**Мок (mock)** - это имитация реального объекта/функции/модуля для тестирования.

Представь, что это как дублёр в кино: выглядит и ведёт себя как настоящий объект, но на самом деле это подмена.

**Зачем нужны моки?**

Изолировать тестируемый код

Ускорить тесты (не нужно ждать реальных API-запросов)

Избежать побочных эффектов

Симулировать разные сценарии (успех/ошибка)

Контролировать поведение зависимостей

**Основные способы создания моков в Jest:**

// 1. jest.fn() - создание мок-функции

const mockFunction = jest.fn();

mockFunction.mockReturnValue('результат');

// 2. jest.mock() - мокание целого модуля

jest.mock('./path/to/module');

// 3. jest.spyOn() - создание шпиона для существующего метода

jest.spyOn(object, 'method');

**Практический пример с React компонентом:**

// UserComponent.js

import axios from 'axios';

function UserComponent() {

  const [user, setUser] = useState(null);

  const fetchUser = async () => {

    const response = await axios.get('/api/user');

    setUser(response.data);

  };

  return (

    <div>

      <button onClick={fetchUser}>Загрузить пользователя</button>

      {user && <div>{user.name}</div>}

    </div>

  );

}

// UserComponent.test.js

import axios from 'axios';

jest.mock('axios');

describe('UserComponent', () => {

  test('загружает и отображает пользователя', async () => {

    // Подготавливаем мок

    const mockUser = { name: 'John Doe' };

    axios.get.mockResolvedValue({ data: mockUser });

    // Рендерим компонент

    render(<UserComponent />);

    // Кликаем по кнопке

    fireEvent.click(screen.getByText('Загрузить пользователя'));

    // Проверяем, что имя пользователя появилось

    await screen.findByText('John Doe');

    // Проверяем, что axios.get был вызван

    expect(axios.get).toHaveBeenCalledWith('/api/user');

  });

});

**Полезные методы для работы с моками:**

// Установка возвращаемого значения

mockFn.mockReturnValue(value);

mockFn.mockResolvedValue(value); // для Promise

// Проверка вызовов

expect(mockFn).toHaveBeenCalled();

expect(mockFn).toHaveBeenCalledWith(arg1, arg2);

expect(mockFn).toHaveBeenCalledTimes(2);

// Очистка моков

jest.clearAllMocks();

**Советы по использованию моков:**

Мокайте только то, что действительно нужно

Старайтесь держать моки максимально простыми

Используйте моки для внешних зависимостей (API, базы данных)

Не злоупотребляйте моками для внутренней логики компонентов

**Часто мокаемые вещи в React:**

API-запросы (axios, fetch)

Таймеры (setTimeout, setInterval)

Внешние библиотеки

React Router

Redux actions/store

Пример с таймерами:

test('компонент с таймером', () => {

  // Используем фейковые таймеры

  jest.useFakeTimers();

  render(<TimerComponent />);

    // Перематываем время вперед

  jest.advanceTimersByTime(1000);

  // Проверяем результат

  expect(screen.getByText('1 секунда')).toBeInTheDocument();

  // Возвращаем реальные таймеры

  jest.useRealTimers();

});

**Пример с React Router:**

jest.mock('react-router-dom', () => ({

  ...jest.requireActual('react-router-dom'),

  useNavigate: () => jest.fn(),

}));

Главное помнить:

Моки помогают изолировать тестируемый код

Используйте их для внешних зависимостей

Держите моки простыми и понятными

Не забывайте очищать моки после тестов

Тестируйте реальное поведение, где это возможно