## ReactJS

O React é uma biblioteca de JavaScript para a construção de interfaces de usuário, ele é mantido pelo Facebook e por uma comunidade de desenvolvedores.

Seu foco é a criação de interfaces de usuário, com componentes reutilizáveis, pode ser utilizado para criar aplicações web, mobile e desktop.

Repositório do React
Documentação React

## Por que React?

- **Reutilizável**: Componentes podem ser reutilizados em diferentes partes da aplicação.
- **Declarativo**: Criação de interfaces de usuário de forma mais simples e intuitiva.
- **Eficiente**: Utiliza o Virtual DOM para atualizar apenas as partes que foram alteradas.
- **Flexível**: Pode ser utilizado com outras bibliotecas e frameworks.
- **Comunidade**: Grande comunidade de desenvolvedores e bibliotecas disponíveis.

# Conceitos fundamentais do React

> A biblioteca React

Apesar da maioria dos códigos React serem escritos utilizando JSX, o React é uma biblioteca de JavaScript, e pode ser utilizado somente com JavaScript.

```
const element = React.createElement(
  'h1', // Tipo do elemento
  { className: 'titulo' }, // Propriedades do elemento
  'Olá, mundo!' // Conteúdo do elemento
  // <p>Elemento filho</p>
  // Aceita elementos filhos como parâmetro
)

ReactDOM.render(element, document.getElementById('root'))
```

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

O código acima cria um elemento `<h1>` com a classe `titulo` e o texto `Olá, mundo!`, e o renderiza no elemento com o id `root`.

O método `React.createElement()` cria um novo elemento React, e o método `ReactDOM.render()` renderiza o elemento no DOM.

## JSX

O JSX é uma extensão de sintaxe do JavaScript, que permite escrever códigos HTML dentro do JavaScript.

```
const element = <h1 className="titulo">Olá, mundo!</h1>

ReactDOM.render(element, document.getElementById('root'))
```

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

Com o JSX podemos escrever códigos mais legíveis e expressivos, e também podemos utilizar expressões JavaScript dentro do JSX, utilizando chaves `{}`.

```
const nome = 'Pedro'
const element = <h1 className="titulo">Olá, {nome}!</h1>
```

Isso é possível pois o JSX é transformado em chamadas de `React.createElement()` pelo Babel, que é um compilador de JavaScript.

## Virtual DOM

O Virtual DOM é uma representação em memória do DOM, ele é uma árvore de elementos React, e é utilizado para otimizar a atualização do DOM.

Toda vez que o estado de um componente é atualizado, o React renderiza o componente novamente na Virtual DOM, e compara com o DOM real, e atualiza apenas as partes que foram alteradas, esse processo é chamado de reconciliação.

## Componentes

Os componentes são a base do React, eles são funções que podem receber propriedades (props) e retornam elementos React. Por padrão começam com letra maiúscula.

```
function Componente() {
  return <h1>Renderizando o Componente!</h1>
}
const root = document.getElementById('root')
ReactDOM.render(<Componente />, root)
```

Os componentes podem ser utilizados em outros componentes, e também podem ser reutilizados em diferentes partes da aplicação.

```
function App() {
  return (
    <div>
      <Componente />
      <Componente />
    </div>
  )
}

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

## Props

As props são propriedades que são passadas para um componente, e são utilizadas para passar dados de um componente pai para um componente filho.

```
function Componente(props) {
  return <h1>Olá, {props.name}</h1>
}
```

```
function App() {
  return (
    <div>
      <Componente name="Pedro" />
      <Componente name="Fulano" />
    </div>
  )
}
```

> Hooks

Os hooks são funções que permitem adicionar funcionalidades a componentes funcionais, como por exemplo, o estado, o ciclo de vida, etc.

Alguns dos hooks mais utilizados são o `useState`, `useEffect`, `useContext`, `useReducer`, `useRef`, `useMemo`, `useCallback`.

Documentação React - Hooks
Regras dos Hooks

## Controlando estados

O hook `useState` é utilizado para adicionar estado a um componente funcional, ele retorna um array com o estado atual e uma função para atualizar o estado.

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

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

Agora o componente `Contador` possui um estado `count` que é inicializado com o valor `0`, e um botão que incrementa o valor do estado ao ser clicado.

Toda vez que o estado é atualizado, o processo de reconciliação é executado, e o componente é renderizado novamente.

## Ciclo de vida

O hook `useEffect` é utilizado para adicionar efeitos a um componente funcional, ele é executado após a renderização do componente, e pode ser utilizado para executar código assíncrono, adicionar eventos, etc.

```
function App() {
  useEffect(() => {
    console.log('Componente montado')
    return () => console.log('Componente desmontado')
  }, [])

  return <h1>Olá, mundo!</h1>
}
```

O `useEffect` recebe uma função de callback como parâmetro, que será executada após a renderização do componente, e um array de dependências, que define quando o efeito será executado.

Se o array de dependências estiver vazio, o efeito será executado apenas uma vez, após a montagem do componente. Caso contrário, o efeito será executado sempre que uma das dependências for alterada.

Também é possível retornar uma função de limpeza, que será executada quando o componente for desmontado.

## Exemplo de useEffect com dependências:

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

  useEffect(() => {
    console.log('Count foi atualizado!')
    return () => console.log('Componente desmontado')
  }, [count])

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