



***Front-end: React***

**React - Iniciando...**

Prof. MSc. Kelson Almeida

# Agenda

- Importante saber...
- Node.js
- NPM
- Estrutura de um projeto ReactJS
- Componentes
- JSX

# Importante saber...

- React é uma **biblioteca JS**
  - Apesar de alguns autores afirmarem que o ReactJS é um framework, ele é oficialmente e mais amplamente difundido como uma **biblioteca**.
- **Single Page Application (SPA)**
  - SPA é uma abordagem de dev de aplicações em que todo o conteúdo é carregado em uma única página da web, em vez de carregar várias páginas diferentes.
  - No React, um SPA é criado utilizando componentes React que são utilizados para construir a interface do usuário.
- Mantida pelo **Facebook**
  - O que é ótimo, pois é mantido por uma grande empresa e há sempre possibilidade de novas atualizações para a biblioteca.



# O que é o NodeJS?

- **Runtime** de JavaScript.
- Node.js é uma plataforma de desenvolvimento de aplicativos JavaScript que permite executar código JavaScript no lado do servidor.



# O que é o NodeJS?

- Embora o React possa ser executado em um navegador da web, ele também pode ser executado no lado servidor usando o Node.js.
- Isso permite que o React seja utilizado em aplicativos SPA ou em aplicações no lado do servidor.
- Além disso, o node.js fornece várias ferramentas úteis para o desenvolvimento de aplicações React.
  - Executar o servidor de desenvolvimento do React
  - Permite que visualize as alterações em tempo real enquanto trabalha na aplicação
  - Executar tarefas de compilação
  - Gerenciamento de pacotes
  - Entre outros.



# O que é o NPM?

- **npm:** (Node Package Manager) é uma ferramenta que permite aos desenvolvedores gerenciar e compartilhar pacotes de software reutilizáveis em projetos.
- Desenvolvedores costumam usar o NPM para instalar e gerenciar pacotes de dependências para seus projetos ReactJS.



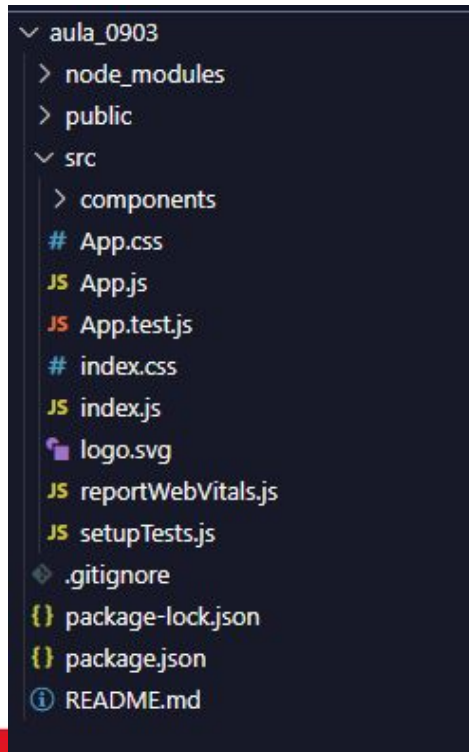
# Vamos praticar?

- Crie um projeto React chamado “aula01”.
- Entre no diretório do projeto criado e inicialize o servidor web
- Envie os prints do VS Code com o projeto criado e do navegador com a página padrão do React que é carregada ao inicializar o servidor.



# Estrutura de diretórios do projeto.

- **node\_modules:** geralmente é criado em um projeto React quando se utiliza um gerenciador de pacotes npm (node package manager) para instalar as dependências do projeto. Quando se cria o projeto React e instala dependências usando o npm, essas dependências são baixadas e armazenadas nesse diretório.

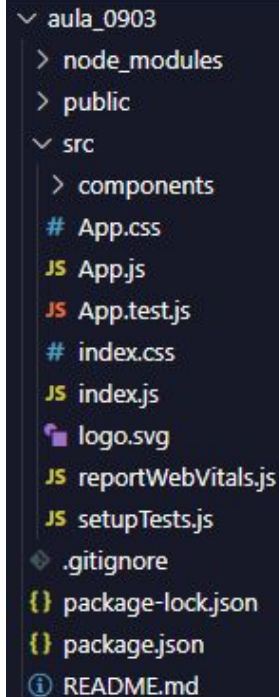




# Estrutura de diretórios do projeto.

- **public:** Esse diretório é criado para armazenar todos os recursos estáticos da aplicação React que são acessados diretamente pelo navegador.

HTML principal, imagens, arquivos de ícone, etc.



```

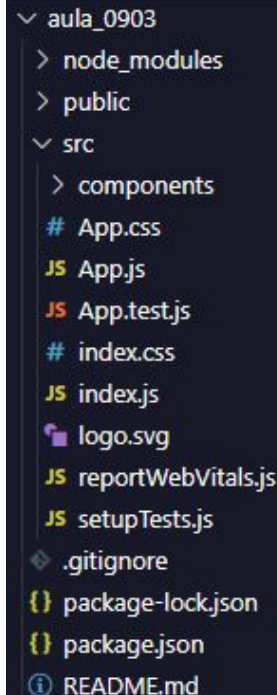
aula_0903
├── node_modules
├── public
├── src
│   ├── components
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   ├── reportWebVitals.js
│   └── setupTests.js
├── .gitignore
├── package-lock.json
├── package.json
└── README.md

```

# Estrutura de diretórios do projeto.

- **src:** (abreviação de “source”), é geralmente onde armazenamos o código-fonte da aplicação (JS, CSS e outros arquivos necessários para a codificação).

Pode haver a pasta uma pasta “components” que contém os componentes React da aplicação, uma pasta “pages” que contém os componentes que representam as páginas da aplicação e uma pasta “utils” que contém funções utilitárias.



```

aula_0903
├── node_modules
├── public
├── src
│   ├── components
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   ├── reportWebVitals.js
│   └── setupTests.js
├── .gitignore
├── package-lock.json
├── package.json
└── README.md

```

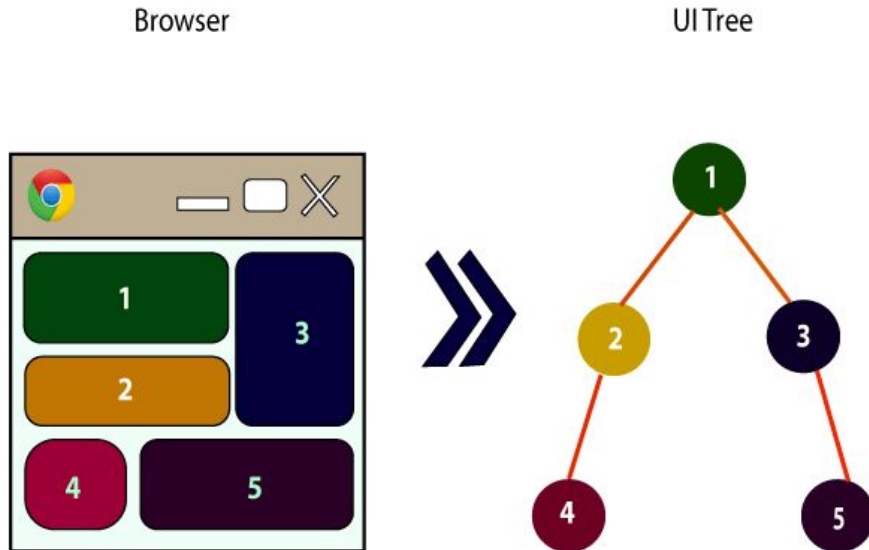
# Componentes

- Um componente é uma unidade fundamental de uma aplicação que pode ser **reutilizada** em diferentes partes da interface do usuário.
- Um componente pode ser definido **como uma função** ou **uma classe** que encapsula uma parte da interface do usuário e possui suas próprias propriedades, estado e comportamentos.



# Componentes

- Os componentes podem ser compostos uns dentro dos outros, permitindo a construção de interfaces de usuário complexas a partir de componentes mais simples e reutilizáveis.
- Os componentes podem ser divididos em dois tipos principais:
  - Componentes Funcionais
  - Componentes de Classe



# Componentes

- Componentes Funcionais: São definidos como **funções** que recebem propriedades como entrada e retornam elementos React que descrevem a interface do usuário.
- São mais simples e mais fáceis de entender e testar do que os componentes de classe, mas têm menos recursos, como estado interno.

jsx

```
import React from 'react';

function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

export default Greeting;
```

jsx

```
import React from 'react';
import Greeting from './Greeting';

function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
      <Greeting name="Charlie" />
    </div>
  );
}

export default App;
```

# Componentes

- No exemplo ao lado, Greeting é um componente funcional que recebe um objeto “props” como argumento e retorna um elemento React que exibe uma saudação personalizada.
- O elemento é renderizado usando a sintaxe de JSX, que posteriormente é compilado para código JavaScript puro.

jsx

```
import React from 'react';

function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

export default Greeting;
```

jsx

```
import React from 'react';
import Greeting from './Greeting';

function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
      <Greeting name="Charlie" />
    </div>
  );
}

export default App;
```

# Componentes

- Este componente pode ser (re)utilizado em outro componente ou aplicação, assim como qualquer outro componente React.
- No exemplo ao lado, “App” é outro componente React que renderiza três instâncias do componente “Greeting” com diferentes nomes. O resultado será uma página que exibe “Hello, Alice!”, “Hello, Bob!” e “Hello, Charlie!” na tela.

jsx

```
import React from 'react';

function Greeting(props) {
  return <h1>Hello, {props.name}</h1>;
}

export default Greeting;
```

jsx

```
import React from 'react';
import Greeting from './Greeting';

function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
      <Greeting name="Charlie" />
    </div>
  );
}

export default App;
```

# Componentes

- No exemplo ao lado, “ExemploComponente” é uma classe que estende a classe “Componente” do React. Ela tem um construtor que define o estado inicial do componente com um contador iniciado em zero.
- O método “incrementarContador” é chamado quando o botão é clicado, e atualiza o estado do componente para incrementar o contador.
- O método “render” é obrigatório em um componente de classe e retorna o JSX que será exibido na tela.
- Renderiza o título com o valor do contador e um botão que chama o método “incrementarContador” quando clicado.
- Por fim, o componente é exportado para ser utilizado em outros lugares do código.

```
javascript

import React, { Component } from 'react';

class ExemploComponente extends Component {
  constructor(props) {
    super(props);
    this.state = { contador: 0 };
  }

  incrementarContador = () => {
    this.setState({ contador: this.state.contador + 1 });
  }

  render() {
    return (
      <div>
        <h1>Contador: {this.state.contador}</h1>
        <button onClick={this.incrementarContador}>Incrementar</button>
      </div>
    );
  }
}

export default ExemploComponente;
```



# JSX

- JSX é uma extensão de sintaxe para o JavaScript que é comumente usada no desenvolvimento de aplicações React.
- O JSX permite que você escreva tags HTML e outros elementos da interface do usuário dentro do seu código JavaScript, tornando mais fácil criar componentes React.
- Em vez de criar esses elementos em uma API separada, como o DOM, o JSX permite criar esses elementos diretamente dentro do seu código JS.

jsx

```
const element = <h1>Hello, world!</h1>;
```

# JSX

- No exemplo ao lado, a sintaxe JSX é usada para criar um elemento `<h1>` que contém o texto “Hello, world!”, o elemento é armazenado em uma variável chamada “element”.
- Atenção: JSX não é JavaScript válido. Então é necessário um compilador para converter o código JSX em JavaScript. O conversor mais comum usado com o React é o Babel.

jsx

```
const element = <h1>Hello, world!</h1>;
```

# JSX

- Além disso, o JSX pode incluir expressões JavaScript dentro das tags usando chaves {}.
  - Chamamos de Templates Expressions.
- Isso permite que você crie elementos de interface do usuário dinamicamente com base em dados ou variáveis.
- No exemplo ao lado, a expressão "{name}" é avaliada como o valor da variável "name", que é "John Doe", o resultado é um elemento <h1> que contém o texto "Hello, John Doe!".

jsx

```
const name = 'John Doe';  
const element = <h1>Hello, {name}!</h1>;
```

# Vamos praticar?

- No React crie 4 componentes no seu projeto: Adicao, Subtracao, Multiplicacao e Divisao
- Esses componentes devem renderizar a seguinte frase dentro de uma tag `<h1>`, "O resultado de **num1** + **num2** é igual a **resultado**"
  - (-) para subtração
  - (\*) para multiplicação
  - (/) para divisão
- num1 , num2 são atributos do seu componente.
- Importe esses componentes criados para o App.js e passe os valores de num1 e num2 como propriedade com o objetivo de exibir a frase inteira na tela.



# Vamos praticar?

- No React crie um componente chamado “PrecisoEstudar.jsx”
- Esse componente deve renderizar a seguinte frase dentro de uma tag `<h1>`, “Preciso estudar NOME-DE-ALGUMA-TECNOLOGIA”
- O componente deve ter uma propriedade chamada “nomeDaTecnologia” que irá exibir o nome da tecnologia na frase.
- Importe esse componente criado para o App.jsx e passe o nome da tecnologia como propriedade com o objetivo de exibir a frase inteira na tela.

