O React funciona com **componentes**, **estado** e um mecanismo interno chamado **Virtual DOM**, que torna a atualização da interface muito rápida e eficiente. Aqui vai uma explicação clara e didática, passo a passo — do conceito ao que realmente acontece “por baixo dos panos”.

Como o React funciona

1. Paradigma Declarativo

No JavaScript comum, você precisa dizer *como* atualizar a interface:

```
document.querySelector('#titulo').innerText = 'Olá';
```

No React, você apenas declara *o que* quer que apareça na tela:

```
<h1>Olá</h1>
```

O React decide **como** aplicar isso no navegador.

2. Componentes — O coração do React

Tudo em React é um **componente**, que pode ser:

- **Função** (mais comum)
- **Classe** (mais antigo)

Exemplo:

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

Cada componente **recebe dados (props)** e pode ter um **estado interno (state)**.

3. JSX – HTML dentro do JavaScript

O React usa **JSX**, que parece HTML mas é, na verdade, JavaScript.

```
function Mensagem() {
  return <p>Olá React!</p>;
}
```

Por trás, isso vira:

```
React.createElement("p", null, "Olá React!")
```

Essa conversão é feita pelo **Babel**.

4. Estado (State) — Quando algo muda, React atualiza a UI

Componentes podem ter estado:

```
const [contador, setContador] = useState(0);
```

Quando você chama `setContador()`, o React:

1. Atualiza o estado
2. Re-renderiza o componente
3. Compara o virtual DOM com o DOM anterior
4. Atualiza só o pedaço que mudou no navegador

Esse processo é chamado de **reconciliação**.

5. Virtual DOM — O truque que deixa tudo rápido

O React mantém uma **cópia virtual** do DOM na memória.

Quando o estado muda:

1. React cria uma nova versão do Virtual DOM
2. Compara com a versão antiga (diferença mínima, chamada *diffing*)

3. Atualiza apenas os elementos necessários no DOM real

➡️ Isso evita centenas de alterações manuais no DOM e aumenta muito a performance.

6. Unidirectional Data Flow — Fluxo de dados em uma direção

Os dados sempre fluem:

Componente pai → Componente filho

Nunca ao contrário (a não ser via callbacks).

Isso torna a aplicação **previsível** e fácil de debugar.

7. Renderização baseada em estado

A regra de ouro do React:

> "A interface é uma função do estado."

Ou seja:

```
UI = f(estado)
```

Se o estado muda → a UI muda automaticamente.

Exemplo completo

```
function Contador() {
  const [valor, setValor] = useState(0);

  return (
    <div>
      <p>Valor: {valor}</p>

      <button onClick={() => setValor(valor + 1)}>
        Somar
      </button>
```

```
    </div>
  );
}
```

Processo:

1. Inicializa estado `valor = 0`
2. Renderiza: "Valor: 0"
3. Usuário clica no botão
4. `setValor(1)` é chamado
5. React re-renderiza só esse componente
6. Virtual DOM compara e atualiza apenas o `<p>`

Resumo rápido

Conceito	O que faz
JSX	Escrever HTML dentro do JS
Babel	Converte JSX para JS
Componentes	Blocos reutilizáveis da interface
Estado	Dados dinâmicos que mudam a UI
Virtual DOM	Copia em memória que permite re-render rápido
Props	Dados enviados ao componente
Renderização declarativa	Diz o *que* renderizar, não *como*
Reatividade	Estado muda → React atualiza a UI sozinho