TESTES DE INTEGRAÇÃO E ESTADO

1. Testando Cliques, Mudanças em Inputs e Simulação de Eventos

Quando criamos testes para eventos como cliques e digitação, queremos simular o comportamento do usuário e garantir que os componentes respondam corretamente.

Exemplo - Testando clique e input:

```
// Componente
import { useState } from 'react';

export function Saudacao() {
  const [nome, setNome] = useState('');

  return (
    <div>
      <input
        type="text"
        placeholder="Digite seu nome"
        onChange={(e) => setNome(e.target.value)}
      />
      <button onClick={() => alert(`Olá, ${nome}!`)}>Saudar</button>
    </div>
  );
}

// Teste
import { render, screen, fireEvent } from '@testing-library/react';
import { Saudacao } from '../components/Saudacao';

test('exibe alerta com o nome digitado', () => {
  render(<Saudacao />);
  fireEvent.change(screen.getByPlaceholderText('Digite seu nome'), {
    target: { value: 'Maria' }
  });
  fireEvent.click(screen.getByText('Saudar'));
  // Não testamos alert diretamente sem mock, mas testamos a interação
});
```

2. Testando Atualizações de Estado com useState

O hook `useState` permite armazenar e atualizar valores em componentes. Podemos testar mudanças de estado por meio da interação do usuário.

Exemplo:

```
// Componente
export function Contador() {
  const [contador, setContador] = useState(0);

  return (
    <div>
      <p>Contador: {contador}</p>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}

// Teste
test('incrementa o contador ao clicar no botão', () => {
  render(<Contador />);
  fireEvent.click(screen.getByText('Incrementar'));
  expect(screen.getByText('Contador: 1')).toBeInTheDocument();
});
```

3. Testando Exibição Condicional e Estados Derivados

Componentes podem mostrar elementos de forma condicional, dependendo do estado.

Exemplo:

```
// Componente
export function MostrarMensagem() {
  const [mostrar, setMostrar] = useState(false);

  return (
    <div>
      <button onClick={() => setMostrar(!mostrar)}>Mostrar/Ocultar</button>
      {mostrar && <p>Você clicou no botão!</p>}
    </div>
  );
}

// Teste
test('mostra e esconde a mensagem ao clicar no botão', () => {
  render(<MostrarMensagem />);
  const botao = screen.getByText('Mostrar/Ocultar');
  fireEvent.click(botao);
  expect(screen.getByText('Você clicou no botão!')).toBeInTheDocument();
  fireEvent.click(botao);
  expect(screen.queryByText('Você clicou no botão!')).toBeNull();
});
```

4. Testando Listas Dinâmicas e Componentes Controlados

Componentes controlados atualizam o estado ao digitar. Podemos gerar listas dinamicamente.

Exemplo:

```
// Componente
export function ListaTarefas() {
  const [tarefas, setTarefas] = useState([]);
  const [novaTarefa, setNovaTarefa] = useState('');

  function adicionarTarefa() {
    setTarefas([...tarefas, novaTarefa]);
    setNovaTarefa('');
  }

  return (
    <div>
      <input
        value={novaTarefa}
        onChange={(e) => setNovaTarefa(e.target.value)}
        placeholder="Nova tarefa"
      />
      <button onClick={adicionarTarefa}>Adicionar</button>
      <ul>
        {tarefas.map((tarefa, i) => <li key={i}>{tarefa}</li>)}
      </ul>
    </div>
  );
}

// Teste
test('adiciona tarefa à lista', () => {
  render(<ListaTarefas />);
  fireEvent.change(screen.getByPlaceholderText('Nova tarefa'), {
    target: { value: 'Estudar React' }
  });
  fireEvent.click(screen.getByText('Adicionar'));
  expect(screen.getByText('Estudar React')).toBeInTheDocument();
});
```