E-BOOK
DO ESTUDANTE

REACT

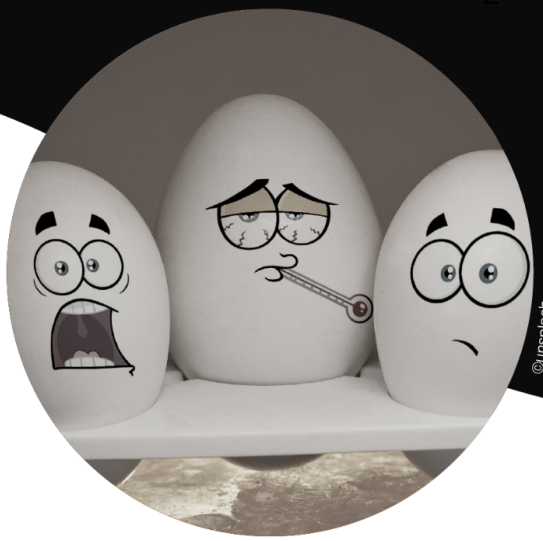
Reações

RESILIA

SUMÁRIO

REACT

Reações



Neste *ebook* iremos conhecer uma das bibliotecas de Javascript mais populares e a mais utilizada no mercado: o **React.js**. Desenvolvido pelo *ex Facebook* e atual *Meta*, o React foi criado para nos ajudar a desenvolver o *front-end* de um site utilizando as melhores e mais atuais tecnologias do mercado. Hoje em dia, as aplicações web e mobile são uma parte do nosso cotidiano, seja para fazer compras online, acessar redes sociais ou mesmo gerenciar finanças, por isso é importante que estas aplicações sejam rápidas, eficientes e fáceis de usar, e, para isso, utilizamos o React! Nesta jornada iremos entender como podemos desenvolver sites utilizando as tecnologias que a grande maioria dos sites utilizam, iremos conhecer e nos aprofundar em react e em seus fundamentos e iremos também conhecer o Vite, um build tolls que nos ajudará a criar projetos de react com velocidade, agilidade e principalmente estabilidade. Estas duas tecnologias juntas nos fornecem a melhor experiência do desenvolvimento web existente dentro do mercado web. Vamos então iniciar nossa aprendizagem em **React** e **Vite** e aprender mais sobre as duas tecnologias mais pedidas e valiosas para o mercado de desenvolvimento web?

CONTEXTUALIZANDO

O react é a **biblioteca de Javascript mais utilizada pelo mercado da programação web** e uma da biblioteca que tem uma comunidade extremamente ativa que pode ajudar a desenvolver e aprimorar seus conhecimentos. Voltado para o desenvolvimento de interfaces de usuário, o React é utilizado para construir aplicações web e mobile de forma eficiente e escalável, permitindo desenvolver aplicações reais que poderiam estar sendo desenvolvidas para o mercado, essa experiência permite a experimentação de diversas tecnologias, sendo uma delas a tecnologia de componentizar, ou seja: no React a aplicação é dividida em pequenos componentes, cada um com sua própria responsabilidade! Imagine uma caixa de construção de brinquedo, cada bloco é um componente que pode ser usado para construir uma casa, um carro, ou outro objeto qualquer,

no react acontece da mesma forma: cada componente no React pode ser usado para construir uma parte da interface do usuário, que, juntos, formam o todo, permitindo a organização dos códigos e promovendo uma melhoria ou um suporte ao código de forma rápida e objetiva. Vamos então iniciar a aprender o react de uma forma mais prática?

INICIANDO EM REACT

Veremos então como criar o primeiro projeto em react, utilizando o Vite. O React é uma biblioteca JavaScript bastante popular para criar aplicações web interativas e o Vite é um gerenciador de pacotes baseado em navegador que fornece uma forma rápida e fácil de começar a desenvolver com React. Antes de começarmos, é importante ter instalado Node.js e npm em sua máquina.

Após termos então o node e o NPM (O gerenciador de pacotes do node) instalados, **iremos instalar o Vite**, e, para isso é necessário abrir o terminal e execute o comando abaixo:

```
npm install -g @vite/cli
```

Este comando instalará o Vite em sua máquina de forma global, o que significa que você poderá usá-lo em qualquer projeto, após o instalarmos iremos iniciar um novo projeto react a partir do Vite, e, para isso iremos executar em nosso terminal o comando abaixo:

```
npm create vite@latest
```

Após executar o comando aparecerão algumas informações para a seleção, iremos utilizar as setas do teclado para mover e o enter para confirmar a escolha. Iremos primeiramente escolher o nome da nossa aplicação, pode-se optar por somente pressionar o enter para manter o nome padrão do projeto como “vite-project”. Após a escolha do nome teremos a escolha do framework. Selecione o react, pressione enter, selecione a variante **Javascript** e pressione enter. Com o projeto foi criado, o Vite criou uma nova pasta chamada "vite-project" (ou o uma pasta com o nome que você atribuiu ao projeto), dentro desta pasta você encontrará todos os arquivos necessários para que possamos começar a desenvolver a aplicação React.

Para acessar a pasta e instalar as dependências que o React e o Vite precisam para o projeto, executar o seguinte comando:

```
cd vite-project
```

Este comando irá permitir acessar a pasta do projeto, então, caso você tenha definido outro nome para o projeto execute o comando **`cd + nomedoprojeto.`**

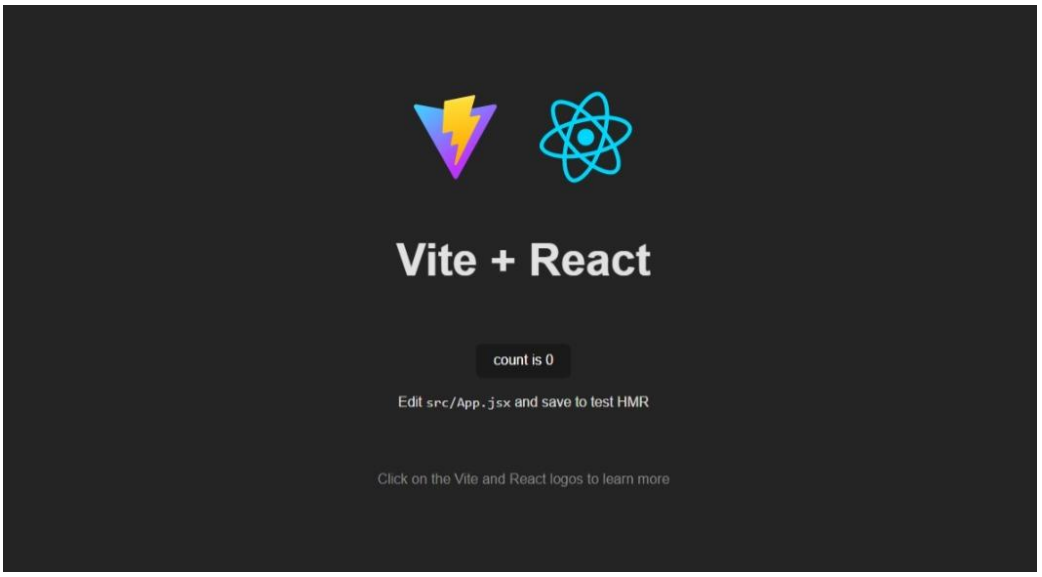
Agora dentro da pasta do projeto, iremos executar o seguinte comando para instalarmos as dependências:

```
npm install
```

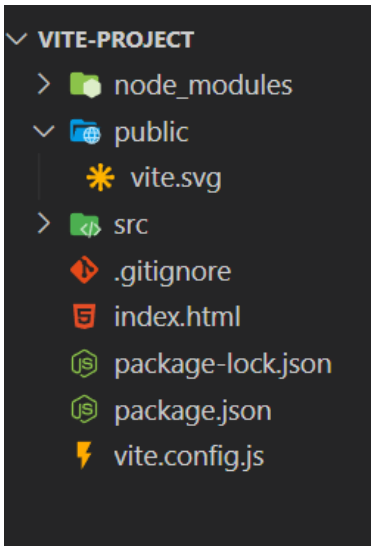
A execução deste comando pode demorar alguns minutos, após terminar (ainda no terminal), execute o seguinte comando para iniciar o servidor em modo de desenvolvimento:

```
npm run dev
```

Este comando iniciará um servidor local (na máquina), onde você poderá visualizar suas aplicações de React em tempo real enquanto desenvolve. Abra o seu navegador e acesse o endereço `"http://localhost:3000"` para ver sua aplicação, como no exemplo abaixo:



É possível notar que esse é um exemplo que o próprio react, juntamente com o vite nos proporciona, vamos então começar a desenvolver o nosso projeto e começar a criar o nosso próprio site, para isso iremos abrir o vscode dentro da pasta do projeto. Logo de início nota-se alguns arquivos e pastas, sendo eles:



Podemos notar que temos uma pasta ***node_modules***, e, dentro dela ficam armazenadas as dependências que o react e o Vite precisam para funcionar. Dentro da pasta ***public*** temos um arquivo chamado *vite.svg*, esta pasta contém arquivos que serão públicos, ou seja, serão arquivos utilitários que a aplicação utilizará. Em seguida, teremos mais alguns arquivos, sendo eles o ***.gitignore*** (este arquivo contém dentro dele todos os arquivos que o git irá ignorar quando for adicionar e realizar o *commit* de alterações) e o **index.html** (este arquivo irá conter toda a estrutura do nosso site). O react utiliza a tecnologia SPA, permitindo assim que a página nunca recarregue e sempre se mantenha em um mesmo arquivo HTML. Temos também os arquivos ***package-lock.json*** e ***package.json***, estes dois são arquivos de informações para o node e o NPM para que eles possam identificar as configurações, dependências e informações do projeto. Por último temos o arquivo ***vite.config.js*** que é um arquivo de configuração do Vite, dentro dele irá ficar toda e qualquer configuração que o Vite necessite para executar o projeto.

Se observarmos, temos uma pasta chamada **SRC** que informa que o projeto está seguindo a estrutura chamada **MVC**. Será dentro desta pasta que iremos encontrar todo o código presente em nosso projeto e todos os arquivos que iremos modificar para podermos construir o nosso próprio site.

Vimos então, até o momento, como instalar e utilizar o Vite para criarmos um novo projeto react; como instalar as dependências; como fazer para visualizar o projeto enquanto ele está sendo desenvolvido ou executando em modo de desenvolvimento; vimos também todos os arquivos presente no projeto e a função de cada um deles.

Glossário

React: Uma biblioteca *JavaScript* de código aberto para construção de interfaces de usuário, ele nos permite criar componentes reutilizáveis que se atualizam automaticamente quando os dados mudam.

CRA: acrônimo de *Create React App*, é uma ferramenta de código aberto desenvolvida pelo *Facebook* para simplificar o processo de configuração e inicialização de aplicativos React.

Vite.js: plataforma de desenvolvimento de aplicativos web de código aberto baseada em *JavaScript*, ele se concentra em velocidade, eficiência e simplicidade, oferecendo uma alternativa ao padrão de criação de aplicativos com ferramentas como o *Create React App*.

NPM: acrônimo de *Node Package Manager*, ele é um gerenciador de pacotes para o Node.js que nos permite instalar, compartilhar e gerenciar pacotes de bibliotecas de software.

MVC: acrônimo de *Model-View-Controller*, ele é um padrão de projeto de software que separa a aplicação em três camadas distintas, sendo elas: o modelo (os dados), a visualização (a interface do usuário) e o controlador (a lógica de negócios). Isso permite que os desenvolvedores trabalhem de forma independente nas diferentes partes do aplicativo, facilitando a manutenção e a evolução do software.

Para refletir

- Por que iniciamos nossa aplicação em modo de desenvolvimento?
- Por que utilizamos o **Vite**?
- Qual é a importância para um Desenvolvedor aprender React?

APROFUNDANDO

Até o momento vimos como podemos instalar e executar um projeto React e para que serve cada arquivo que o **Vite** criou, agora nosso foco será nos arquivos dentro da pasta **SRC** e para isso mergulharemos nas tecnologias que o react nos proporciona! Dentro da pasta **SRC** temos o arquivo **main.jsx,** observa-se que o arquivo tem a junção de **js** (Javascript) juntamente com o **x** (XmlElements), ou seja, este arquivo pode receber códigos Javascript juntamente com elementos xml, mais conhecido como elementos **HTML**, vamos então observar o que este arquivo está fazendo:

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Olhando para o código, podemos observar algumas coisas que ainda não tivemos o contato, como por exemplo o **import** , que é uma das declarações de módulos do JavaScript usada para importar funções, objetos ou valores de outros arquivos ou bibliotecas em nosso código, permitindo assim que possamos importar o código do pacote react e utilizar todos os métodos que ele nos proporciona no arquivo que importamos.

Sintaxe do import vem do **ES6 Modules** (a última versão do Javascript) e é bastante simplista, para executarmos um importe precisamos então declarar a palavra import juntamente com o nome que iremos atribuir ao módulo que estaremos importando, e, por fim utilizamos o *from* juntamente com a localização do módulo que estamos importando. Por exemplo: estamos importando o React do módulo react, agora toda vez que chamarmos o nome “React” estaremos utilizando os recursos disponibilizados pelo módulo que importamos. Temos mais três *importes* sendo eles, o **react-dom** (responsável em criar os elementos na tela), o arquivo **App** (este arquivo irá conter todo o código de nossa página) e por final temos um importe de um arquivo **css** (este importe funciona igualmente a tag link que utilizamos no HTML para *linkar* o css em nossa página). Logo após os *importes*, temos um código de onde usaremos o *ReactDOM* e criaremos no *root* (na raiz do documento) todos os nossos

elementos e, por fim, renderizar os elementos, criá-los e inserir o código. Vale lembrar que **renderizar** é permitir que o código seja interpretado como elementos HTML e que esses elementos possam aparecer na tela como tal.

Quando observamos o código do arquivo *main.jsx* notamos que todo o código da página fica dentro do arquivo *App.jsx*, devemos então observar o que há dentro desse arquivo:

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <div className="App">
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src="/vite.svg" className="logo" alt="Vite logo" />
        </a>
        <a href="https://reactjs.org" target="_blank">
          <img src={reactLogo} className="logo react" alt="React
logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.jsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </div>
  )
}
export default App
```

Como pudemos observar, foi importado o chamado de **useState** através do react. Este **useState** é chamado de **estado**, um estado de uma aplicação é um dado reativo, ou seja um dado que o react observa e sempre que houver alguma mudança, toda a página atualiza para receber esse valor atualizado. Isso acontece em tempo real! Por exemplo, se você for no site que está rodando em modo de desenvolvimento, e apertarmos no botão “count is 0”, nota-se que o valor 0 mudou para 1 e isso em tempo real. Isso ocorre porque, assim como o próprio nome da biblioteca “React”, o react é totalmente reativo ao seus estados, toda vez que um valor no estado muda a tela atualizará para poder receber esse novo valor, permitindo desenvolver aplicações que interagem com o usuário em tempo real. Após então o importe do estado temos também outros *importes*, sendo eles: o importe da logo do react (um arquivo svg dentro da pasta assets) e, como vimos anteriormente, temos o importe de um arquivo css (neste caso o *App.css)*.

Após os ***importes*** temos a nossa função de nome **App.** dentro dela temos uma constante que está criando e armazenando o nosso estado count, note que, temos então uma constante que está utilizando uma desestruturação de array, e, se notarmos, temos dentro dos colchetes o nome do estado, o nome da função que altera o estado _ por exemplo temos nossa const que está contendo o estado de nome count e depois da vírgula, ela está contendo então a função que altera o estado de nome ***setCount***, ou seja, um estado ele é somente legível. Não é possível atribuir um novo valor a ele de forma direta, por isso, para mudar o valor de dentro do estado é preciso então utilizar a função que muda o estado, em nosso exemplo esta função é a ***setCount.*** Nota-se na sintaxe da criação do estado continua, a constante está informando que esses dados, no caso o estado e a função, irão vir do **useState** (método que foi importado do react).

Depois do estado podemos observar que temos o nosso ***return***. Tudo dentro do return é o que estaremos enviando para a tela, note que até antes do return estávamos utilizando uma sintaxe Javascript e, dentro do return, iremos utilizar a sintaxe html para conseguirmos criar os nossos elementos e ao final teremos um arquivo export default App, ou seja, para que possamos importar este arquivo, antes temos que o exportar, neste caso estamos exportando por padrão a função App que sabemos estar sendo importada dentro do arquivo *main.jsx*.

Vamos então limpar tudo o que está dentro do return para que possamos criar o nosso próprio código, criando uma nova pasta chamada de *components*. Esta pasta irá ficar dentro da pasta SRC e a pasta *components* irá conter todos os nossos componentes (todas as nossas partes de código). Vamos então criar nosso primeiro componente e, para isso, iremos criar um arquivo dentro da pasta *components* chamado de *Button.jsx*, **atente-se que todos os componentes devem conter a primeira letra em maiúscula para que possa ser diferenciado**

de uma tag do HTML,

Após a criação do arquivo iremos então começar criando uma função dentro dele de nome **Button**. Dentro da função iremos colocar o nosso return e iremos retornar o elemento HTML `<button>Clique aqui!</button>`, agora precisamos exportar nosso componente para que possamos o importar em nosso App.jsx, e, para isso, iremos adicionar um export default Button. Note que sempre exportamos o nome da função, isso é importante porque o que é importado e exportado são funções, objetos, variáveis, dados, mas nunca arquivos, então, sempre que importamos e exportamos estamos importando ou exportando algum **recurso**, seja uma função, um dado ou um objeto, mas nunca um arquivo, sabendo disso, vamos dar uma olhada como o nosso componente Button.jsx ficou:

```
function Button() {
  return <button>Clique aqui!</button>;
}
export default Button;
```

Agora que temos o nosso componente Button pronto, precisamos colocar na tela e, para isso, iremos importa-lo dentro do arquivo *App.jsx:*

```
import { useState } from 'react'
import Button from './components/Button'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return ()
}

export default App
```

Após importarmos o componente iremos então chamá-lo como um elemento dentro do nosso return, para que ele possa aparecer na tela, para isso iremos adicionar uma *tag div* e dentro dela iremos então chamar o nosso componente:

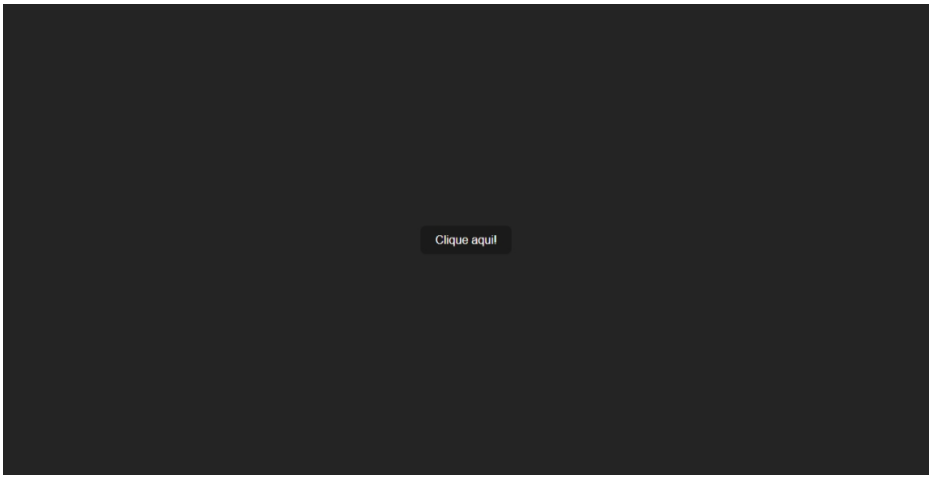
```
import { useState } from "react";
import Button from "./components/Button";
import "./App.css";

function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <Button />
    </div>
  );
}

export default App;
```

Agora que importamos e chamamos o nosso componente dentro do return, vamos ver como está o nosso site:



Já temos o componente na tela! Porém, quando clicamos no botão não acontece nada, o valor não sofre alteração! Vamos então utilizar aquele estado que deixamos em nosso *App.jsx*, e, para isso, iremos atribuir ao nosso componente de botão uma propriedade (prop).**Propriedade** é quando passamos valores através de componentes, e, para fazer isso, iremos dentro da tag do componente e iremos definir um nome de atributo, por exemplo, "*incremento*" (para simbolizar que sempre que executarmos o incremento ele irá executar a função que muda o estado incrementando, sempre mais um ao valor atual do estado), e abrir chaves, sempre quando nos abrimos chaves dentro do return de uma função em um arquivo JSX significa que iremos executar um código Javascript dentro do return, podemos abrir chaves para executar funções, ou para pegar um

dado, uma variável. Nesse caso, abriremos chaves e, dentro delas, iremos criar uma arrow function que poderá passar uma callback que está executando a função de nome *setCount* e adicionando mais um ao estado atual, feito isso, verifiquemos como ficou nosso arquivo *App.jsx:*

```
import { useState } from "react";
import Button from "./components/Button";
import "./App.css";

function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <Button incremento={() => setCount(count + 1)} />
    </div>
  );
}

export default App;
```

Agora que estamos passando uma propriedade, precisamos pegar e utilizar esta propriedade do outro lado, no caso dentro do nosso componente. Para tanto, iremos precisar definir o que queremos “pegar” dentro dos parênteses da função. Por exemplo: Queremos pegar a propriedade *incremento*? Para isso iremos realizar o seguinte:

```
function Button({incremento}) {
  return <button>Clique aqui!</button>;
}
export default Button;
```

Observa-se que dentro dos parênteses da função há uma abertura de chaves. Dentro dessas chaves estamos definindo o que queremos pegar, isso ocorre porque as *props* vem dentro de um objeto, e estamos desestruturando o objeto para pegarmos somente o que queremos utilizar, neste caso o *incremento*.

Agora então que conseguimos pegar a *prop* que nos foi passado do arquivo App.jsx, precisamos utilizá-la. Dentro do react, conseguimos utilizar os eventos através de atributos em elementos HTML. Para criarmos um ouvinte de evento de click para o botão, iremos adicionar o atributo **onClick** em nosso elemento *button*, e iremos então atribuir para ele a nossa prop incremento, lembre-se que sempre que formos utilizar ou chamar algo do Javascript dentro do *return* temos que utilizar as **chaves**, então o nosso código fica assim:

```
function Button({ incremento }) {
  return <button onClick={incremento}>Clique aqui!</button>;
}

export default Button;
```

Se você observar, o incremento foi somente “chamado” e não executado. Colocando os parênteses como uma função normal, realizou-se dessa forma para que o evento o executasse. Se fosse colocado os parênteses, o Javascript iria executar assim que o componente fosse lido, e esta não é a intenção.

Agora temos o nosso componente **Button** que está incrementando o nosso estado *count*, iremos precisar adicionar uma visualização do valor do estado em nossa tela para vermos se está realmente incrementando, para isso, iremos então dentro do *App.jsx* e adicionaremos o seguinte código dentro do return:

```
import { useState } from "react";
import Button from "./components/Button";
import "./App.css";

function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h1>{count}</h1>
      <Button incremento={() => setCount(count + 1)} />
    </div>
  );
}

export default App;
```

Note então que, adicionamos uma tag H1 que está utilizando as chaves para poder pegar o valor do estado *count*.

Se abrirmos a nossa aplicação em modo de desenvolvimento e o visualizarmos na web, iremos notar que temos então um título que é o valor do estado e um botão. Toda vez que clicamos no botão, o valor do título atualiza em tempo real, sempre incrementando o valor do estado de um em um.

Até aqui, abordamos estados, componentes, propriedades e eventos, além de como criar um projeto React usando o Vite. Esse não é um conteúdo simples para aprender e exige bastante prática! Por isso, é muito importante colocar a mão na massa. Depois de dominar o React, tudo ficará mais fácil e você poderá aplicar suas funcionalidades em suas aplicações.

Glossário

JSX: O JSX é uma extensão do JavaScript que permite escrever código HTML-like dentro do código JavaScript. Ele é amplamente utilizado no React, onde é possível definir componentes visuais como código JSX e renderizá-los na página.

ES6 Module: O ES6 Module é uma forma padrão de organizar e compartilhar código JavaScript, incluída na especificação ECMAScript 6 (ES6). Ele permite importar e exportar funções, variáveis e outros recursos entre diferentes arquivos JavaScript.

Pacote/biblioteca node: Um pacote ou biblioteca node é um conjunto de códigos JavaScript prontos para uso que podem ser instalados e usados em projetos Node.js, eles são gerenciados através do npm (Node Package Manager) e ajudam a resolver tarefas comuns, como conexão com banco de dados, validação de formulários, entre outras.

useState: O useState é uma hook do React que permite gerenciar o estado de um componente. Ele permite que o estado seja atualizado de forma dinâmica, ajudando a garantir a atualização da interface do usuário sempre que os dados forem alterados.

Para refletir

- Qual a maior vantagem em se utilizar o react?
- Por que “componentizar” partes do código? Qual é a vantagem que percebemos quando “componentizamos” uma parte de um código?

Atividade: Criando meu portfolio

Crie um novo projeto react, dentro dele, desenvolva uma página de portfólio contendo:

- Um título com seu nome;
- Uma imagem sua;
- Um parágrafo com sua BIO;
- Uma lista com suas tecnologias e habilidades.

Lembre-se de utilizar componentes para te ajudar a gerenciar todos os elementos que foram criados, utilize também outros recursos que vimos ao longo do e-book, como estados e propriedades.

Para ir além

- Para que você possa se aprofundar em **estados** e entender mais profundamente como ele funciona, recomendamos o seguinte site: <https://pt-br.reactjs.org/docs/hooks-reference.html#usestate>
- Para conhecer mais funcionalidades que o react pode oferecer, recomendamos a documentação oficial do react, onde pode-se encontrar tudo sobre o framework: <https://pt-br.reactjs.org/>

NUVEM DE PALAVRAS