**1. Тестирование встроенных хуков**

**useState**

При тестировании компонента с useState важно проверять правильное начальное состояние и то, как состояние меняется при взаимодействии

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

import { useState } from 'react';

function Counter() {

  const [count, setCount] = useState(0);

  return (

    <>

      <p>Count: {count}</p>

      <button onClick={() => setCount(count + 1)}>Increment</button>

    </>

  );

}

test('increments count on button click', () => {

  render(<Counter />);

  expect(screen.getByText(/count: 0/i)).toBeInTheDocument();

  fireEvent.click(screen.getByText(/increment/i));

  expect(screen.getByText(/count: 1/i)).toBeInTheDocument();

});

**useEffect**

Для тестирования useEffect важно проверять побочные эффекты – например, запрос данных или изменение заголовка страницы.

import { render, screen } from '@testing-library/react';

import { useEffect, useState } from 'react';

function FetchData() {

  const [data, setData] = useState(null);

  useEffect(() => {

    setTimeout(() => {

      setData('Hello, world!');

    }, 500);

  }, []);

  return <div>{data ? data : 'Loading...'}</div>;

}

test('displays fetched data after loading', async () => {

  render(<FetchData />);

  expect(screen.getByText(/loading.../i)).toBeInTheDocument();

  // Используем findBy для ожидания обновления DOM

  const dataElement = await screen.findByText('Hello, world!');

  expect(dataElement).toBeInTheDocument();

});

**2. Тестирование кастомных хуков**

Если у вас есть кастомный хук, то для его тестирования удобно использовать @testing-library/react-hooks.

Пример кастомного хука

import { useState } from 'react';

export function useCounter(initialValue = 0) {

  const [count, setCount] = useState(initialValue);

  const increment = () => setCount((prev) => prev + 1);

  const decrement = () => setCount((prev) => prev - 1);

  return { count, increment, decrement };

}

import { renderHook, act } from '@testing-library/react-hooks';

import { useCounter } from './useCounter';

test('handles increment and decrement correctly', () => {

  const { result } = renderHook(() => useCounter(5));

  // Проверяем начальное состояние

  expect(result.current.count).toBe(5);

  // Делаем increment

  act(() => result.current.increment());

  expect(result.current.count).toBe(6);

  // Делаем decrement

  act(() => result.current.decrement());

  expect(result.current.count).toBe(5);

});

**3. Тестирование хуков с моками**

Когда хук использует внешнюю зависимость, например, работу с API, нужно использовать моки (mock). Это можно сделать с помощью Jest.

Пример хука:

import { useState, useEffect } from 'react';

export function useFetch(url) {

  const [data, setData] = useState(null);

  useEffect(() => {

    fetch(url)

      .then((res) => res.json())

      .then((data) => setData(data));

  }, [url]);

  return data;

}

import { renderHook } from '@testing-library/react-hooks';

import { useFetch } from './useFetch';

global.fetch = jest.fn(() =>

  Promise.resolve({

    json: () => Promise.resolve({ message: 'Hello, world!' }),

  })

);

test('fetches and returns data', async () => {

  const { result, waitForNextUpdate } = renderHook(() => useFetch('/api/test'));

  // Ждем обновления состояния

  await waitForNextUpdate();

  expect(result.current).toEqual({ message: 'Hello, world!' });

  expect(fetch).toHaveBeenCalledWith('/api/test');

});