



DESENVOLVIMENTO FRONT-END 2

IPOG

EDITORA IPOG

Todos os direitos quanto ao conteúdo desse material didático são reservados ao(s) autor(es). A reprodução total ou parcial dessa publicação por quaisquer meios, seja eletrônico, mecânico, fotocópia, de gravação ou outros, somente será permitida com prévia autorização do IPOG.

F383d Ferreira, Joelma de Moura.
Desenvolvimento Front-End 2. / Joelma de Moura Ferreira. – Goiânia:
Editora IPOG, 2024.
2,04 Mb; PDF

ISBN:

1. Front-End 2. Full-stack 3. JavaScript 4. Conceitos 5. React.

CDU: 004.45

IPOG

Instituto de Pós Graduação
e Graduação

Sede

Av. T-1 esquina com Av. T-55 N. 2.390 -
Setor Bueno - Goiânia-GO.

www.ipog.edu.br | (62) 3945-5050

Sumário

Sumário	3
APRESENTAÇÃO.....	4
OBJETIVOS.....	5
1.1 Front-End, Back-End E Full-Stack	6
1.2 Frameworks x Bibliotecas	6
1.3 A Biblioteca React	7
1.4. Single-Page Application e Server-Side Rendering.....	7
1.5. Introdução ao Javascript	9
1.5.1 Declaração de Variáveis	10
1.5.2 Estruturas Condicionais.....	10
1.5.3. Laços de Repetição.....	11
1.5.4 Template String	12
1.5.5 PASSAGEM DE PARÂMETROS.....	13
1.5.6 VALORES DEFAULT	13
1.5.7 FUNÇÃO ANÔNIMA.....	14
1.5.8 ARROW FUNCTIONS.....	15
2.1. Instalação do Ambiente.....	16
2.1.1 Instalação do Visual Studio Code (VSCode).....	17
2.1.2 INSTALAÇÃO DO Node.js.....	19
2.1.3 CRIANDO UM PROJETO COM O Vite	22
2.2 Pilares do React	26
2.3 Document Object Model (DOM)	27
COMPONENTES NO REACT.....	30
3.1 Conceito de Componentes	30
3.1.1 Componentes de Função (Function Component).....	31
3.1.2 Componentes de Classe (Class Components)	40
GERENCIAMENTO DE ESTADO E PROPRIEDADES DO REACT	48
4.1 Utilização de Propriedades (Props) No React.....	48
4.2 Utilização de Estados (States) no React	52
CICLO DE VIDA DOS COMPONENTES.....	55
5.1 Implementação de Rotas no React.....	55
5.2 Utilização de Hooks	55
5.3 Estilização de Componentes com Styled-Components	56
PARA FINALIZAR.....	57
Sobre o autora.....	58
Referências Bibliográficas	59

APRESENTAÇÃO

Seja bem-vindo ou bem-vinda à disciplina de Desenvolvimento Front-End 2.

Nesta jornada de aprendizado, vamos explorar os fundamentos essenciais que compõem o universo do desenvolvimento front-end. Iniciaremos abordando os conceitos de Front-End, Back-End e Full-Stack, além de discorrer sobre a distinção entre Frameworks e Bibliotecas. Na sequência adentraremos no estudo do JavaScript, linguagem muito utilizada por desenvolvedores front-end, para então aprofundarmos nosso conhecimento na biblioteca React (também conhecido como React.js ou ReactJS), compreendendo suas aplicações e conceitos importantes.

Estou confiante de que esta jornada será enriquecedora e desafiadora, e estou animada para vê-lo explorar cada tópico com curiosidade e entusiasmo. Esteja preparado para desafiar suas ideias preconcebidas, expandir seus horizontes e criar uma base sólida para o seu crescimento como profissional de desenvolvimento.

Vamos começar esta jornada juntos e aproveitar ao máximo. Prepare-se para uma experiência de aprendizado transformadora. Boa leitura e estudos!

Joelma de Moura Ferreira

OBJETIVOS

OBJETIVO GERAL

Capacitar o aluno a dominar a criação de interfaces de usuário dinâmicas (front-end) e responsivas usando React e JavaScript.

OBJETIVOS ESPECÍFICOS

- Ensinar os elementos pilares do React: componentes, props, state, rotas e hooks;
- Organizar código, estruturar componentes, gerenciar estado e manipular eventos;
- Criar componentes reutilizáveis.

Conheça como esse conteúdo foi organizado

Unidade Desenvolvimento Front-End;

2: Introdução ao React;

Unidade 3: Components no React;

Unidade 4 Gerenciamento de Estado e Propriedades no React;

Unidade 5: Ciclo de Vida dos Componentes.

UNIDADE 1

DESENVOLVIMENTO FRONT-END

Ao final dos estudos, você deverá ser capaz de: entender as diferenças entre Front-End, Back-End e Full-Stack e o que é a biblioteca React e sua aplicação.

1.1 Front-End, Back-End E Full-Stack

Front-end, Back-end e Full-stack são termos fundamentais no desenvolvimento de software, cada um desempenhando um papel crucial no funcionamento de uma aplicação web.

O Front-end, também conhecido como client-side, é a parte da aplicação que os usuários interagem diretamente. Ele engloba tudo o que é visível e acessível diretamente no navegador do usuário. Isso inclui o layout, design, conteúdo e interatividade. Tecnologias comuns usadas no front-end incluem HTML, CSS e JavaScript.

O Back-end, ou server-side, é a parte "invisível" de uma aplicação web que lida com a lógica de negócios, manipulação de dados, segurança e integrações com outros sistemas. Ele funciona fora do ambiente do navegador, geralmente em servidores, e é responsável por processar as solicitações do cliente, executar operações no banco de dados e enviar os resultados de volta para o front-end. As tecnologias comuns do back-end incluem linguagens de programação como Python, Java, Ruby, PHP e frameworks como Node.js, Django, Spring, entre outros.

Um desenvolvedor Full-stack possui habilidades tanto no front-end quanto no back-end. Isso significa que eles são capazes de trabalhar em todas as camadas de uma aplicação web, desde a interface do usuário até o servidor e o banco de dados. Eles têm uma compreensão abrangente de como as diferentes partes de uma aplicação se integram e são capazes de construir e manter um projeto web por conta própria.

1.2 Frameworks x Bibliotecas

Uma biblioteca é uma coleção de recursos organizados para ser utilizada na criação de aplicações, ou seja, é um conjunto de funções pré-escritas que os desenvolvedores podem usar em seus

projetos para realizar tarefas específicas, como manipulação de DOM, chamadas de API ou cálculos matemáticos. As bibliotecas são projetadas para serem modulares, o que significa que os desenvolvedores podem escolher quais partes usar e integrá-las em seus próprios códigos conforme necessário. Exemplos de bibliotecas populares incluem React e jQuery.

Já os frameworks possuem uma coleção de recursos organizados, mas também possuem um fluxo de trabalho ou uma estrutura pré-criados para serem seguidos, ou seja, é uma estrutura mais abrangente que fornece uma arquitetura de base para o desenvolvimento de aplicativos. Ele define uma estrutura para a organização do código, fluxo de controle e padrões de design, além de fornecer ferramentas e utilitários para facilitar o desenvolvimento. Os frameworks geralmente impõem uma certa maneira de fazer as coisas e tendem a ser mais opinativos em relação ao design e à estrutura do código, sendo assim mais potente que apenas uma biblioteca. Exemplos de frameworks populares incluem Angular, Django e Vue.js.

1.3 A Biblioteca React

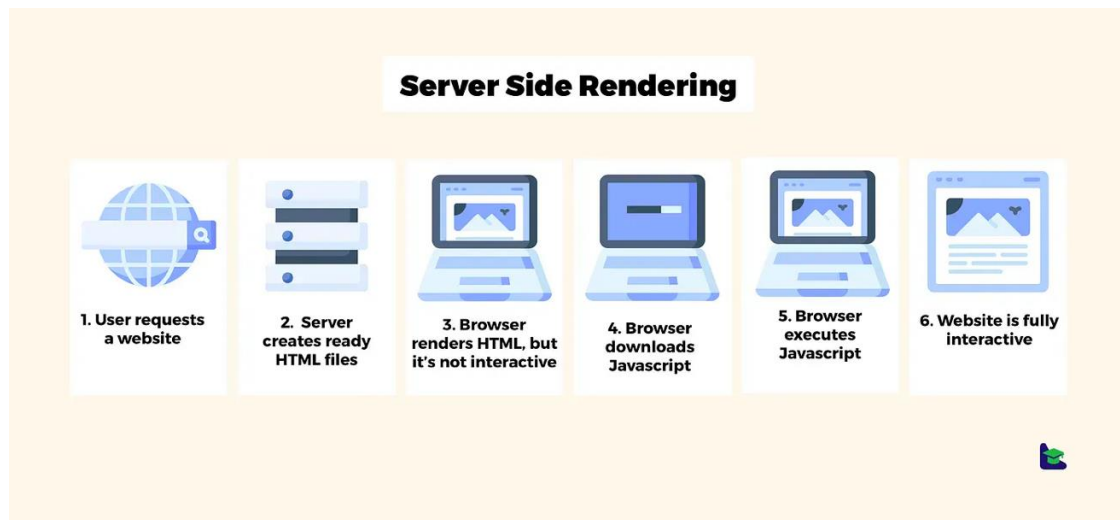
O React é uma biblioteca JavaScript para construção de interfaces interativas criada pelo Facebook e utilizada por grandes empresas como Netflix e Airbnb. Essas interfaces podem ser web (React JS), mobile (React Native) ou interfaces de realidade virtual (React 360). Importante reforçar que o React não é um framework, ele é uma biblioteca responsável por renderizar componentes reutilizáveis da camada de visualização. Vários frameworks utilizam o React, como por exemplo, o NextJS que é um framework que usa o React como base e permite integrar tanto o front-end quanto o back-end na própria aplicação, aumentando assim a potencialidade do React. Exemplos de frameworks populares baseados na biblioteca React são o NextJS, Gatsby e Electron.

1.4. Single-Page Application e Server-Side Rendering

As Single-page Applications (SPA) e Server-Side Rendering (SSR) são duas abordagens diferentes para a construção de aplicativos web, cada uma com suas próprias características e vantagens. A principal diferença entre SPAs e SSRs reside no momento em que o conteúdo é renderizado e como ele é entregue ao usuário. Basicamente, nas SPAs, o conteúdo é renderizado no navegador do usuário usando JavaScript, enquanto no SSR, o conteúdo é renderizado no servidor antes de ser enviado ao navegador.

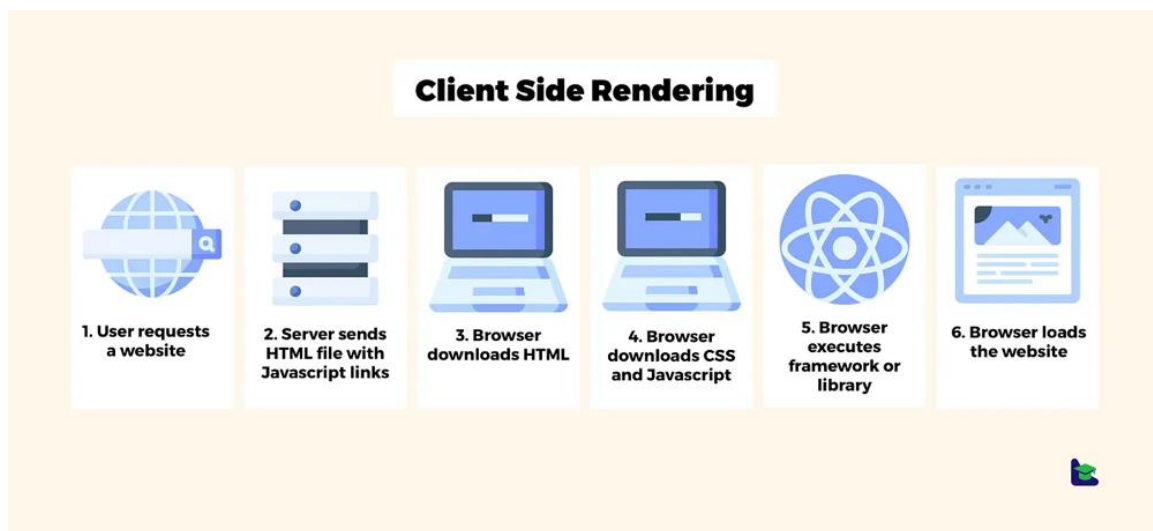
Server-Side Rendering é uma técnica na qual as páginas da web são renderizadas no servidor e

enviadas ao navegador do usuário como HTML completo. Isso significa que, em vez de carregar uma página em branco e preenchê-la com conteúdo usando JavaScript, o servidor gera o HTML completo com base no estado atual da aplicação e o envia ao navegador. Isso pode melhorar significativamente o tempo de carregamento inicial e a renderização do conteúdo, especialmente em dispositivos com recursos limitados ou conexões de internet lentas.



FONTE: <https://alexandreserra.com/articles/web-design-patterns>

Uma Single-page Application é uma aplicação web que carrega uma única página HTML inicial e, em seguida, atualiza dinamicamente partes da página conforme o usuário interage com a aplicação. Em uma SPA, todo o código JavaScript, CSS e conteúdo necessário é carregado de uma vez, geralmente durante a primeira visita do usuário ao site. Depois disso, as interações subsequentes do usuário são gerenciadas pelo JavaScript, que manipula as transições de página, solicitações de dados e atualizações de interface do usuário sem precisar recarregar a página inteira. Isso resulta em uma experiência de usuário mais fluida e responsiva.



Single-page Applications (SPA) e Server-Side Rendering (SSR) são padrões de renderização (rendering patterns), mas não são os únicos. Para conhecer outros padrões visite o site: <https://www.freecodecamp.org/news/rendering-patterns/>

Nesta disciplina construiremos páginas SPA, isso porque queremos tirar do servidor a responsabilidade de renderizar uma página. Por exemplo, suponha que precisamos mostrar na página uma lista de clientes que está armazenada no banco de dados. A aplicação Back-end não precisará ser responsável por montar a estrutura visual da aplicação. Ela fará toda consulta no banco de dados e montará uma estrutura de dados, como um json, por exemplo, e retornará isso para uma outra aplicação, o Front-end que converterá tudo isso para HTML, CSS e JavaScript.

O Back-end poderia ser escrito em Python, Java, C# ou qualquer outra linguagem ou framework. Já o Front-end, que é o que estamos aprendendo nessa disciplina, poderia ser escrito em Angular, Vue ou ReactJS, este último é o que vamos usar.

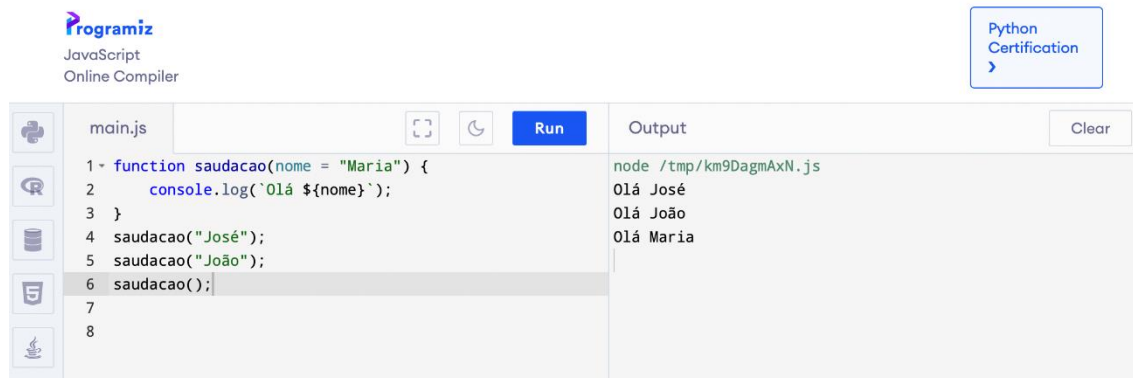
1.5. Introdução ao Javascript

JavaScript é uma linguagem de programação interpretada. É uma linguagem versátil, que permite a criação de aplicações tanto no lado do cliente quanto no lado do servidor, graças a ambientes como o Node.js.

JavaScript é amplamente utilizada para adicionar interatividade e dinamismo a páginas web, como validação de formulários, animações, manipulação do DOM (Document Object Model) e comunicação assíncrona com o servidor por meio de requisições AJAX (Asynchronous JavaScript and XML). Além disso, é uma das linguagens fundamentais para o desenvolvimento de aplicativos web modernos, sendo a base de muitos frameworks e bibliotecas populares, como Angular, React e Vue.js.

Para testar os códigos JavaScript de exemplos dessa unidade você pode usar um compilador online gratuito o Programiz JavaScript Online Compiler, disponível no endereço <https://www.programiz.com/javascript/online-compiler/>

Basta digitar o código e clicar em RUN para ele executar e mostrar o resultado no Console (Output) ao lado.



1.5.1 Declaração de Variáveis

A declaração de variáveis no JavaScript usa as palavras-chave "var", "let" ou "const". A palavra-chave "var" era a forma mais comum de declarar variáveis no JavaScript antes do ECMAScript 6 (ES6), enquanto "let" e "const" foram introduzidos no ES6 e são preferidos em situações mais modernas de desenvolvimento. A diferença principal entre "let" e "const" é que "let" permite que a variável seja reatribuída, enquanto "const" declara uma variável com um valor que não pode ser alterado.

```
var nome = "João";
let idade = 30;
const PI = 3.14;

console.log("Nome: ", nome);
console.log("Idade: ", idade);
console.log("Valor de PI: ", PI);
```

O comando `console.log` mostra o valor passado no console (tela) do ambiente.

1.5.2 Estruturas Condicionais

A estrutura básica de uma condicional do JavaScript é:

```
If () { <comandos> }
else {<comandos> }
```

Exemplo

```
var nome = "João";
let idade = 30;

if (idade >= 18) {
  console.log(nome, "é maior de idade.");
} else {
```

```
    console.log(nome, "é menor de idade.");  
}
```

1.5.3. Laços de Repetição

Laço for

O laço for é usado quando o número de iterações é conhecido. Sua estrutura básica é:

```
for (let i = 0; i < 5; i++) {  
    console.log("Número: " + i);  
}
```

Laço while

O laço while é usado quando o número de iterações não é conhecido de antemão e é baseado em uma condição.

```
let i = 0;  
while (i < 5) {  
    console.log("Número: " + i);  
    i++;  
}
```

Laço do-while:

O laço do-while é semelhante ao laço while, mas garante que o bloco de código seja executado pelo menos uma vez, mesmo que a condição seja inicialmente falsa.

```
let i = 0;  
do {  
    console.log("Número: " + i);  
    i++;  
} while (i < 5);
```

Escopo de variável

No JavaScript, quando uma variável é declarada dentro de um bloco utilizando a palavra-chave let ou const, ela possui um escopo de bloco. Isso significa que a variável só é acessível dentro desse bloco específico e não é visível fora dele. No exemplo fornecido:

```
flag = true  
var topico = "JavaScript";  
if (flag) {  
    let topico = "React";  
    console.log("Bloco: ", topico);  
}  
console.log("Global: ", topico);
```

Dentro do bloco do `if`, uma nova variável `TOPICO` é declarada usando `let` e é inicializada com o valor `"React"`. Essa variável `TOPICO` só é válida dentro do bloco do `if`, portanto, ao fazer `console.log` dentro do bloco, o valor de `TOPICO` será `"React"`. No entanto, fora do bloco do `if`, o `console.log` irá se referir à variável `TOPICO` declarada anteriormente com `var`, que possui escopo de função ou global (dependendo do contexto). Como a variável `TOPICO` declarada com `var` está em um escopo diferente daquela declarada com `let` dentro do bloco, elas são consideradas como variáveis diferentes, e a atribuição feita dentro do bloco não afeta o valor da variável `TOPICO` fora dele. Assim, o `console.log` fora do bloco irá imprimir o valor `"JavaScript"`, pois é o valor atribuído inicialmente à variável `TOPICO`.

1.5.4 Template String

Utilizamos o `template literals` em JavaScript para criar uma string que contém variáveis ou expressões embutidas. `Template literals` são uma forma mais flexível e legível de criar strings, permitindo a interpolação de variáveis ou expressões usando a sintaxe `${}` dentro de uma string delimitada por crase (```).

```
let nome = "José";
const mensagem = `Olá ${nome}`;
console.log(mensagem);
```

No exemplo, a variável `nome` é interpolada dentro da string utilizando `${}`. Isso significa que o valor de `nome` será inserido na string no lugar da expressão `${nome}`. Por exemplo, se o valor de `nome` for `"João"`, a variável `mensagem` receberá a string `"Olá João"`. O uso de `template literals` simplifica a concatenação de strings e torna o código mais legível e fácil de manter, especialmente ao lidar com strings longas ou com várias variáveis. Além disso, ao declarar a variável com `const`, garantimos que o valor de `mensagem` não será reatribuído, ou seja, ele permanecerá constante após a sua inicialização.

Declaração de funções

A declaração de função ou definição de função começa com a palavra-chave `function`, seguida pelo nome da função.

```
let nome = "José";

function saudacao() {
  console.log(`Olá ${nome}`);
}
```

Para invocá-la basta chamar o seu nome.

```
saudacao();
```

1.5.5 PASSAGEM DE PARÂMETROS

Podemos passar parâmetros e deixar a função mais genérica.

```
function saudacao(nome) {  
    console.log(`Olá ${nome}`);  
}  
saudacao("José");  
saudacao("João");
```

Um outro exemplo

```
function calcular(x1, x2, operador){  
    return eval(`${x1} ${operador} ${x2}`);  
}  
let resultado = calcular(3,2,"+");  
console.log(resultado);
```

O código apresentado define uma função chamada `calcular` que recebe três parâmetros: `x1`, `x2` e `operador`. A função utiliza o método `eval()` para avaliar uma string contendo uma expressão matemática construída a partir dos parâmetros fornecidos. Dentro da string, o valor de `x1`, o `operador` e o valor de `x2` são concatenados para formar a expressão a ser avaliada.

A função calculará $3 + 2$ e retornará o resultado, que é 5. Esse resultado é atribuído à variável `resultado`.

Este código demonstra como passar parâmetros para uma função em JavaScript e como a função pode usar esses parâmetros para realizar operações e retornar um resultado específico. No entanto, é importante notar que o uso de `eval()` deve ser feito com cautela, pois pode representar riscos de segurança se usado com entrada não confiável.

Um exemplo utilizando vetores

```
function calcular(valores, operador){  
    return eval(`${valores[0]} ${operador} ${valores[1]}`);  
}  
  
const valores = [3,2];  
let resultado = calcular(valores,"+");  
console.log(resultado);
```

1.5.6 VALORES DEFAULT

Os valores padrão (ou default) de uma função em JavaScript são valores que são atribuídos aos parâmetros da função caso não sejam fornecidos na chamada da função. Isso significa que você pode definir valores padrão para os parâmetros da função, garantindo que a função possa ser chamada com menos argumentos do que o número total de parâmetros definidos na função, sem causar erros. Os valores padrão são especificados na definição da função, utilizando a sintaxe de atribuição de parâmetros padrão:

```
function saudacao(nome = "Maria") {  
    console.log(`Olá ${nome}`);  
}  
saudacao("José");  
saudacao("João");  
saudacao();
```

1.5.7 FUNÇÃO ANÔNIMA

Uma função anônima é uma função que não possui um nome identificador associado a ela. Em vez de ser definida com um nome específico, ela é criada como uma expressão de função e geralmente é usada em contextos onde é necessária uma função temporária ou local.

```
let soma = function(x, y) {  
    return x + y;  
};  
console.log(soma(2, 3))
```

Neste código, a função anônima não tem nome entre a palavra-chave `function` e os parênteses `()`. Como precisamos chamar a função em algum momento, ela foi atribuída a uma variável.

- `let soma =`: Declaração de uma variável chamada `soma` usando a palavra-chave `let`.
- `function(x, y) { ... }`: Esta é a definição da função anônima. A função recebe dois parâmetros `x` e `y`.
- `{ return x + y; }`: Dentro do corpo da função, temos uma única instrução que retorna a soma dos parâmetros `x` e `y`.

Quando desejamos criar uma função e executá-la imediatamente após a declaração, pode usar a função anônima auto invocada (IIFE).

```
let resultado = (function(x1, x2, operador){  
    return eval(`${x1} ${operador} ${x2}`);  
})(3, 2, "+");
```



```
console.log(resultado)
```

(function(x1, x2, operador) { ... }): Esta é a função anônima auto invocada. Ela recebe três parâmetros: x1, x2 e operador. A expressão (3, 2, "+") após a definição da função são passados como argumentos para a função anônima. Isso significa que a expressão 3 + 2 será calculada imediatamente.

1.5.8 ARROW FUNCTIONS

As arrow functions são uma forma mais concisa de escrever funções anônimas em JavaScript. Elas fornecem uma sintaxe mais compacta e uma maneira mais simples de definir funções, especialmente para funções de retorno de uma única expressão

A sintaxe básica de uma arrow function é a seguinte:

```
const minhaFuncao = (parametro1, parametro2) => {  
  // corpo da função  
};
```

- **const (ou let):** A declaração de variável que define a função. Você pode usar const ou let dependendo da necessidade de retribuição do valor da função.
- **minhaFuncao:** O nome da função. Ele é opcional e pode ser omitido se você estiver atribuindo a função a uma variável ou usando-a como uma expressão.
- **(parametro1, parametro2):** Os parâmetros da função, entre parênteses. Pode ser zero ou mais parâmetros.
- **=>:** A seta (ou "arrow"), que substitui a palavra-chave function em uma função tradicional.
- **{ }:** O corpo da função, onde você coloca o código que a função executará.

Exemplos:

```
const dobrar = (numero) => numero * 2;  
let valores = (x, y) => { return x + y; };  
  
console.log(dobrar(2))  
console.log(valores(2,3))
```

UNIDADE 2

INTRODUÇÃO AO REACT

Ao final dos estudos, você deverá ser capaz de: apresentar os pilares fundamentais do React, bem como discorrer sobre o Document Object Model (DOM) e explicar como o React interage com ele para atualizar eficientemente a interface do usuário em resposta a mudanças no estado do aplicativo. Além de apresentar uma sequência de passos para instalar o ambiente de desenvolvimento necessário para executar as atividades.

2.1. Instalação do Ambiente

Para desenvolver Front-ends, ou até Back-ends, é preciso instalar um ambiente de desenvolvimento para o conjunto de tecnologia que pretendemos usar. Não existe uma forma única. Se acostume a testar tecnologias, ambiente, configurações diferentes. Depois escolha a que melhor se adapta ao propósito da sua aplicação.

Para esse curso precisamos escolher o conjunto de frameworks e bibliotecas que nos auxiliarão no desenvolvimento. Por questões didáticas escolhemos: ReactJS, Vite e Node.

Sobre o ReactJS já falamos um pouco. Agora falaremos sobre o Vite, vamos deixar o Node para depois. Lembrando, isso foi apenas uma escolha.

Primeiro, para executar qualquer linguagem precisamos de um compilador. Em React programamos em JavaScript, React é apenas o nome da biblioteca. JavaScript é compilado no próprio navegador, compilador padrão do JavaScript amplamente usado nos navegadores é o Babel. Sua função principal é converter o código escrito em JavaScript moderno (ES6+, TypeScript, etc.) em uma versão compatível com navegadores.

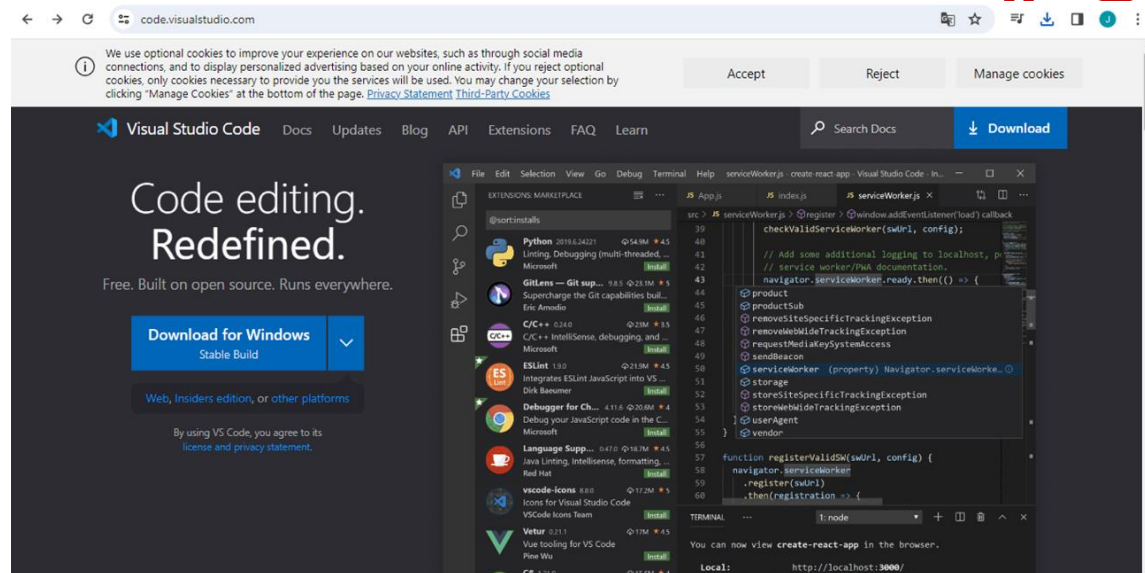
Uma aplicação em JavaScript pode ter vários arquivos ou módulos que se referenciam (um chama o outro). Para que o browser entenda isso é necessário usarmos o Webpack (Bundle) que é um empacotador de módulos para aplicações JavaScript. Sua principal função é organizar e gerenciar as dependências do projeto, além de empacotar o código em bundles otimizados para produção. O Webpack permite que os desenvolvedores dividam o código em módulos, definam dependências entre esses módulos e, em seguida, empacotem tudo em um ou mais pacotes de saída (bundles).

O Vite é uma ferramenta de desenvolvimento de Front-end que vai facilitar nossa vida quanto o uso do compilador e do empacotador. Desenvolvido pela equipe por trás do Vue.js, Vite utiliza uma arquitetura de desenvolvimento baseada em ESM (ECMAScript Modules) para oferecer tempos de compilação e recarga instantâneos. Em vez de empacotar todo o código do projeto, como o Webpack faz, o Vite fornece um servidor de desenvolvimento que entrega os módulos ES diretamente ao navegador, sem a necessidade de processamento adicional. Além disso, Vite é agnóstico em relação ao framework, o que significa que pode ser utilizado com qualquer framework JavaScript, como React, Vue.js, Svelte, entre outros.

Enquanto Babel e Webpack são ferramentas amplamente estabelecidas no ecossistema de desenvolvimento web, Vite é uma alternativa mais recente que visa fornecer uma experiência de desenvolvimento mais rápida e eficiente. Enquanto Babel se concentra na compilação de código JavaScript para garantir a compatibilidade com navegadores antigos, Webpack trata do empacotamento e otimização de recursos para produção. Por outro lado, Vite adota uma abordagem diferente, aproveitando a capacidade dos navegadores modernos de importar módulos ES diretamente, sem a necessidade de empacotamento adicional. Isso resulta em tempos de compilação e recarga muito mais rápidos durante o desenvolvimento.

2.1.1 Instalação do Visual Studio Code (VSCode)

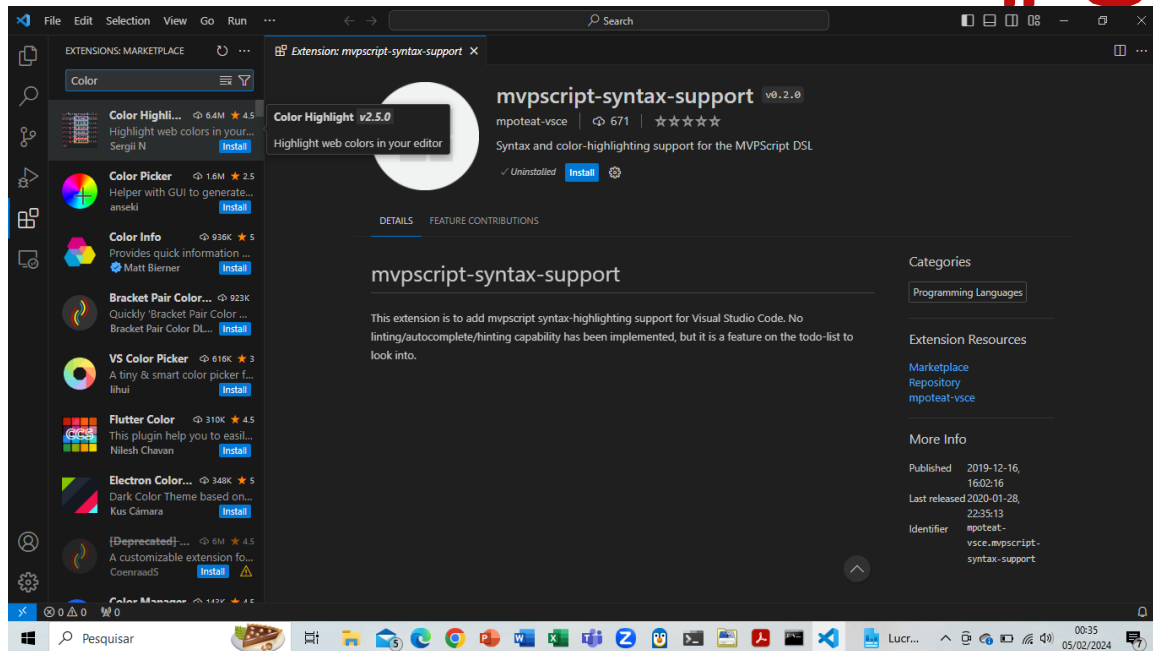
É importante salientar que todo o processo de instalação do ambiente descrito nesse Ebook é para Windows. No material complementar você encontra referências para outros sistemas operacionais. Vamos primeiro instalar o programa Visual Studio Code (VSCode). O VSCode é um editor de código desenvolvido pela Microsoft. Ele é conhecido por sua interface intuitiva, extensibilidade e suporte a uma ampla variedade de linguagens de programação. Para instalar o VSCode acesse o site oficial do Visual Studio Code em <https://code.visualstudio.com/>. Na página inicial selecione o botão de download do seu sistema operacional (Windows, macOS ou Linux).



Após o download ser concluído vá para a pasta onde baixou o arquivo e execute o instalador. Siga as instruções na tela para concluir a instalação. Normalmente, isso envolve aceitar os termos de uso, escolher o local de instalação e selecionar as opções de inicialização rápida, se disponíveis. Após a instalação ser concluída, você pode encontrar o Visual Studio Code em seu menu de aplicativos ou área de trabalho, dependendo das configurações do seu sistema operacional.

- **ColorHighlight:** Esta extensão destaca automaticamente as cores CSS em seu código, permitindo uma visualização rápida e fácil das cores utilizadas.
- **JSX HTML <tags/>:** Esta extensão melhora a formatação e a sintaxe do JSX, fornecendo realces de sintaxe aprimorados para elementos JSX que contêm HTML.
- **IntelliCode:** O IntelliCode é uma extensão que utiliza aprendizado de máquina para oferecer sugestões de código inteligentes enquanto você digita, economizando tempo e melhorando a produtividade.

Para instalar, abra o Visual Studio Code, vá para a aba de extensões (clique no ícone de quadrado com quatro quadrados menores à esquerda) e pesquise o nome da extensão. Clique em "Instalar".



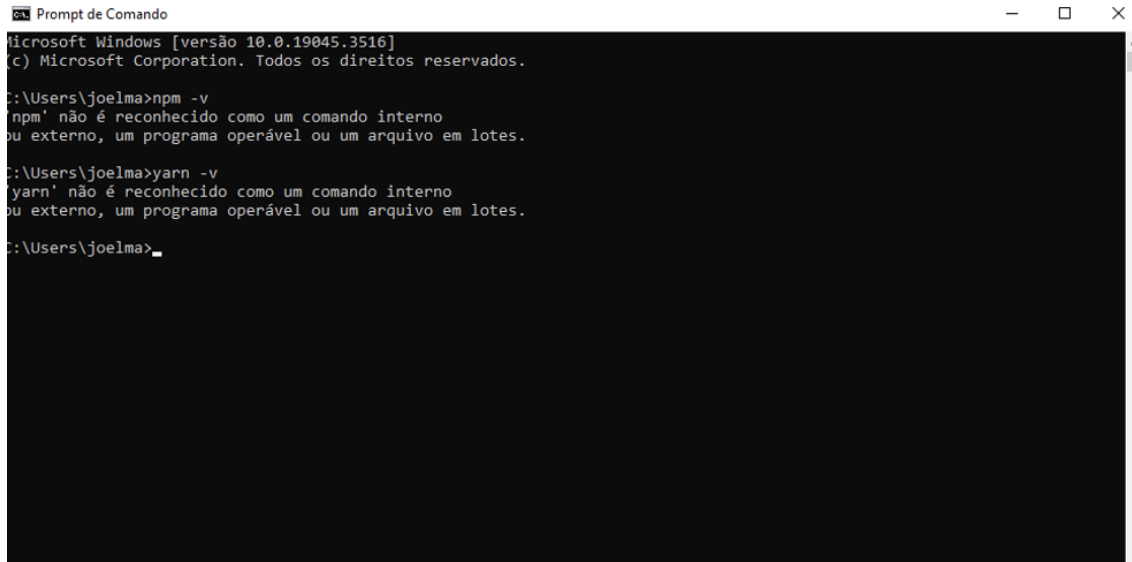
2.1.2 INSTALAÇÃO DO Node.js

Como vamos programar em JavaScript ou TypeScript é preciso instalar um ambiente que execute o código gerado. Instalar o Node.js é fundamental ao desenvolver aplicações com ReactJS, pois o Node.js fornece um ambiente de execução JavaScript fora do navegador, o que é essencial para o desenvolvimento e execução de aplicações React no ambiente de servidor. O Node.js, muitas vezes referido como Node.

O ReactJS é uma biblioteca JavaScript que geralmente é executada em um navegador da web. No entanto, durante o desenvolvimento, é crucial ter um ambiente de execução JavaScript no servidor para compilar, construir e executar a aplicação. O Node.js fornece esse ambiente, permitindo que você execute JavaScript no servidor e interaja com o sistema de arquivos, execute tarefas de compilação e inicie um servidor de desenvolvimento local.

Primeiro vamos verificar se o Node não está instalado na máquina. Abra o Terminal e digite o comando que mostrará a versão do Node instalado ou uma mensagem casos não tenha Node na sua máquina.

```
node -v
```



```
Prompt de Comando
Microsoft Windows [versão 10.0.19045.3516]
(c) Microsoft Corporation. Todos os direitos reservados.

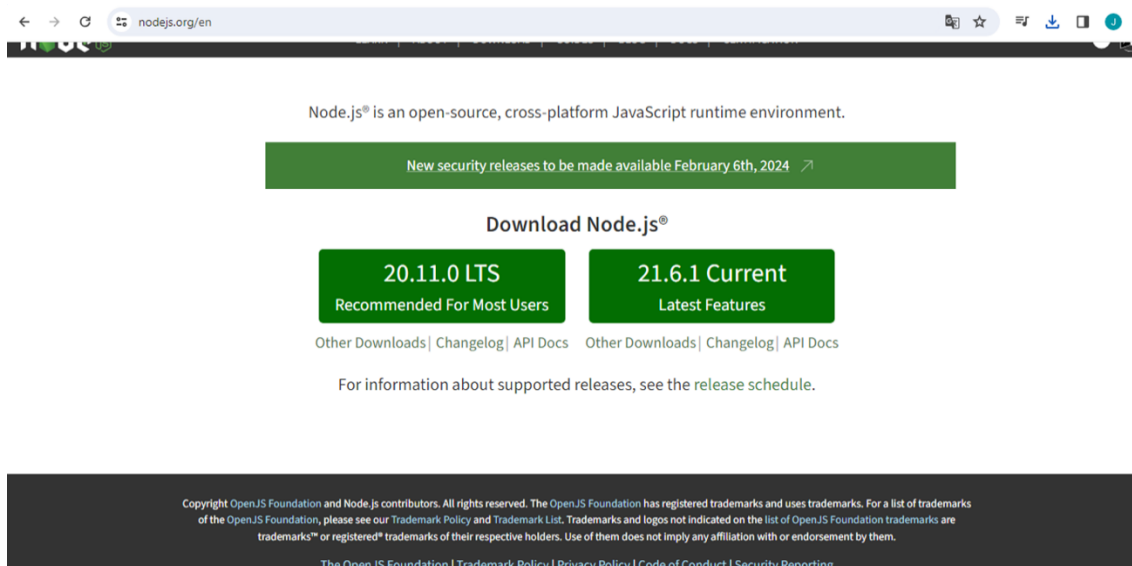
C:\Users\joelma>npm -v
npm' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.

C:\Users\joelma>yarn -v
yarn' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.

C:\Users\joelma>
```

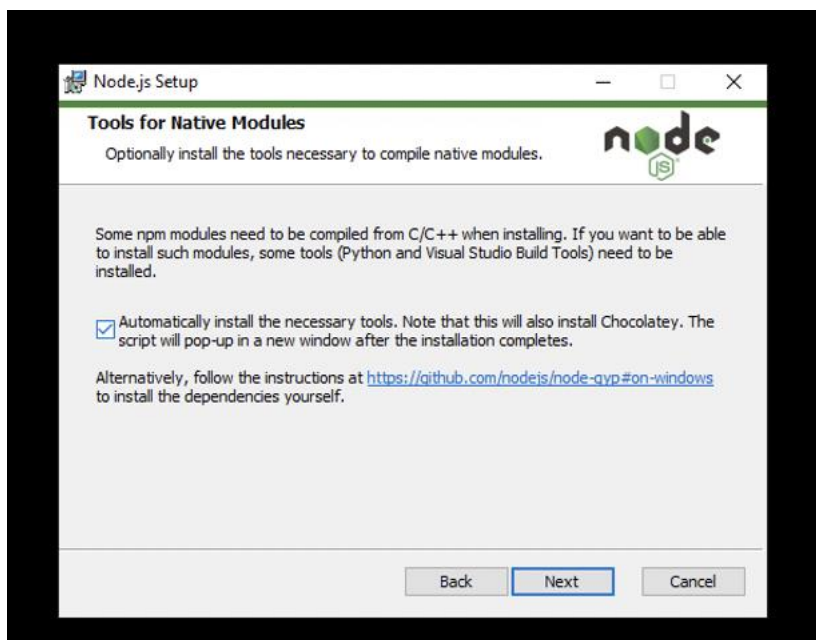
Para instalar o Node.js, siga os passos abaixo:

Acesse o site oficial do Node.js em <https://nodejs.org/>. Na página inicial, você encontrará o botão de download. O site detectará automaticamente o sistema operacional em que você está e oferecerá a versão adequada do Node.js para download. Escolha a opção Recommended For Most Users. Como as versões são sempre atualizadas por ser que o número da versão no momento que for executar seja outro.



Clique no botão de download para obter o instalador correspondente ao seu sistema operacional (Windows, macOS ou Linux). Após o download ser concluído, execute o instalador.

Siga as instruções na tela para concluir a instalação. Normalmente, isso envolve aceitar os termos de uso, escolher o local de instalação e selecionar as opções de instalação adicionais, se disponíveis.



Pode acontecer de durante a instalação o PowerShell abra e peça para confirmar algumas instalações através do ENTER.

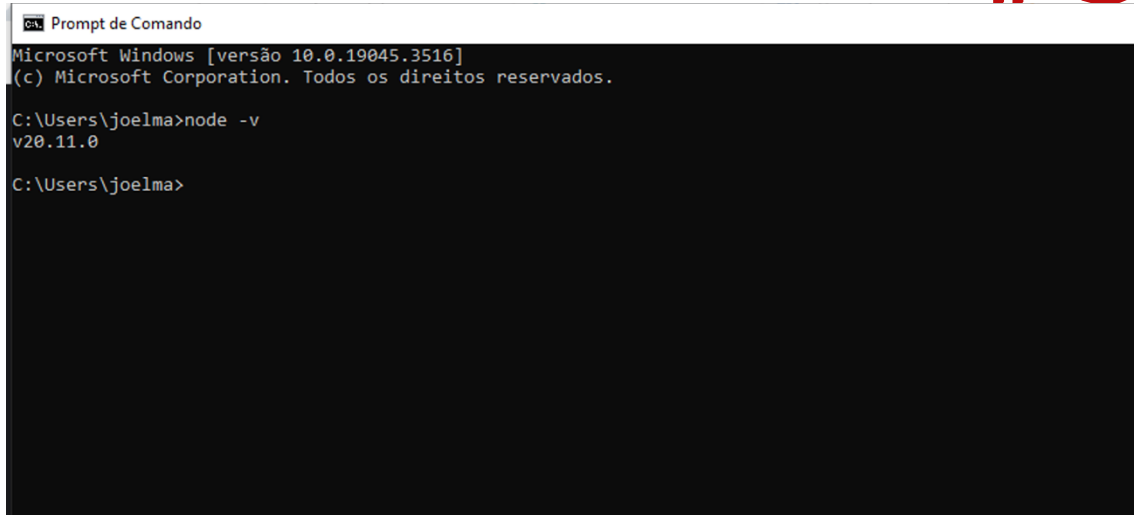
```

Administrador: Windows PowerShell
Installed/updated chocolatey-visualstudio extensions.
The upgrade of chocolatey-visualstudio.extension was successful.
Software installed to 'C:\ProgramData\chocolatey\extensions\chocolatey-visualstudio'
Progress: Downloading dotnetfx 4.8.0.20220524... 100%
dotnetfx v4.8.0.20220524 [Approved]
dotnetfx package files upgrade completed. Performing other installation steps.
Microsoft .NET Framework 4.8 or later is already installed.
The upgrade of dotnetfx was successful.
Software install location not explicitly set, it could be in package or
default install location of installer.
Progress: Downloading visualstudio-installer 2.0.3... 100%
visualstudio-installer v2.0.3 [Approved]
visualstudio-installer package files upgrade completed. Performing other installation steps.
The upgrade of visualstudio-installer was successful.
Software install location not explicitly set, it could be in package or
default install location of installer.
Progress: Downloading visualstudio2019buildtools 16.11.33.0... 100%
visualstudio2019buildtools v16.11.33 [Approved]
visualstudio2019buildtools package files upgrade completed. Performing other installation steps.
Downloading channel manifest
from 'https://aka.ms/vs/16/release/channel'
Progress: 100% - Completed download of C:\Users\joelma\AppData\Local\Temp\chocolatey\chocolatey-visualstudio.extension\C
hannelManifest_602808024.man (131.58 KB).
Download of ChannelManifest_602808024.man (131.58 KB) completed.
Downloading catalog manifest
from 'https://download.visualstudio.microsoft.com/download/pr/5a378c6a-0e85-4ebe-b6c8-59490e0c210b/bc09c4e89c890653c8f
2c6c58cf6933e87d46d8a97331836bc82b23c611a0807/VisualStudio.vsmn'
Progress: 100% - Completed download of C:\Users\joelma\AppData\Local\Temp\chocolatey\chocolatey-visualstudio.extension\C
atalog_1909569062.man (11.27 MB).
Download of Catalog_1909569062.man (11.27 MB) completed.
Downloading visualstudio2019buildtools
from 'https://download.visualstudio.microsoft.com/download/pr/5a378c6a-0e85-4ebe-b6c8-59490e0c210b/56af51de6d361198c39
81045cb6ac1a58a9d5bdda0e485b4304f5f6bf8fbf381/vs_BuildTools.exe'
Progress: 100% - Completed download of C:\Users\joelma\AppData\Local\Temp\chocolatey\visualstudio2019buildtools\16.11.33
\vs_BuildTools.exe (3.59 MB).
Download of vs_BuildTools.exe (3.59 MB) completed.
Hashes match.
Installing visualstudio2019buildtools...

```

Após a instalação ser concluída, você pode verificar se o Node.js foi instalado corretamente abrindo um terminal ou prompt de comando e digitando o seguinte comando:

```
node -v
```



```
Prompt de Comando
Microsoft Windows [versão 10.0.19045.3516]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\joelma>node -v
v20.11.0

C:\Users\joelma>
```

Se aparecer a versão significa que o Node foi instalado com sucesso.

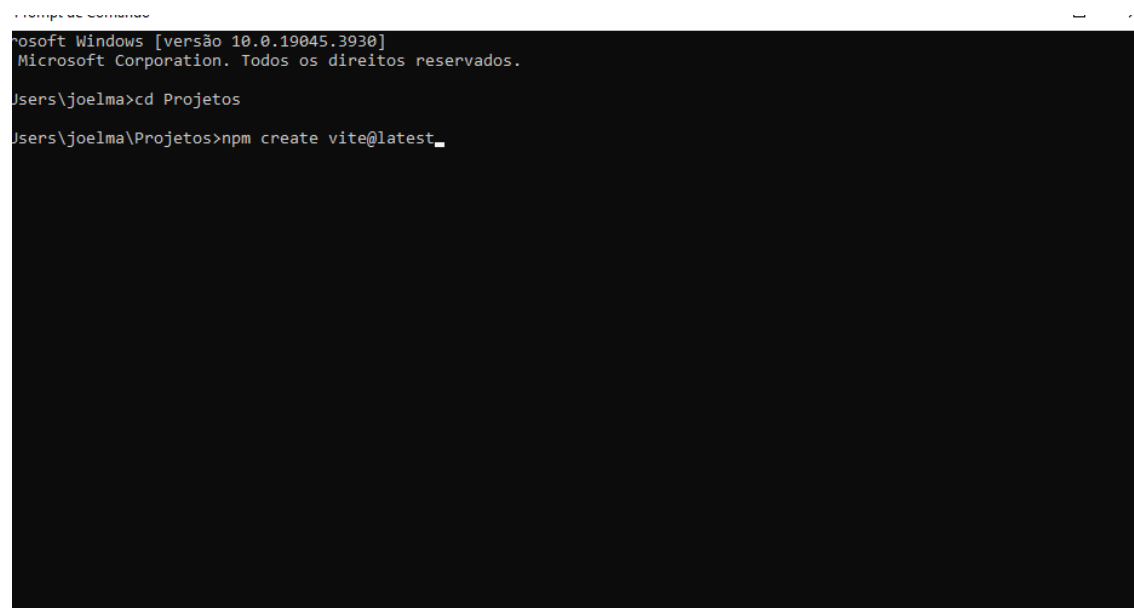
2.1.3 CRIANDO UM PROJETO COM O Vite

Crie uma pasta no seu computador para ser o repositório do seu projeto. Abra o Prompt de Comando do Windows (cmd) e vá para essa pasta. Por exemplo:

```
cd Documents\Projetos
```

Coloque o endereço de onde você criou a pasta. Depois digite:

```
npm create vite@latest
```



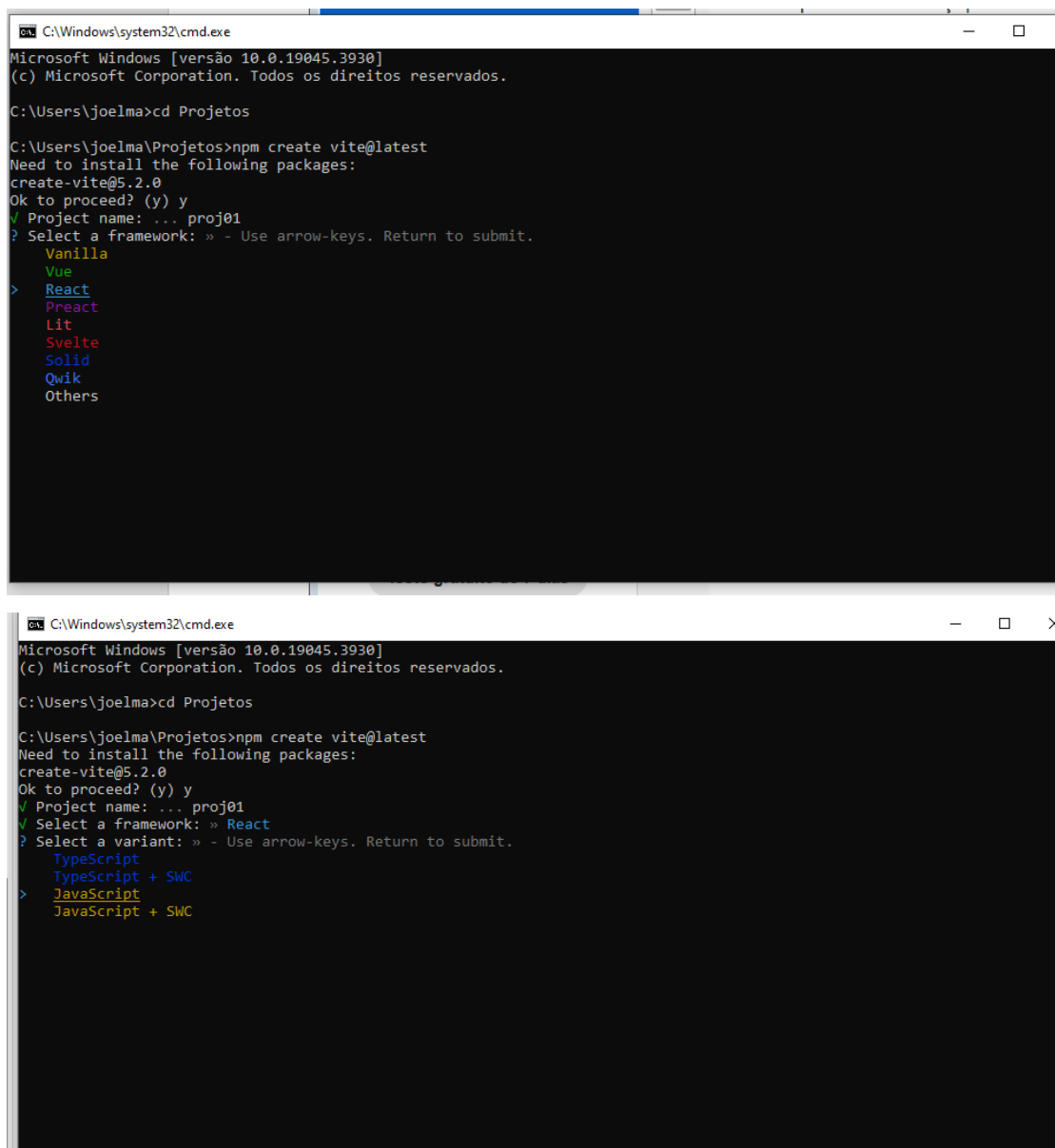
```
Prompt de Comando
Microsoft Windows [versão 10.0.19045.3930]
Microsoft Corporation. Todos os direitos reservados.

Users\joelma>cd Projetos
Users\joelma\Projetos>npm create vite@latest
```

O NPM ou Node Package Manager é o gerenciado de pacotes do Node.js, é como se fosse uma ferramenta de instalação.

O comando "npm create vite@latest" é usado para criar um novo projeto com Vite. Ao executar esse comando, o npm irá instalar a versão mais recente do Vite e usá-la para gerar a estrutura inicial do projeto.

Seguir os passos selecionando Y para proceder, dando um nome ao projeto (sugestão projteste, para ficar como os dos exemplos desse ebook), escolhendo React como framework e depois a linguagem JavaScript.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [versão 10.0.19045.3930]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\joelma>cd Projetos

C:\Users\joelma\Projetos>npm create vite@latest
Need to install the following packages:
create-vite@5.2.0
Ok to proceed? (y) y
✓ Project name: ... proj01
? Select a framework: » - Use arrow-keys. Return to submit.
  Vanilla
  Vue
>  React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Others

C:\Windows\system32\cmd.exe
Microsoft Windows [versão 10.0.19045.3930]
(c) Microsoft Corporation. Todos os direitos reservados.

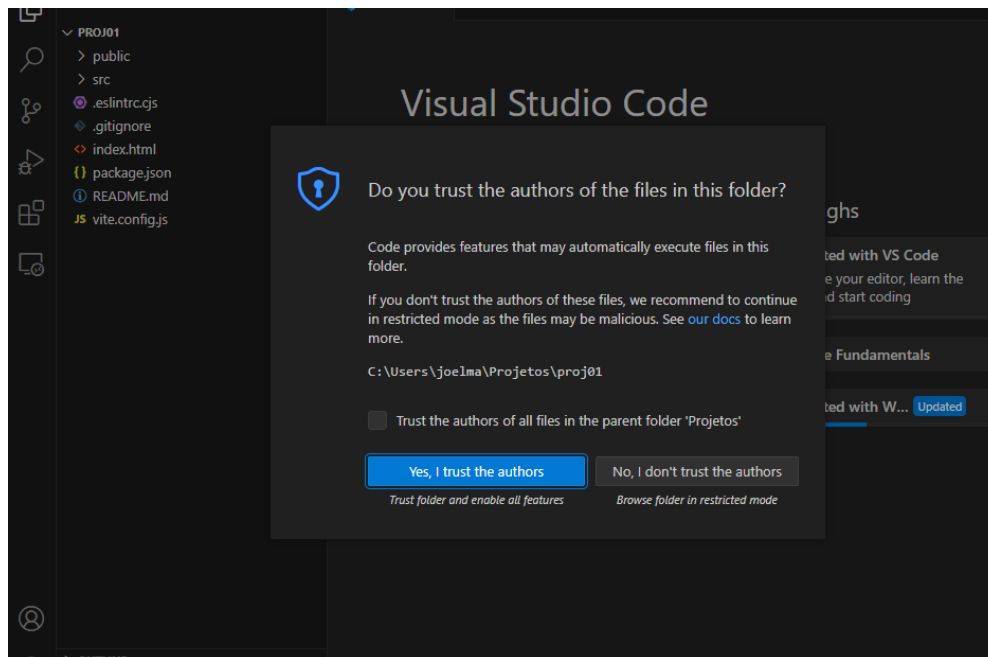
C:\Users\joelma>cd Projetos

C:\Users\joelma\Projetos>npm create vite@latest
Need to install the following packages:
create-vite@5.2.0
Ok to proceed? (y) y
✓ Project name: ... proj01
✓ Select a framework: » React
? Select a variant: » - Use arrow-keys. Return to submit.
  TypeScript
  TypeScript + SWC
>  JavaScript
  JavaScript + SWC
```

Depois do processo completado, abra o Visual Studio Code em escolha a opção File => Open Folder, escolha a pasta que com o nome do projeto que escolheu no passo anterior.

Caso aparecer essa janela abaixo apenas selecione, Yes, I trust the authors (Sim, eu confio nos

autores)



Agora é preciso instalar as dependências no projeto, todas as bibliotecas necessárias para ele executar. Abra o terminal do próprio VSCode, escolha na barra de menu Terminal => New Terminal.

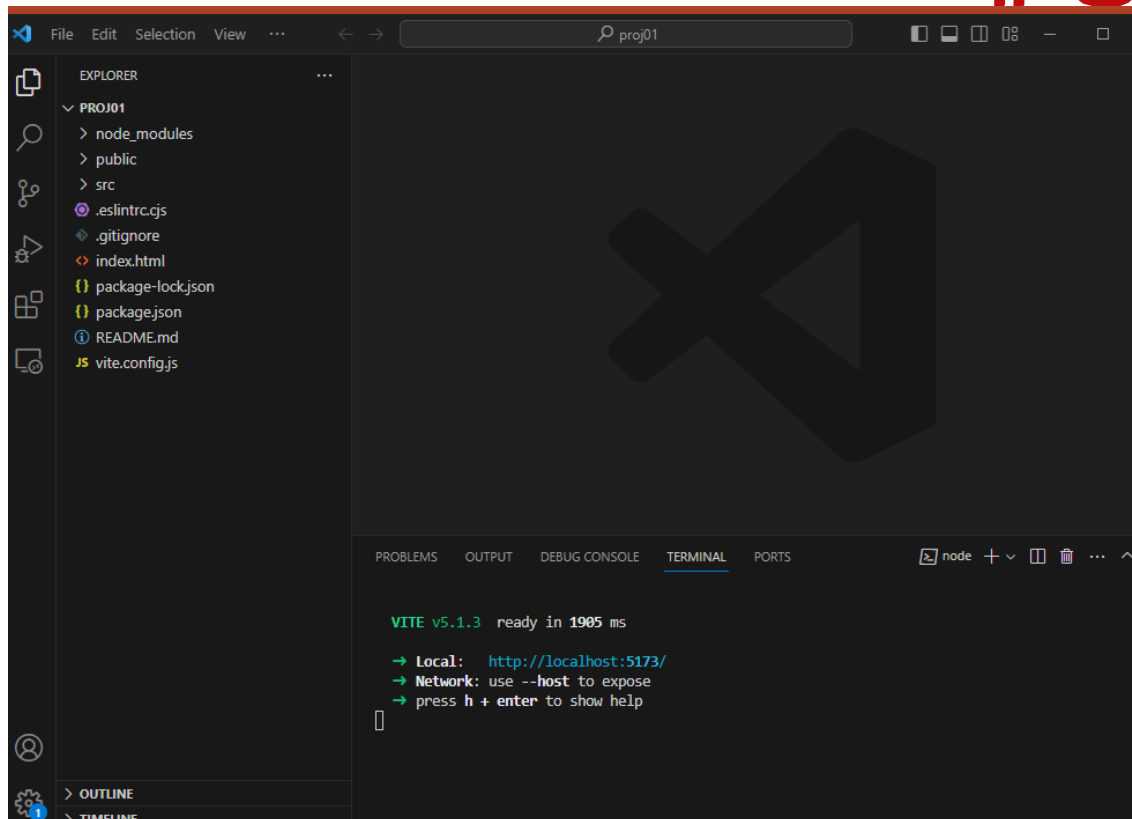
Agora digite:

```
npm install
```

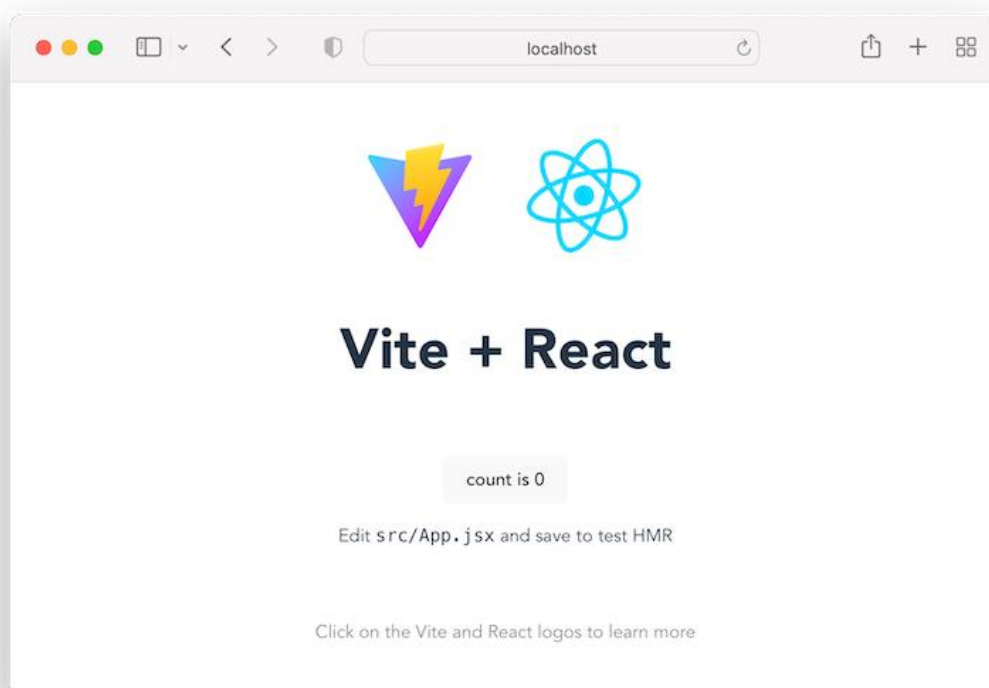
Vamos agora executar o template da aplicação criada para ver se a criação do projeto deu certo:

```
npm run dev
```

O comando "npm run dev" é usado para executar o script de desenvolvimento definido no arquivo "package.json" de um projeto Node.js. Geralmente, esse script é configurado para iniciar o servidor de desenvolvimento da aplicação, permitindo que você visualize e teste seu projeto localmente durante o desenvolvimento. Na imagem o endereço <http://localhost:5173> é onde o script será executado.

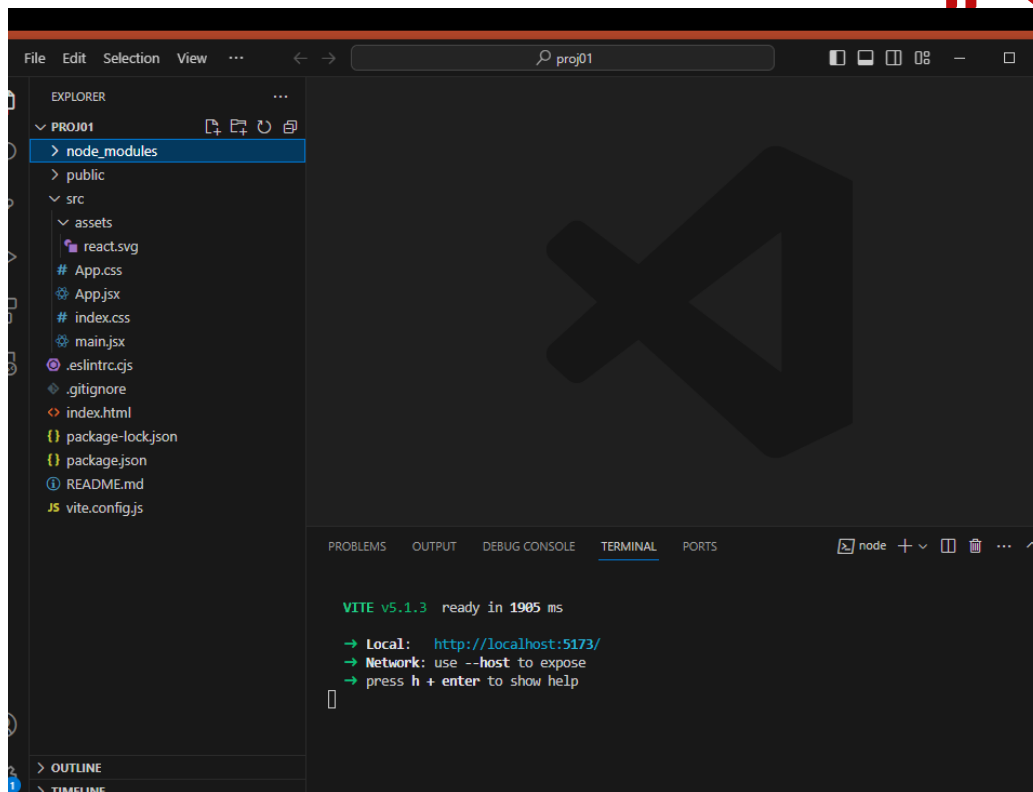


Abra o endereço no seu browser. Se aparecer a tela significa que a criação e execução do projeto foi bem sucedida.



Dica: Vamos usar como navegador sempre o Chrome.

A seguinte estrutura de pastas é criada:



2.2 Pilares do React

Quando criamos uma página (view) podemos dividi-la em partes como: cabeçalho, rodapé, menu, corpo etc. Essas partes podem ser definidas como “componentes” (User Interface), que são partes independentes e reutilizáveis. Podemos, por exemplo, reutilizar o cabeçalho em várias páginas diferentes.

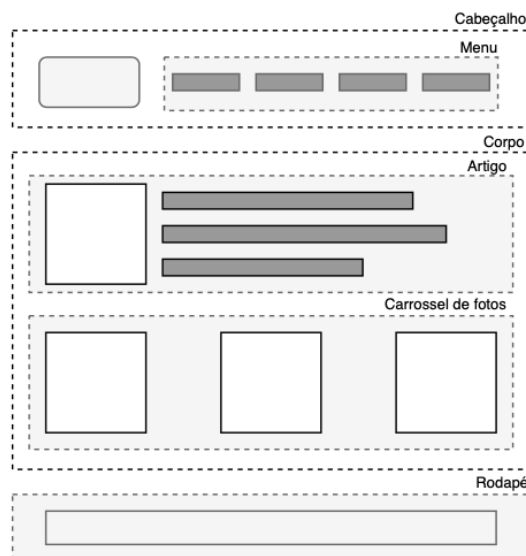


Figura: Estruturação de uma página em componentes.

Claro que é possível criar uma página como um único grande componente, mas isso não é aconselhável. O ideal é que uma página web seja dividida em componentes conforme a funcionalidade de cada um. Os componentes costumam estar relacionados a funcionalidade de uma

página, por exemplo, um calendário, uma galeria de fotos, uma área de envio de mensagens. Todos esses são exemplos de componentes que poderiam aparecer em diversas páginas de um site. Assim, seria interessante fazer esses componentes apenas uma vez e reutilizá-lo em diversos locais diferentes.

O papel principal do React é renderizar a página através da manipulação de seus componentes UI (User Interface). Um componente React é criado para ser encapsulado, reutilizável e combinável. Encapsulado porque cada componente possui a sua função própria, sendo que os outros componentes só precisam conhecer a sua interface. Reutilizável pois podem ser utilizados em diversas partes da aplicação sem precisar de modificação. E por fim combináveis, pois podemos compor um componente maior a partir de outros componentes que trabalharão em conjunto.

Os componentes React possuem um ciclo de vida composto de métodos bem definidos como montagem, atualização, desmontagem etc., que podem ser acessados em momentos distintos. É essa característica que permite ao React criar uma interface que não precisa ser recarregada inteira, toda vez que um componente mudar.

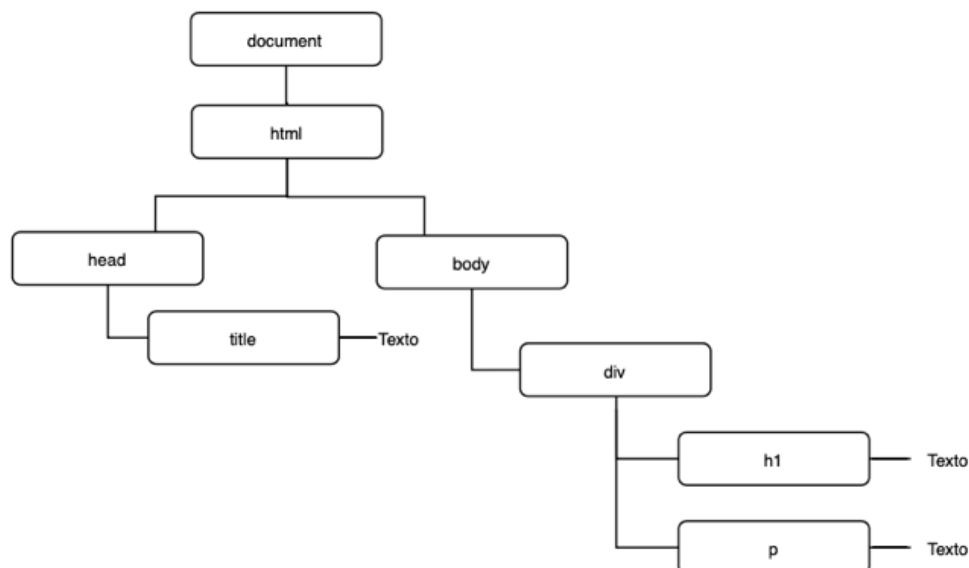
2.3 Document Object Model (DOM)

Document Object Model (DOM) é a representação de um documento (HTML e XML) carregado pelo navegador. Ele é composto por uma hierarquia de objetos que reflete a estrutura da página. É via DOM que podemos acessar, armazenar e manipular diferentes partes de um documento.

Vejamos um exemplo de uma página HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Título da Página </title>
  </head>
  <body>
    <div>
      <h1> Texto </h1>
      <p> Texto </p>
    </div>
  </body>
</html>
```

Ela pode ser representada pela seguinte estrutura DOM:



As propriedades, métodos e eventos são organizados em objetos que os desenvolvedores podem manipular através de comandos como, por exemplo:

- `document.getElementsByTagName(name);`
- `document.getElementById(id);`
- `document.createElement(name);`

O principal objeto é o ***document*** que representa o próprio documento, ele está no topo da hierarquia seguido pelo elemento root `<html>`.

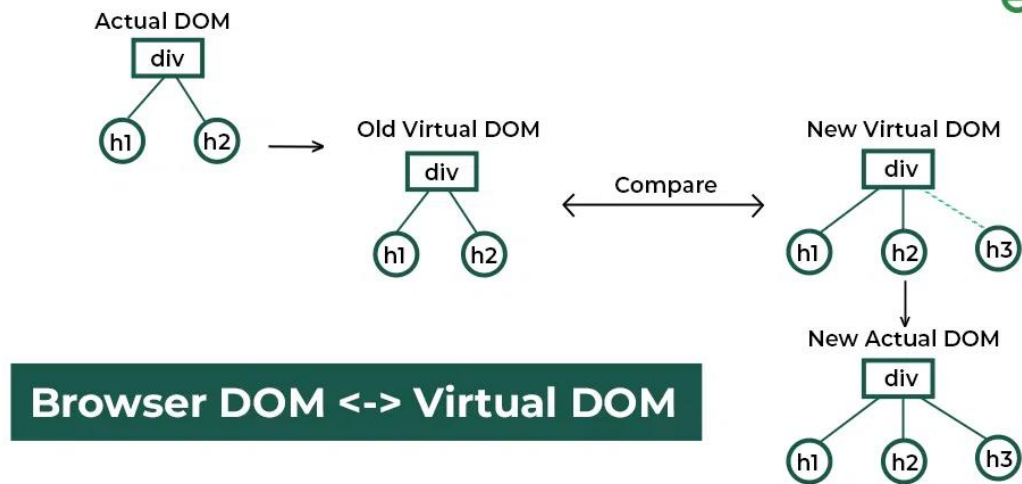
A manipulação direta do DOM não é algo tão simples. Se um programador que modificar um elemento HTML, por exemplo, mudar a cor do `<p>` ele deve acessar seguindo a hierarquia do DOM usando, por exemplo, um dos comandos `getElement...` para chegar no elemento desejado.

Sem contar que o acesso ao DOM leva um tempo, e cada alteração obriga o navegador a renderizar novamente a parte da tela afetada e isso traz problemas de performance.

Para resolver esse problema, bibliotecas como o React usa o Virtual DOM (VDOM). Ele é uma representação do DOM mantida em memória. As alterações são então sempre feitas em memória, para depois ser atualizado o DOM. Nessa abordagem o DOM é acessado menos vezes.

O React não altera o DOM diretamente, mas sim o DOM Virtual. Os componentes UI que são criados e manipulados pelo React ficam entre o DOM e a camada lógica. Essa camada intermediária é uma cópia do DOM e é chamada de DOM Virtual, sendo seu acesso mais rápido do que seria via API do DOM. Dessa forma, o React só precisa alterar no DOM aqueles componentes que foram modificados

e apenas quando todas as modificações forem finalizadas.



Fonte: <https://www.geeksforgeeks.org/reactjs-virtual-dom/>

UNIDADE 3

COMPONENTES NO REACT

Ao final dos estudos, você deverá ser capaz de: entender duas abordagens fundamentais para criar componentes em React: class components e function components. Desvendar as nuances de cada tipo de componente, entender suas características distintas e descobrir como aproveitar ao máximo suas funcionalidades.

3.1 Conceito de Componentes

As interfaces web são criadas a partir de um conjunto de componentes. Quando criamos uma página ela pode ser dividida em uma estrutura hierárquica de componentes que podem representar desde um simples botão até estruturas maiores como cabeçalho, rodapé, navegação e tudo mais que possa, inclusive, ser reutilizado.

A figura abaixo é um exemplo de divisão de uma página em componentes. Vemos os componentes: cabeçalho, menu, corpo, artigo, carrossel e rodapé. Para montar essa página usando o React poderíamos criar uma estrutura hierarquizada de componentes.

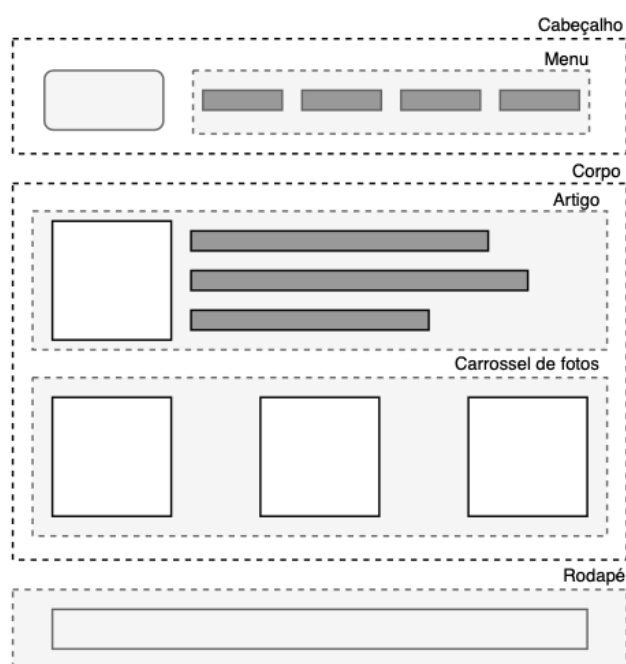


Figura: Estruturação de uma página em componentes.

Mas, os componentes do React podem ser mais poderosos do que isso e cuidar do roteamento, ou da formatação dos dados, por exemplo. Os componentes aceitam dados de entrada e retornam os

elementos React que apareceram na tela.

Uma das características principais dos componentes é a garantia da sua consistência. Um componente pode receber dados externos (props) e manter seu estado interno (states). A partir dos dados e estado é capaz de atualizar uma UI. O React monitora as mudanças que ocorrem em um componente e sempre que a entrada para um componente é alterada, ele é renderizado novamente, garantindo assim a atualização da interface.

Os componentes React podem ser construídos como funções (function components) ou classes (class components) JavaScript.

3.1.1 Componentes de Função (Function Component)

Os function components são criados usando funções do JavaScript. Eles são utilizados quando não se pretende gerenciar o estado do componente, sendo mais importante a apresentação dos dados, por isso são identificados também como componentes funcionais sem estado.

Possuem a seguinte aparência:

```
function MyButton() {  
  return (  
    <button>I'm a button</button>  
  );  
}
```

Uma vez criado um componente, esse representa um botão, ele pode ser utilizado em outras partes do site simplesmente declarando seu nome como se fosse uma tag HTML:

```
export default function MyApp() {  
  return (  
    <div>  
      <h1>Welcome to my app</h1>  
      <MyButton />  
    </div>  
  );  
}
```

MyApp seria uma página que possui uma tag <H1> e também um botão, mas não um botão normal do HTML, mas sim o componente <MyButton /> que foi definido como um componente React. Essa seria a saída no Browser:

Welcome to my app

I'm a button

Observação: nesse ebook utilizamos alguns códigos do site oficial do React. Para saber se a fonte é esse site, todos os códigos em inglês virão do site: <https://react.dev/learn>, como é o caso do exemplo acima.

VAMOS PRATICAR:

Vamos praticar e criar nosso primeiro projeto React.

Espero que você já tenha instalado todos os softwares necessários. Caso não tenha ainda volte à unidade 2 e siga os passos de instalação e depois execute todos os procedimentos da seção 2.3.1 para criar um projeto React+Vite.

Não esqueça depois de abrir o VSCode executar o:

```
npm install
```

Uma vez que o projeto esteja criado e rodando, vamos agora remover os arquivos desnecessários

- Remova alguns dos arquivos que estão na pasta src que não usaremos inicialmente:
 - App.css
 - Index.css
- Remova o arquivo react.svg da pasta assets.

Abra o arquivo **main.jsx** e retire a linha do `import './index.css'`. E também altera `import App from './App.jsx'` para `import {App} from './App.jsx'` colocando as `{}`. O código ficará assim:

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import {App} from './App.jsx'
```

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


Abra o Arquivo **index.html** e mude a tag `<title>Vite + React</title>`, para `<title>Meu Primeiro Projeto React </title>`

Abra o arquivo **App.jsx** e retire as linhas com os comandos

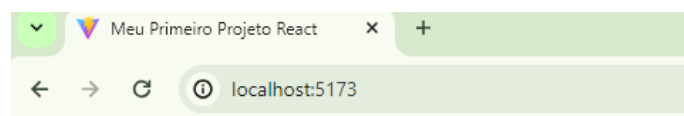
```
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
```

Apague também todo o código que está dentro do RETURN e trocar por uma mensagem "Olá, Mundo!". No final o arquivo App.jsx deve ficar EXATAMENTE assim:

```
export function App() {
  return (
    <h1>Olá, mundo!</h1>
  )
}
```

Mais uma vez, observe que foi retirado a parte `export default App` e também foi modificado o cabeçalho da função `export function App()`. O que foi feito foi a retirada do export default e colocado no function.

Visualize no browser como ficou agora a página.



Olá, mundo!

(ASSISTA AS VIDEO-AULAS PARA ENTENDER O PROCEDIMENTO COMPLETO).

Em um projeto React os arquivos Index.html, Main.jsx e App.jsx estão interconectados para fornecer a estrutura básica e funcionalidade de uma aplicação web.

O index.html é o ponto de entrada principal da sua aplicação web.

Ele contém a estrutura HTML básica, como tags `<html>`, `<head>` e `<body>`.

O `Index.html` é o arquivo que será carregado inicialmente pelo servidor e que contém um elemento no qual a aplicação React será montada.

```
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
```

Perceba que dentro da tag `<body>` é informado que o arquivo a ser carregado é o `Main.jsx`, e existe uma `<div>` vazia de `id=root`

```
<div id="root"></div>
```

Será dentro dessa tag que a página que você construir com os componentes será renderizado.

O `Main.jsx` é o arquivo de entrada JavaScript para sua aplicação. Ele é responsável por inicializar a aplicação React e montá-la no DOM. Além disso, no `Main.jsx`, você pode configurar outras bibliotecas, como roteamento (por exemplo, React Router), e quaisquer outras configurações globais necessárias para sua aplicação.

A tag `<App />` no `Main.jsx`, importa o componente `App`, definido no arquivo `App.jsx`, e o renderiza dentro da `div` com o `id root` no `Index.html`.

O `App.jsx` é o componente raiz da sua aplicação React, que contém a estrutura básica da interface do usuário da sua aplicação e pode ser composto por outros componentes menores. Dentro do `App.jsx`, você pode definir o layout principal da sua aplicação, lidar com o roteamento, e outros aspectos que são comuns a toda a aplicação.

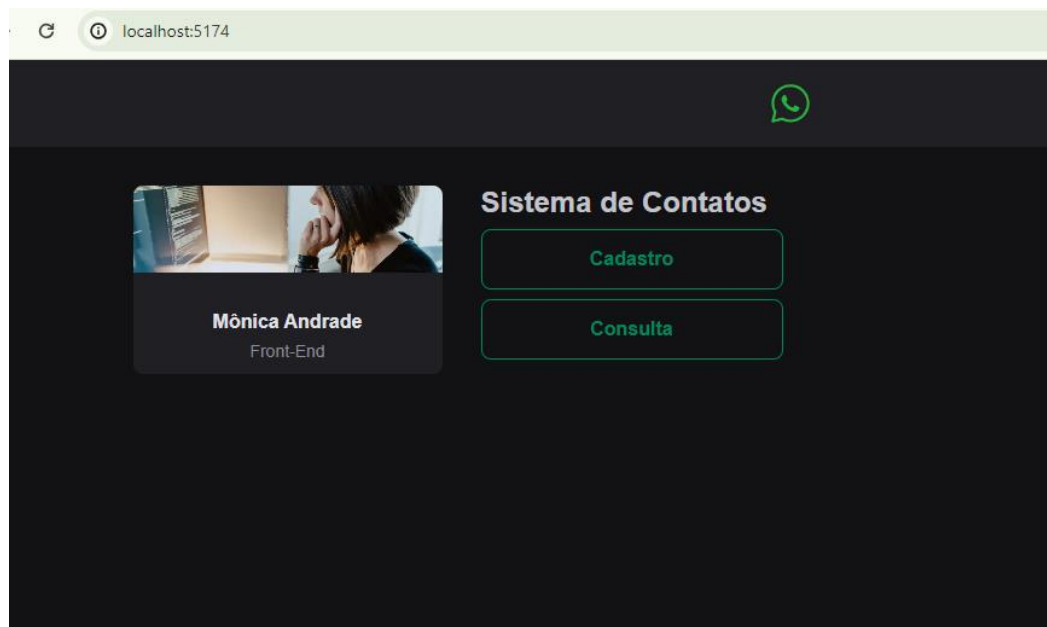
O `App.jsx` tem apenas uma função Javascript que retorna HTML. Isso é um componente.

```
function App() {
  return (
    <h1>Olá, mundo!</h1>
  )
}
```

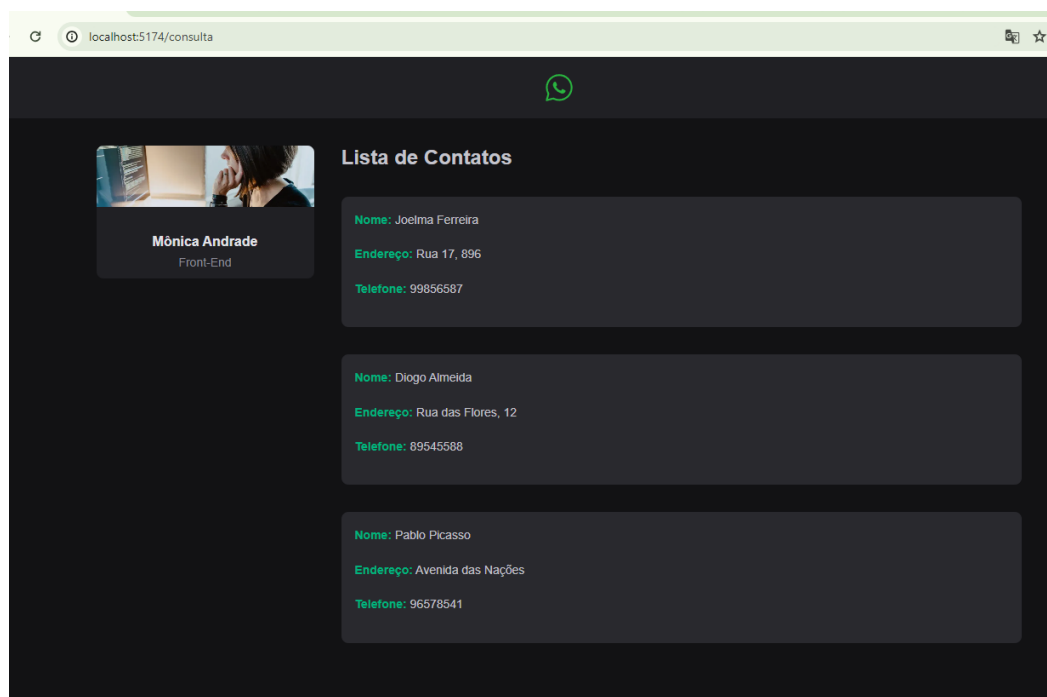
Todos os componentes do React devem ter a extensão `jsx` que é a indicação de um arquivo JavaScript que contém HTML.

ESTUDO DE CASO:

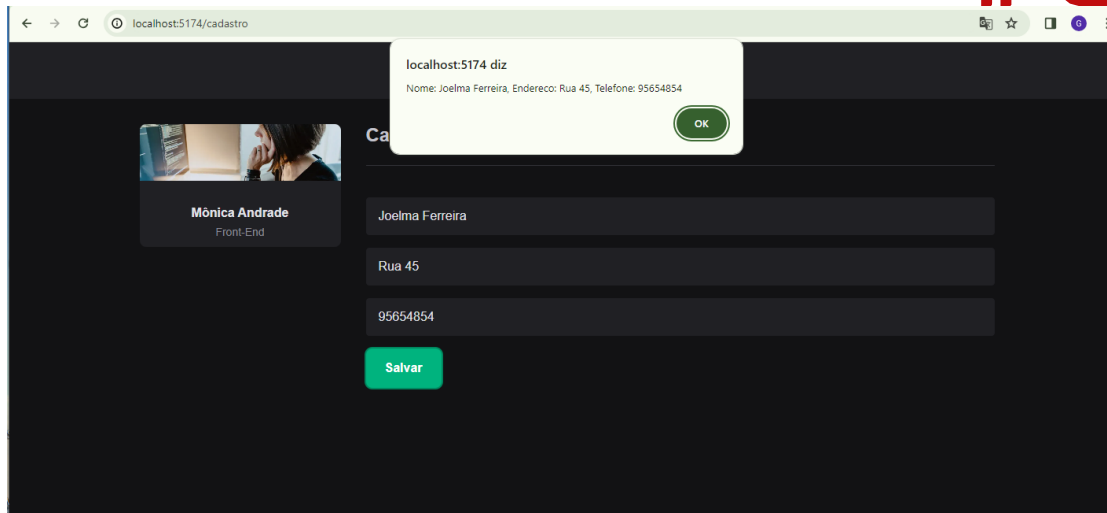
Vamos começar a criar uma aplicação web que norteará todo o restante dessa disciplina. Criaremos o Front-End de um sistema para cadastrar contatos. Teremos uma Home com uma parte com perfil e foto e na parte principal da página dois links para um para a tela de cadastro e outro para consulta.



A tela de consulta mostra a lista de contatos



A tela de cadastro para cadastrar os dados, mas como estamos fazendo apenas o Front não registraremos os dados em banco de dados, apenas o recolhemos e mostraremos em um **alert**.



(ASSISTA AS VIDEO-AULAS PARA ENTENDER O PROCEDIMENTO DE CONSTRUÇÃO DO ESTUDO DE CASO, EM APOIO A DESCRIÇÃO DO EBOOK)

Crie um novo projeto React, conforme procedimento anterior. Dê o nome de **contatos**. Abra o VsCode e execute no Terminal na sequência:

```
npm install
```

```
npm install react-router-dom axios
```

```
npm run dev
```

Apague os arquivos desnecessários. Não esqueça de mudar o title do arquivo Index.html

```
<title>Sistema de Cadastro</title>
```

Agora crie uma pasta chamada ***components*** e dentro dela quatro arquivos:

- Cadastro.jsx
- Consulta.jsx
- Contato.jsx
- Home.jsx

O código de cada um será:

```
export function Cadastro() {
  return <h2>Cadastro</h2>;
}
```

```
}

```

```
export function Consulta() {
  return (
    <div>
      <h2>Lista de Contatos</h2>
    </div>
  );
}
```

```
export function Contato() {
  return (
    <div>
      <div>
        <p>Nome: Joelma Ferreira </p>
        <p>Endereço: Rua 17, 896 </p>
        <p>Telefone: 99856587 </p>
        <br/>
        <p>Nome: Diogo Almeida </p>
        <p>Endereço: Rua das Flores, 12 </p>
        <p>Telefone: 89545588 </p>
        <br/>
        <p>Nome: Pablo Picasso </p>
        <p>Endereço: Avenida das Nações </p>
        <p>Telefone: 96578541 </p>
        <br/>
      </div>
    </div>
  );
}
```

```
export function Home() {
  return (
    <div>
      <h2>Sistema de Contatos</h2>
    </div>
  )
}
```

O que precisamos fazer agora é colocar em Home um link para cada uma das páginas Cadastro de Consulta. Mas para navegar entre as páginas de uma aplicação React precisaremos criar rotas, onde cada rota vai representar uma tela, para isso temos que criar uma rota, usando o react-router-dom. O React em si não possui um sistema de roteamento integrado, mas é comumente usado em conjunto com bibliotecas de roteamento populares, como React Router, para adicionar funcionalidades de roteamento à aplicação.

Nós já importamos o React Router usando:

```
npm install react-router-dom
```

Agora no arquivo Home.jsx crie dois links:

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

export function Home() {
  return (
    <div>
      <h2>Sistema de Contatos</h2>
      <p><Link to="cadastro">Cadastro</Link></p>
      <p><Link to="consulta">Consulta</Link></p>
    </div>
  )
}
```

O Link é usado para criar links entre diferentes rotas em uma aplicação React. `to="consulta"` ou `to="cadastro"` são atributos do componente Link que especifica o destino do link, ou seja, para qual página deve redirecionar quando clicado. A definição de qual componente/página está associado a cada um desses nome precisa ser definido no Route previamente.

Vamos fazer essa definição no Arquivo App.jsx. Ele completo ficará da seguinte forma:

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

import { Home } from '../components/Home';
import { Cadastro } from '../components/Cadastro';
import { Consulta } from '../components/Consulta';

export function App() {
  return (
    <div>

      <Routes>
        <Route path="/" element={ <Home/> } />
        <Route path="cadastro" element={ <Cadastro/> } />
        <Route path="consulta" element={ <Consulta/> } />
      </Routes>
    </div>
  )
}
```

Vamos analisar por partes:

```
<Routes>
  <Route path="/" element={ <Home/> } />
  <Route path="cadastro" element={ <Cadastro/> } />
  <Route path="consulta" element={ <Consulta/> } />
```

</Routes>

- **<Routes>:** Este é um componente fornecido pelo pacote react-router-dom que **envolve todas as rotas da sua aplicação**. Ele é usado para definir o contexto de roteamento da aplicação.
- **<Route>:** Este é um componente que **define uma rota específica** na sua aplicação. Ele mapeia um determinado caminho (path) da URL para um componente que deve ser renderizado quando esse caminho é acessado.
- **path=:** Este é o atributo path do componente <Route>, que **especifica o caminho da URL** que esta rota deve corresponder. O "/" diz respeito a url home, e path="cadastro" e path="consulta":
- **element={ <Home/> }, element={ <Cadastro/> } e element={ <Consulta/> }:** Estes são os atributos element que especificam os componentes a serem renderizados quando os caminhos correspondentes forem acessados.

Para terminar de configurar o roteamento acrescento no Main.jsx o import para o BrowserRouter, que é usado para envolver a aplicação React e habilitar o roteamento no navegador.

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import {App} from './App.jsx'
import { BrowserRouter } from "react-router-dom";

ReactDOM.createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById("root")
);
```

Perceba que não vemos os três contatos quando selecionamos a consulta isso acontece porque não incluímos o componente contatos no Consulta.jsx.

```
import {Contato} from './Contato';

export function Consulta() {
  return (
    <div>
      <h2>Lista de Contatos</h2>
      <Contato />
    </div>
  );
}
```

3.1.2 Componentes de Classe (Class Components)

Diferentemente dos componentes de funções, os componentes de classe gerenciam o próprio estado e manipulam eventos. Quando criamos um componente usando uma classe do JavaScript, essa herda os métodos do ciclo de vida do React. As aplicações modernas em React evitam usar class components. A preferência é usar function components com hooks.

```
class App extends React.Component {
  render() {
    return (
      <div>
        <p>Cabeçalho</p>
        <p>Conteúdo</p>
        <p>Rodapé</p>
      </div>
    );
  }
}
```

Não vamos usar por enquanto class components no nosso projeto.

ESTUDO DE CASO:

Vamos melhorar um pouco mais nossa aplicação colocando um cabeçalho e estilizando as páginas. Primeiramente vamos criar um estilo global para todas as páginas. Crie um arquivo global.css, na mesma pasta do App.jsx:

```
:root {
  --white: #fff;
  --gray-100: #e1e1e6;
  --gray-300: #c4c4cc;
  --gray-400: #8d8d99;
  --gray-600: #323238;
  --gray-700: #29292e;
  --gray-800: #202024;
  --gray-900: #121214;

  --green-300: #00B37E;
  --green-500: #00875f;

  --red-500: #F75A68;
}

:focus {
  outline: transparent;
  box-shadow: 0 0 0 2px var(--green-500);
}
```



```

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  background: var(--gray-900);
  color: var(--gray-300);
  -webkit-font-smoothing: antialiased;
}

body, input, textarea, button {
  font-family: "Roboto", sans-serif;
  font-weight: 400;
  font-size: 1rem;
}

```

Importe no App.jsx:

```

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

import {Home} from '../components/Home';
import {Cadastro} from '../components/Cadastro';
import {Consulta} from '../components/Consulta';

import './global.css';

export function App() {
  return (
    <div>
      <Routes>
        <Route path="/" element={ <Home/> } />
        <Route path="cadastro" element={ <Cadastro/> } />
        <Route path="consulta" element={ <Consulta/> } />
      </Routes>
    </div>
  )
}

```

Vamos agora criar um componente para representar o cabeçalho das páginas: Header.jsx. Crie esse arquivo na pasta componentes.

```

import logo from '../assets/logo.svg';

export function Header() {
  return (
    <header>
      <img src={logo} alt="Logotipo do Sistema de contatos" />
    </header>
  )
}

```

```
);
}
```

Coloque o arquivo `logo.svg` na pasta `assets`.

No arquivo `App.js` importar o cabeçalho:

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

import {Home} from './components/Home';
import {Cadastro} from './components/Cadastro';
import {Consulta} from './components/Consulta';
import {Header} from './components/Header';

import './global.css';

export function App() {
  return (
    <div>
      <Header />
      <div>
        <Routes>
          <Route path="/" element={ <Home/> } />
          <Route path="cadastro" element={ <Cadastro/> } />
          <Route path="consulta" element={ <Consulta/> } />
        </Routes>
      </div>
    </div>
  )
}
```

Verifique que como o cabeçalho foi colocado no `App.jsx` ele aparece em todas as páginas, pois ele representa a estrutura básica de todas as páginas.

Vamos agora estilizar o cabeçalho.

Quando você escreve estilos em um arquivo CSS convencional, esses estilos podem afetar todos os elementos HTML na página. Isso significa que, se você tiver dois componentes diferentes que usam classes CSS com os mesmos nomes, pode haver conflitos e os estilos podem se sobrepor, causando problemas de renderização.

Para evitar conflitos de estilos entre diferentes vamos usar os estilos “escopados” do `em CSS Modules`.

Com os estilos “escopados” do `CSS Modules`, cada componente possui seu próprio escopo de estilo.

Isso é alcançado automaticamente pelo compilador de CSS Modules, que renomeia automaticamente as classes CSS de cada componente para garantir que elas sejam únicas dentro do escopo desse componente.

Para que isso seja possível não criaremos os arquivos de estilo apenas com o nome seguido de .css, mas o nome seguido de .module.css.

Para carregar os estilos apenas do componente Header.jsx, criar um arquivo Header.module.css, coloque na mesma pasta do Header.jsx:

```
.header {
  background: var(--gray-800);
  display: flex;
  justify-content: center;
  padding: 1.25rem 0;
}

.header img {
  height: 2rem;
}
```

No Header.jsx importar o CSS import styles from './Header.module.css', observe que é preciso dar um nome ao estilo que nesse exemplo foi chamado de styles.

Agora é só aplicar o estilo desejado ao elemento do componente. Vamos estilizar a própria tag <header> usando a classe .header que foi definida no arquivo .css:

```
<header className={styles.header}>
```

O código completo do componente Header.jsx:

```
import styles from './Header.module.css';
import logo from '../assets/logo.svg';

export function Header() {
  return (
    <header className={styles.header}>
      <img src={logo} alt="Logotipo do Sistema de contatos" />
    </header>
  );
}
```

Agora vamos também criar um Module CSS apenas para o componente App.jsx, chame ele de

App.module.css, coloque na mesma pasta do App.jsx:

```
.wrapper {
  max-width: 70rem;
  margin: 2rem auto;
  padding: 0 1rem;

  display: grid;
  grid-template-columns: 256px 1fr;
  gap: 2rem;
  align-items: flex-start;
}

@media (max-width: 768px) {
  html {
    font-size: 87.5%;
  }

  .wrapper {
    grid-template-columns: 1fr;
  }
}
```

No arquivo App.js, import o CSS e estilize a tag `div className={styles.wrapper}>` O código agora fica da seguinte forma:

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

import { Home } from '../components/Home';
import { Cadastro } from '../components/Cadastro';
import { Consulta } from '../components/Consulta';
import { Header } from '../components/Header';

import styles from './App.module.css';
import './global.css';

export function App() {
  return (
    <div>
      <Header />
      <div className={styles.wrapper}>
        <Routes>
          <Route path="/" element={ <Home/> } />
          <Route path="cadastro" element={ <Cadastro/> } />
          <Route path="consulta" element={ <Consulta/> } />
        </Routes>
      </div>
    </div>
  )
}
```

Vamos criar a sidebar que mostrará o perfil do usuário. Crie o componente Sidebar.jsx. Coloque o arquivo developer.png na pasta assets.

```
import styles from './Sidebar.module.css';
import dev from '../assets/developer.png';

export function Sidebar() {
  return (
    <aside className={styles.sidebar}>
      <img className={styles.cover} src={dev}/>
      <div className={styles.profile}>
        <strong>Mônica Andrade</strong>
        <span>Front-End</span>
      </div>
    </aside>
  );
}
```

Para este componente o CSS Module Sidebar.module.css:

```
.sidebar {
  background: var(--gray-800);
  border-radius: 8px;
  overflow: hidden;
}

.cover {
  width: 100%;
  height: 72px;
  object-fit: cover;
}

.profile {
  display: flex;
  flex-direction: column;
  align-items: center;

  margin-top: calc(0.5rem );
  margin-bottom: calc( 0.5rem );
}

.profile strong {
  margin-top: 1rem;
  color: var(--gray-100);
  line-height: 1.6;
}

.profile span {
  font-size: 0.875rem;
  color: var(--gray-400);
  line-height: 1.6;
}
```

}

Acrescente a sidebar no arquivo App.jsx:

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

import {Home} from './components/Home';
import {Cadastro} from './components/Cadastro';
import {Consulta} from './components/Consulta';
import {Header} from './components/Header';
import { Sidebar } from './components/Sidebar';

import styles from './App.module.css';
import './global.css';

export function App() {
  return (
    <div>
      <Header />
      <div className={styles.wrapper}>
        <Sidebar />
        <main>
          <Routes>
            <Route path="/" element={ <Home/> } />
            <Route path="cadastro" element={ <Cadastro/> } />
            <Route path="consulta" element={ <Consulta/> } />
          </Routes>
        </main>
      </div>
    </div>
  )
}
```

Vamos configurar os botões de link da Home.jsx. Criar um arquivo CSS Module na mesma pasta de nome Home.module.css:

```
.context a {
  width: 100%;
  background: transparent;
  color: var(--green-500);
  border: 1px solid var(--green-500);
  border-radius: 8px;
  width: 250px;
  height: 50px;
  padding: 0 1.5rem;
  font-weight: bold;
  display: block;
  text-decoration: none;

  display: flex;
  align-items: center;
```

```

justify-content: center;

gap: 0.5rem;

margin-top: calc(0.5rem );
margin-bottom: calc( 0.5rem );

transition: color 0.1s, background-color 0.1s;
}

.context a:hover {
  background: var(--green-500);
  color: var(--white);
}

```

Alterar o arquivo Home.jsx para:

```

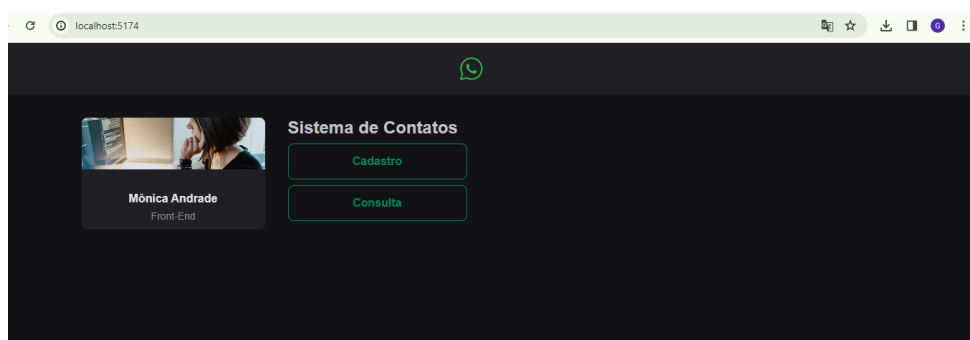
import { Link } from "react-router-dom";
import styles from './Home.module.css';

export function Home() {
  return (
    <div>
      <h2>Sistema de Contatos</h2>
      <div
        to="cadastro">Cadastro</Link></div>
      <div
        to="consulta">Consulta</Link></div>
    </div>
  )
}

```

className={styles.context}><Link
className={styles.context}><Link

A página inicial deve estar com essa aparência agora:



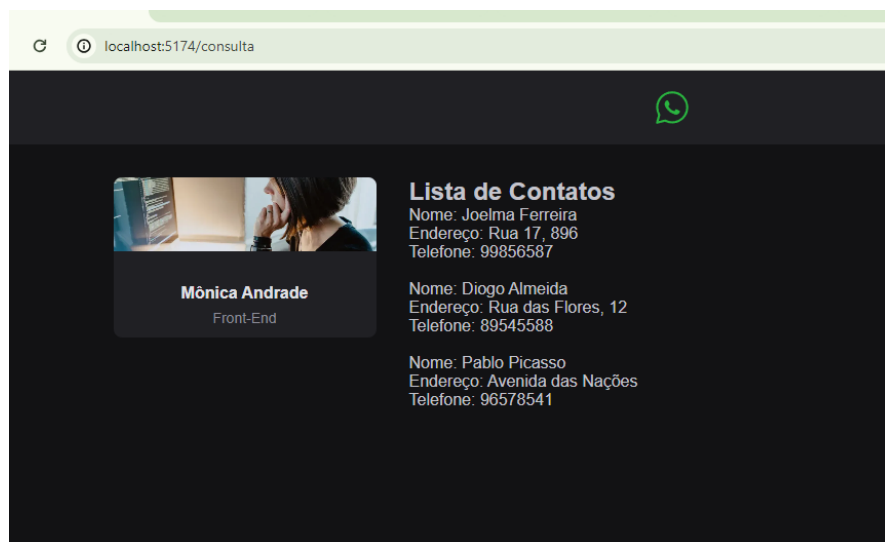
UNIDADE 4

GERENCIAMENTO DE ESTADO E PROPRIEDADES DO REACT

Ao final dos estudos, você deverá ser capaz de: compreender e aplicar os conceitos de props e state. Esses dois elementos são essenciais para o funcionamento das aplicações React, permitindo a passagem de dados entre componentes e a gestão dinâmica do estado da aplicação.

4.1 Utilização de Propriedades (Props) No React

Na hora que selecionamos consulta aparecem 3 contatos fixos na página:



Estarem fixo nesse momento não tem problema porque estamos fazendo o Fron-end, sem acesso a base de dados nenhuma, mas olhando o código de Contato.jsx vemos que existe uma repetição de partes que poderiam ser um componente Contato único, com parâmetros.

```
export function Contato() {
  return (
    <div>
      <div>
        <p>Nome: Joelma Ferreira </p>
        <p>Endereço: Rua 17, 896 </p>
        <p>Telefone: 99856587 </p>
        <br/>
        <p>Nome: Diogo Almeida </p>
```



```

        <p>Endereço: Rua das Flores, 12 </p>
        <p>Telefone: 89545588 </p>
        <br/>
        <p>Nome: Pablo Picasso </p>
        <p>Endereço:Avenida das Nações </p>
        <p>Telefone: 96578541 </p>
        <br/>
    </div>
</div>
);

```

Já Consulta.jsx chama só um componente:

```

import {Contato} from '../Contato';

export function Consulta() {
    return (
        <div>
            <h2>Lista de Contatos</h2>
            <Contato />
        </div>
    );
}

```

Vamos mudar essa estrutura, iremos criar um com componente Contatos e chamá-lo três vezes em consulta passando os dados que precisam ser renderizados como propriedade. O arquivo Contato.jsx ficaria assim:

```

export function Contato(props) {
    return (
        <div>
            <p>Nome: {props.nome} </p>
            <p>Endereço: {props.endereco} </p>
            <p>Telefone: {props.telefone} </p>
            <br/>
        </div>
    );
}

```

O arquivo Consulta.jsx alterado para criar três contatos:

```

import {Contato} from '../Contato';

export function Consulta() {
    return (
        <div>
            <h2>Lista de Contatos</h2>
            <Contato nome="Joelma Ferreira" endereco="Rua 17,
896" telefone="99856587" />

```

```

        <Contato nome="Diogo Almeida" endereco="Rua das Flores,
12" telefone="89545588" />
        <Contato nome="Pablo Picasso" endereco="Avenida das
Nações" telefone="96578541" />
    </div>
    );
}

```

Agora aplicaremos um estilo. Crie um arquivo Contato.module.css:

```

.content {
  background: var(--gray-700);
  border-radius: 8px;
  padding: 1rem;
  margin-bottom: 1rem;
  margin-top: 2rem;
}

.field {
  font-size: 0.875rem;
}

.field strong {
  line-height: 1.6;
  color: var(--green-300);
}

```

Aplique no Contato.jsx:

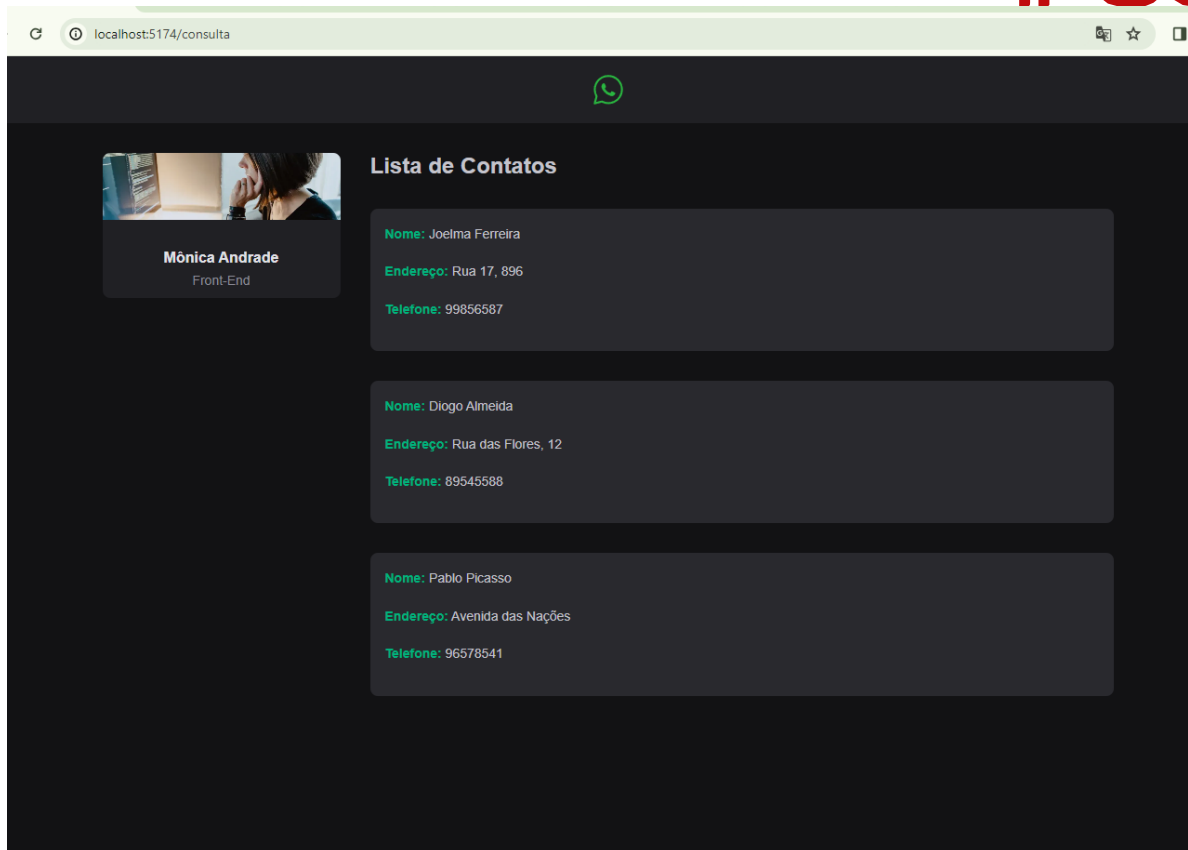
```

import styles from './Contato.module.css';

export function Contato(props) {
  return (
    <div className={styles.content}>
      <div className={styles.field}><p><strong>Nome:</strong>
{props.nome} </p></div><br/>
      <div className={styles.field}><strong>Endereço:</strong>
{props.endereco} </div><br/>
      <div className={styles.field}><strong>Telefone:</strong>
{props.telefone} </div>
      <br/>
    </div>
  );
}

```

Agora a página de consulta estará assim:



Vamos agora apenas organizar o arquivo Consulta.jsx para ler os dados de um vetor, varrer essa lista e montar os contatos dinamicamente:

```
import {Contato} from '../Contato';

const contatos = [
  {
    nome: 'Joelma Ferreira',
    endereco: '"Rua 17, 896"',
    telefone: "99856587"
  },
  {
    nome: 'Diego Fernandes',
    endereco: 'Rua das Flores, 12',
    telefone: "89545588"
  },
  {
    nome: 'Pablo Picasso',
    endereco: 'Avenida das Nações',
    telefone: "96578541"
  },
];

export function Consulta() {
  return (
    <div>
      <h2>Lista de Contatos</h2>
      {contatos.map(contato => {
        return (
          <Contato
```

```

        nome={contato.nome}
        endereco={contato.endereco}
        telefone={contato.telefone}
      />
    )
  )))
</div>
);
}

```

4.2 Utilização de Estados (States) no React

Vamos agora fazer a nossa tela de cadastro. Primeiro vamos criar o Cadastro.module.css:

```

.commentForm {
  width: 100%;
  margin-top: 1.5rem;
  padding-top: 1.5rem;
  border-top: 1px solid var(--gray-600);
}

.commentForm input {
  width: 100%;
  background: var(--gray-800);
  border: 0;
  resize: none;
  height: 3rem;
  padding: 1rem;
  border-radius: 0.25rem;
  color: var(--gray-100);
  line-height: 1.4;
  margin-top: 1rem;
}

.commentForm button[type="submit"] {
  padding: 1rem 1.5rem;
  margin-top: 1rem;
  border-radius: 8px;
  border: 0;
  background: var(--green-500);
  color: var(--white);
  font-weight: bold;
  cursor: pointer;

  transition: background-color 0.1s;
}

.commentForm button[type="submit"]:not(:disabled):hover {
  background: var(--green-300);
}

```

```
.commentForm button:disabled {
  opacity: 0.7;
  cursor: not-allowed;
}
```

Modificar o arquivo Cadastro.jsx para salvar os dados em state quando o usuário clicar no botão de cadastro:

```
import styles from './Cadastro.module.css';
import {useState} from 'react';

export function Cadastro() {
  const [formData, setFormData] = useState({nome: "",endereco:
  "",telefone: ""});

  const handleChange = (event) => {
    const { name, value } = event.target;
    setFormData((prevFormData) => ({ ...prevFormData, [name]: value
  }));
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Nome: ${formData.nome}, Endereco: ${formData.endereco},
    Telefone: ${formData.telefone}`
  );
  };

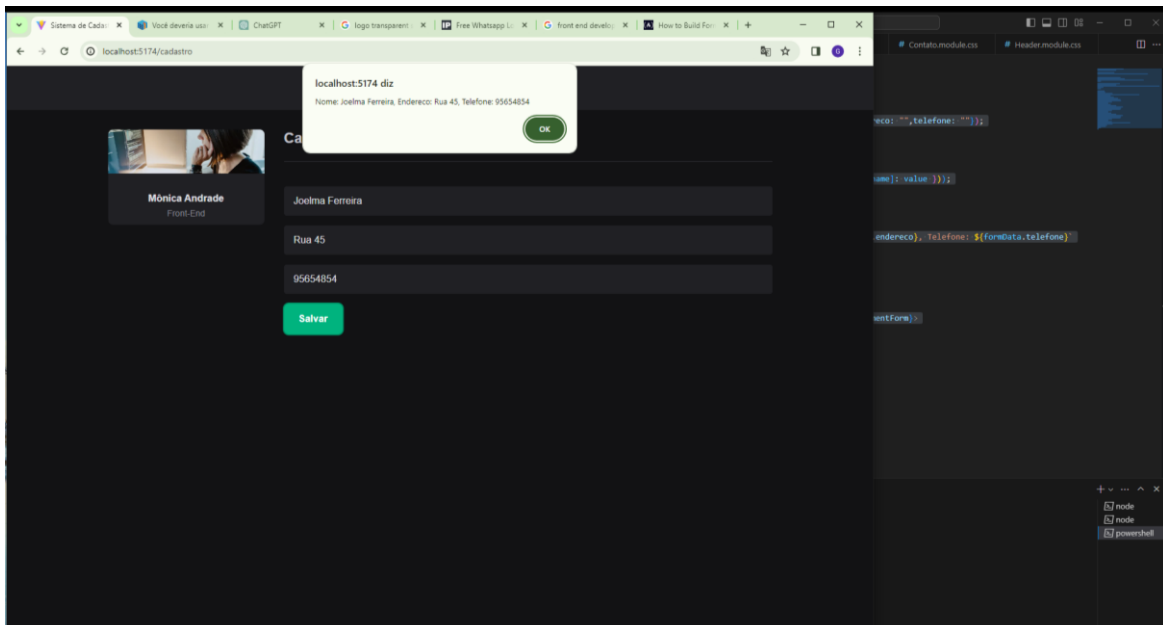
  return (
    <div>
      <h2>Cadastro</h2>
      <form onSubmit={handleSubmit} className={styles.commentForm}>
        <input
          name="nome"
          placeholder="Nome"
          value={formData.nome}
          onChange={handleChange}
          required
        />
        <input
          name="endereco"
          value={formData.endereco}
          onChange={handleChange}
          placeholder="Endereço"
          required
        />
        <input
          name="telefone"
          value={formData.telefone}
          onChange={handleChange}
          placeholder="Telefone"
          required
        />
      </form>
    </div>
  );
}
```

```

        />
        <footer>
        <button type="submit" >Salvar</button>
        </footer>
    </form>
</div>
);
}

```

A tela ficará assim:



UNIDADE 5

CICLO DE VIDA DOS COMPONENTES

Ao final dos estudos, você deverá ser capaz de: explorar conceitos mais avançados do React. Tratar dos Hook, Styled-components e reforçar conceitos de Rota.

5.1 Implementação de Rotas no React

Ao desenvolver aplicações web com React, muitas vezes é necessário gerenciar a navegação entre diferentes páginas ou componentes da aplicação. Para isso, é comum utilizar uma biblioteca de roteamento, como o React Router.

O React Router permite que você defina rotas para diferentes URLs e mapeie cada rota para um componente específico.

Mais informação no: <https://reactrouter.com/en/main>

5.2 Utilização de Hooks

Os Hooks são uma característica introduzida no React a partir da versão 16.8. Eles permitem que você utilize estado e outras características do React sem a necessidade de escrever classes. Os Hooks mais comuns incluem `useState`, `useEffect`, `useContext`, entre outros.

Os Hooks proporcionam uma maneira mais simples e concisa de escrever componentes funcionais, tornando o código mais legível e fácil de dar manutenção. Eles também facilitam a reutilização de lógica entre diferentes componentes, já que a lógica encapsulada em Hooks pode ser facilmente compartilhada.

```
import React, { useState } from 'react';

function Example() {
  // Declare uma nova variável de state, a qual chamaremos de
  "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
    </div>
  );
}
```

Esse é um conceito avançado que precisa ser entendido devagar. Mais informação no: <https://pt-br.legacy.reactjs.org/docs/hooks-intro.html>

5.3 Estilização de Componentes com Styled-Components

Styled Components é uma biblioteca para estilizar componentes em aplicações React utilizando CSS-in-JS. Em vez de criar arquivos separados para estilos CSS e importá-los em seus componentes, você pode definir estilos diretamente no arquivo do componente usando a sintaxe de template literals do JavaScript.

Isso torna a estilização mais encapsulada e fácil de manter, pois os estilos estão diretamente associados aos componentes que eles estilizam. Além disso, Styled Components oferece recursos poderosos, como a capacidade de definir estilos com base em propriedades dinâmicas e a capacidade de estilizar componentes com base no estado deles.

O código abaixo por exemplo poderia ir diretamente dá página. Jsx

```
const Button = styled.button`
  background: transparent;
  border-radius: 3px;
  border: 2px solid #BF4F74;
  color: #BF4F74;
  margin: 0 1em;
  padding: 0.25em 1em;
```

Mais informação no: <https://styled-components.com/>

PARA FINALIZAR

Chegamos ao fim desta jornada de exploração do desenvolvimento front-end. Durante esta disciplina, discutimos fundamentos e práticas essenciais para dominar esta área da tecnologia da informação.

Exploramos as nuances entre Front-End, Back-End e Full-Stack, compreendendo o papel essencial de cada um no ecossistema de desenvolvimento de software. Analisamos as diferenças entre Frameworks e Bibliotecas, reconhecendo seu impacto no processo de desenvolvimento. Dedicamos tempo para entender a biblioteca React e suas aplicações no desenvolvimento de interfaces de usuário dinâmicas e interativas.

Fizemos uma análise introdutória sobre a linguagem JavaScript, explorando desde conceitos básicos, como a declaração de variáveis, até tópicos mais avançados, como Arrow Functions e Template strings, fundamentais para o desenvolvimento front-end.

Na segunda parte da disciplina, focamos na introdução ao React, abordando desde a configuração do ambiente de desenvolvimento até os pilares essenciais dessa biblioteca, incluindo o Document Object Model (DOM), que desempenha um papel fundamental na manipulação da estrutura de uma página web.

Utilizando uma abordagem prática utilizamos Componentes no React para criar interfaces reutilizáveis e modularizadas de um problema proposto, explorando estratégias de Gerenciamento de Estado e Propriedades no React.

Convido você a continuar explorando e aprofundando seus conhecimentos neste campo do desenvolvimento front-end. Mantenha-se atualizado com as últimas tendências e tecnologias, e nunca deixe de praticar e aprimorar suas habilidades. Lembre-se de que o desenvolvimento front-end é um campo em constante evolução, e seu domínio pode abrir portas para oportunidades em sua carreira profissional.

Profa. Dra. Joelma de Moura Ferreira



Dra. **Joelma de Moura Ferreira.**

Sobre o autora

Joelma de Moura Ferreira é doutora em Ciência da Computação pela Universidade Federal de Goiás, com mestrado em Ciência da Computação pela Universidade Federal de Goiás, MBA em Gerenciamento de Projetos pela Fundação Getúlio Vargas, especialização em Redes de Computadores pela Universidade Salgado de Oliveira, MBA em Tecnologia para Negócios: AI, Data Science e Big Data pela Pontifícia Universidade Católica do Rio Grande do Sul e graduação em Ciência da Computação pela Universidade Católica de Goiás. Tendo atuado por mais de 20 anos como docente de graduação e pós-graduação em diversas instituições de ensino superior, incluindo Faculdade Sul-Americana, Universidade Paulista, Faculdade Estácio de Sá de Goiás, Pontifícia Universidade Católica, Centro Universitário Alves Farias. Desempenhou a função de coordenadora do curso de graduação de Sistemas de Informação e dos cursos de pós-graduação em Gestão de Projetos, Gestão de Tecnologia da Informação e Arquitetura e Engenharia de Software no Centro Universitário Alves Faria, onde também exerceu a atividade de pesquisadora no Mestrado em Desenvolvimento R Fora do domínio acadêmico, exerce a função de Cientista de Dados no Tribunal de Justiça do Distrito Federal e Territórios.

Referências Bibliográficas

CHAK, A. **Como Criar Sites Persuasivos**. São Paulo: Pearson Prentice Hall, 2004.

DEGEN, R. **Aprenda Programação Orientada a Objetos em 21 dias**. São Paulo: Pearson Prentice Hall, 2002.

FLANAGAN, D. **JavaScript: o guia definitivo**. 6ª ed. Porto Alegre: Bookman, 2014.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação: a construção de algoritmos e estruturas de dados**. 3ª ed. São Paulo: Pearson Prentice Hall, 2005.

MARCOLINO, A. S. **Frameworks Front End**. 1ª ed. São Paulo: Saraiva, 2021.

MILETTO, E. M. **Desenvolvimento de software II: introdução ao desenvolvimento web com html, css, javascript e php**. Porto Alegre: Bookman, 2014.

SEGURADO, V. S. **Projeto de interface com o usuário**. 1ª ed. São Paulo: Pearson, 2016.

SIMAS, V. L. et. al. **Desenvolvimento para dispositivos móveis - Volume 2**. Porto Alegre: Sagah, 2019.

