Aula 04 - Renderização Condicional

Prof. Me. José Olinda

O que é renderização condicional?

- Em React, renderização condicional significa mostrar ou ocultar elementos/componentes com base em condições (variáveis de estado, props, resultados de funções, etc.).
- Padroniza a criação de UIs dinâmicas: formulários distintos, mensagens de erro, loaders, permissões de usuário, etc.

Abordagens comuns

- `if` / `else` fora do JSX (mais claro para lógica complexa)
- Variáveis de elemento (element variables)
- Operador ternário inline
- Operador lógico `&&` para renderizar quando verdadeiro
- Retornar `null` para não renderizar nada

1) `if` / `else` (fora do JSX)

```jsx
function Usuario({ usuario }) {
  if (!usuario) {
    return <p>Carregando usuário...</p>;
  }

  if (usuario.erro) {
    return <p>Erro ao carregar usuário.</p>;
  }

  return (
    <div>
      <h2>{usuario.nome}</h2>
      <p>{usuario.email}</p>
    </div>
  );
```

2) Variáveis de elemento

```
function Alerta({ tipo, mensagem }) {
  let elemento;

  if (tipo === 'sucesso') {
    elemento = <div className="alert success">{mensagem}</div>;
  } else if (tipo === 'erro') {
    elemento = <div className="alert error">{mensagem}</div>;
  } else {
    elemento = null;
  }

  return <>{elemento}</>;
}
```

3) Operador ternário (inline)

```
function Login({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? <p>Bem-vindo!</p> : <button>Entrar</button>}
    </div>
  );
}
```

Prático para condições simples dentro do JSX.

4) Operador `&&` (inline if)

```jsx
function Lista({ itens }) {
  return (
    <div>
      {itens.length > 0 && (
        <ul>
          {itens.map(i => <li key={i.id}>{i.nome}</li>)}
        </ul>
      )}
      {itens.length === 0 && <p>Sem itens</p>}
    </div>
  );
}
```

Lembre-se: se a expressão à esquerda for um número `0`, ele será

5) Retornar `null` para não renderizar

```jsx
function MensagemOpcional({ texto }) {
  if (!texto) return null; // nada é renderizado
  return <p>{texto}</p>;
}
```

Isso é útil para componentes que às vezes não devem aparecer na árvore DOM.

Boas práticas

- Mantenha a lógica de renderização simples no JSX — mova condições complexas para funções auxiliares ou fora do JSX.
- Prefira `if` / `return` quando existirem múltiplos caminhos.
- Use ternário e `&&` para condicionais curtas e legíveis.
- Evite duplicação de JSX: extraia componentes quando necessário.

Exemplo

```jsx
// Perfil.jsx
import { useState } from 'react';

function Perfil({ user }) {
  return (
    <div>
      <h3>{user.nome}</h3>
      <p>{user.email}</p>
    </div>
  );
}
```

```jsx
// App.jsx
import { useState } from 'react';
import Perfil from './Perfil';
function App() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(false);

  const login = async () => {
    setLoading(true);
    // simula requisição
    setTimeout(() => {
      setUser({ nome: 'Ana', email: 'ana@exemplo.com' });
      setLoading(false);
    }, 1000);
  };
```

```
  return (
    <div>
      <h1>Minha App</h1>
      {loading && <p>Carregando...</p>}
      {!loading && !user && <button onClick={login}>Entrar</button>}
      {user ? <Perfil user={user} /> : null}
    </div>
  );
}

export default App;
```

Exercícios

1. Crie um componente `PedidoStatus` que mostra:
   - "Aguardando" quando `status === 'pendente'`;
   - "Em preparo" quando `status === 'preparando'`;
   - "Pronto" quando `status === 'pronto'`;
   - nada (retornar `null`) quando `status === 'cancelado'`.
2. Modifique o `App.jsx` de exemplo para mostrar um componente `Loader` enquanto `loading` for `true`, e um botão "Sair" quando houver `user`.

Referências

- Documentação React — Renderização condicional: https://react.dev/learn/conditional-rendering