

API funcionado, mas com erro

### Agora vamos conectar o formulário no API

Usando AXIOS, uma biblioteca, no terminal coloca: **npm install axios**

Voltando para a página de formulário, ‘depois que recebemos os dados, eu quero enviar os dados para API’:

Antes adiciona uma constante:

// Declarar a variável para receber a mensagem

const [message, setMessage] = usestate("");

Adiciona async na função sendMsg

const sendMsg = async (e) => {

    e.preventDefault(); //para bloquear o recarregamento da pagina

    // Manipular os dados recebidos, por exemplo, enviar os dados para API

    // Concatenar e imprimir utilizando Template strings

    console.log(`Nome: ${data.name}`);

    console.log(`E-mail: ${data.email}`);

    console.log(`Telefone: ${data.telefone}`);

    console.log(`Assunto: ${data.assunto}`);

    console.log(`Conteudo: ${data.msg}`);

    // Criar a constante com os dados do cabeçalho add essa parte:

    const headers = {

      'headers': {

          // Indicar que sera enviado os dados em formato de objeto

          'Content-Type': 'application/json'

        }

      };

    // Fazer a requisição para o servidor utilizando axios, indicando o método da requisição, o endereço, enviar os dados do formulário e o cabeçalho

    await axios.post('http://localhost:8080/message', data, headers)

    .then((response) => { // acessa o then qnd a API retornar status 200

      // Atribuir a msg no state mensage

      setMessage(response.data.message);

    }).catch((err) => {

        if (err.response) {

          setMessage(err.response.data.message);

        } else {

          setMessage("Erro de conexão com o servidor");

        }

      });

};

No return, depois do <h2>

{/\* Imprimir a mensagem retornada da API \*/}

{message ? <p>{message}</p> : ""}

Para não dar erros, instala no terminal: **npm i cors**

Vai no api > app.js e importa o cors, antes de executar a função express:

// Importar a biblioteca para permitir conexão externa

const cors = require("cors");

Adiciona:

// Middleware: permitir requisições externas com CORS (coloque isso sozinho)

app.use(cors());

// Criar o middleware para permitir requisição externa

app.use((req, res, next) => {

  // Qualquer endereço pode fazer requisição

  res.header("Access-Control-Allow-Origin", "\*");

  // Tipos de método que a API aceita

  res.header("Access-Control-Allow-Methods", "GET, PUT, POST, DELETE");

  // Permitir o envio de dados para API

  res.header("Access-Control-Allow-Headers", "Content-Type");

  // Executar o cors

  app.use(cors());

  // Quando não houver erro deve continuar o processamento

  next();

});

Se quiser continuar com esse formulario, eu comentei ele, e não esta mais visível

[postman](https://web.postman.co/workspace/My-Workspace~a8fe2784-d6ab-4a08-853d-abe1b9106f3f/request/41794704-fed39c53-161e-4d41-9bab-16646ea2cfcd?tab=params)

[localhost/phpmyadmin](http://localhost/phpmyadmin/index.php?route=/sql&pos=0&db=celke&table=messages)

## Quando recarregar a página o conteúdo se mante, usando LocalStorage

É um recurso de navegadores que permite armazenar dados no dispositivo do usuário. É uma forma de armazenamento persistente, ou seja, os dados não são apagados quando o navegador é fechad

No app.jsx adiciona: (onde estiver os elementos)

  //Salvar no localStorage

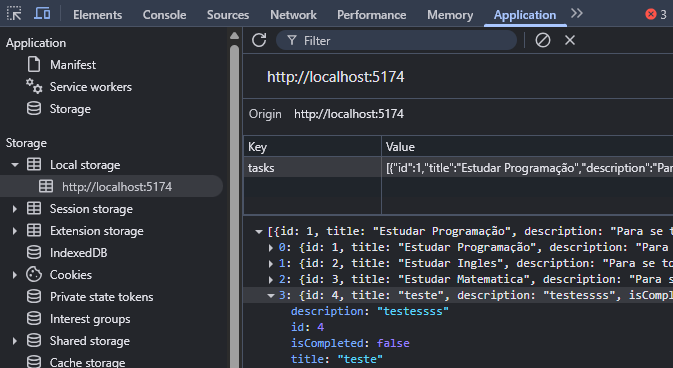
  useEffect(() => {

    localStorage.setItem("tasks", JSON.stringify(tasks));

  }, [tasks]); //toda vez que salvar uma tarefa ele salva no localStorage

* **useEffect**: Um efeito colateral que roda toda vez que o array tasks mudar.
* **JSON.stringify(tasks)**: Converte o array tasks para uma string JSON, pois o localStorage só armazena strings.
* **localStorage.setItem("tasks", JSON.stringify(tasks))**: Salva a string JSON no localStorage com a chave "tasks".
* **[tasks]**: O useEffect é acionado sempre que o estado tasks for atualizado. Isso garante que as alterações nas tarefas sejam salvas automaticamente no localStorage.

Para ver onde salvou, inspeciona o site. Adiciona uma nova tarefa  
Vai em application > Local storage



Agora, para manter mostrando na pagina, adiciona na lista de cima:

const [tasks, setTasks] = useState(

    JSON.parse(localStorage.getItem("tasks")) || []

  );

* **useState**: Define o estado tasks, que armazenará a lista de tarefas.
* **localStorage.getItem("tasks")**: Obtém os dados salvos no localStorage com a chave "tasks". Se houver tarefas salvas anteriormente, elas serão carregadas.
* **JSON.parse(...)**: Como os dados são salvos no localStorage como strings, usamos JSON.parse() para convertê-los de volta para um array de objetos.
* **|| []**: Se não houver nada salvo no localStorage, o estado começa como um array vazio ([])

### Esta dando muito erro de require e module?

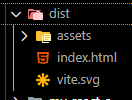
Adiciona no eslint.config.js:

env: {

   node: true,

},

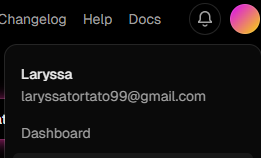
# Deploy colocando a aplicação online

No terminal coloque:

* Npm run build -> ele vai gerar uma pasta dist

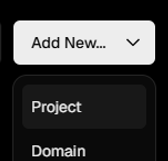
Os assets, ele inclui o arquivo js ele compila todo o

código js, para ficar mais performática possível

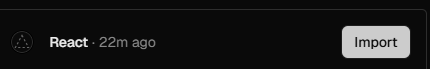


Entre no site vercel faça o login, depois vai em 🡪

Dashboard

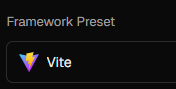


Depois em Add New 🡪



Importar o repositório 🡪

desejado



Colocar o Framework que utilizou 🡪

E clicar em Deploy em baixo

Dps ir para o projeto