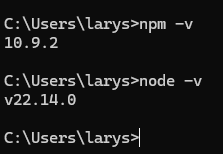
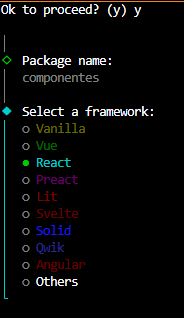
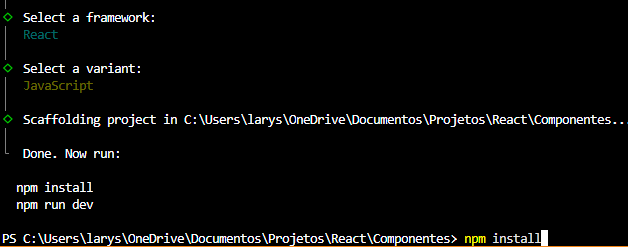
# **React**

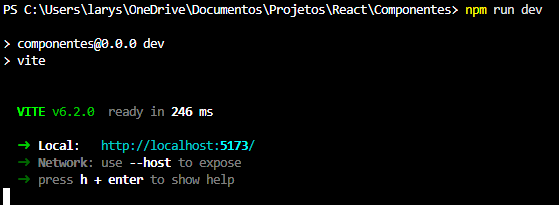
O React é uma biblioteca usada para criar interfaces de usuário interativas.

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.

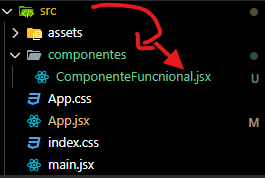
## Vamos começar em componentes.

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>

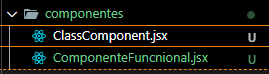
    </>

  )

}

//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>

    </>

  )

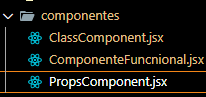
}

//Sempre temos que exportar

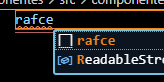
export default App;

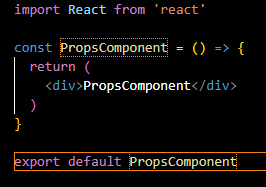
### Props

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",

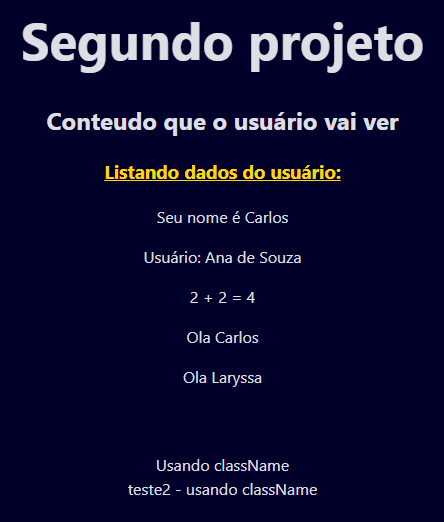
    };

    //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 />

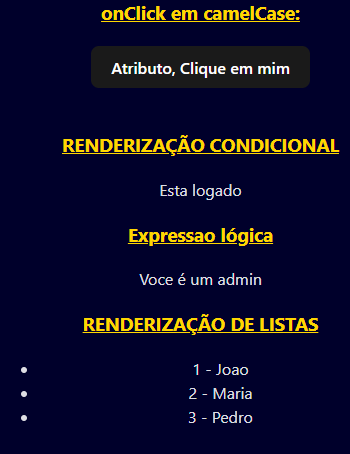
        <h3>onClick em camelCase:</h3>

        <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>

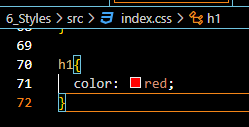
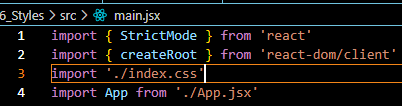
  )

}

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>

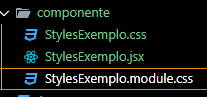
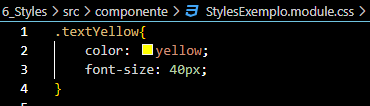
    </div>

  )

}

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

## 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)

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:

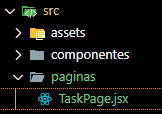
* npm install -D tailwindcss postcss autoprefixer

## 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)

Criar outra pasta em src

No main.jsx:

import { createBrowserRouter, RouterProvider } from "react-router-dom";

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

//RouterProvider: Fornece o roteador criado para a aplicação.

import TaskPage from "./paginas/TaskPage.jsx";

//TaskPage: Importa a página que será exibida quando o usuário acessar a rota correspondente.

const router = createBrowserRouter([

  {

    path: "/", //O path define o caminho da URL

    element: <App />, //element define o componente que será renderizado.

  },

  {

    path: "/task", //o nome da pagina, renderiza o componente <TaskPage /> (página de tarefas).

    element: <TaskPage />,

  },

]);

createRoot(document.getElementById("root")).render(

  <StrictMode>

    <RouterProvider router={router} />{" "}

    {/\* Passa o roteador criado para a aplicação, permitindo a navegação entre páginas. \*/}

    {/\* <App /> \*/}

  </StrictMode>

);

No TaskPag.jsx:

import { FaChevronLeft } from "react-icons/fa";

import { useNavigate, useSearchParams } from "react-router-dom";

function TaskPage() {

  const [searchParams] = useSearchParams(); //permite gerenciar os parâmetros de pesquisa de uma URL.

  const title = searchParams.get("title");

  const description = searchParams.get("description");

  const navigate = useNavigate();

  return (

    <div className="h-screen w-screen p-6 bg-slate-500">

      <div className="w-[500px] space-y-4">

        <div className="flex justify-center relative mb-6">

          <button

            onClick={() => navigate(-1)} //para voltar a pagina

            className="absolute left-0 top-0 text-slate-100 bg-transparent"

          >

            <FaChevronLeft></FaChevronLeft>

          </button>

          <h1 className="text-4xl text-slate-200 font-bold text-center">

            Detalhes da Tarefa

          </h1>

        </div>

        <div className="bg-slate-200 p-4 rounded-md">

          <h1 className="text-xl font-bold text-black">{title}</h1>

          <h2 className="text-black">{description}</h2>

        </div>

      </div>

    </div>

  );

}

export default TaskPage;

No Task.jsx

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

function Tasks({ tasks, onTaskClick, onDeleteTaskClick }) {

  const navigate = useNavigate(); //permite navegar entre páginas

  function onSeeDetailClick(tasks) {//criando uma função para chamar o titulo e a descrição

    const query = new URLSearchParams();

    query.set("title", tasks.title);

    query.set("description", tasks.description);

    navigate(`/task?${query.toString()}`);

  }

Adicionando no botão para chamar a outra pagina:

<button

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

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

>

## 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

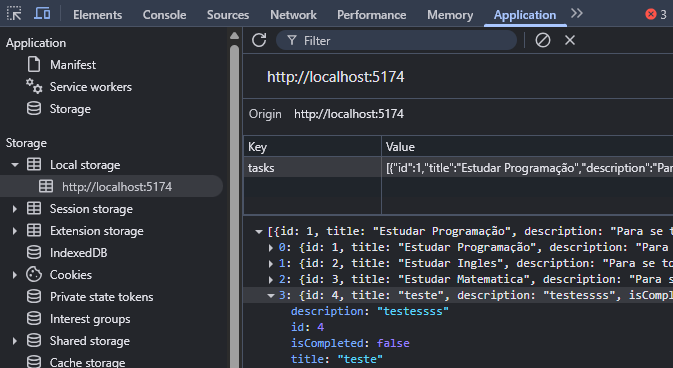
  useEffect(() => {

    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 ([])

# 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>

  );

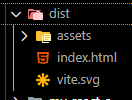
}

export default Title;

Usando a função, importando primeiro dps adicionando onde desejar

<Title>Gerenciador de Tarefas</Title>

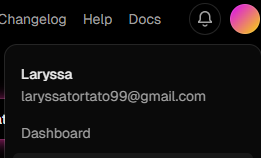
# Deploy colocando a aplicação online

No terminal coloque:

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

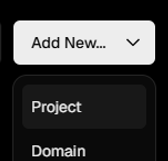
Os assets, ele inclui o arquivo js ele compila todo o

código js, para ficar mais performática possível

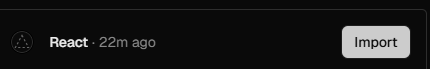


Entre no site vercel faça o login, depois vai em 🡪

Dashboard

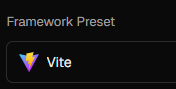


Depois em Add New 🡪



Importar o repositório 🡪

desejado



Colocar o Framework que utilizou 🡪

E clicar em Deploy em baixo

Dps ir para o projeto