**1. React: Компоненты, рендер, вызов и параметры**

**Компонент** — это часть UI, описанная как функция или класс.  
**Вызов:** <MyComponent prop1="value" />. Параметры передаются через props.  
**Рендер:** JSX превращается в виртуальный DOM, сравнивается с текущим, изменения применяются к реальному DOM.  
**Запуск:** через Create React App, Vite, Next.js.

**2. Props, вложенные компоненты, списки, условия**

**Props:** передаются от родителя к дочернему компоненту, неизменяемы.  
**Вложенные компоненты:** компоненты могут содержать другие компоненты.  
**Списки:** выводятся с помощью .map(), каждому элементу нужен key.

items.map((item, index) => <li key={index}>{item}</li>)

**Условия:** через тернарный оператор или &&.

isLoggedIn ? <p>Welcome</p> : <p>Please log in</p>;

**3. FC vs Class, жизненный цикл, хуки**

**FC (Functional Component):** проще, используют хуки.  
**Class Component:** используют методы жизненного цикла.  
**Жизненный цикл:**

* Монтирование: componentDidMount.
* Обновление: componentDidUpdate.
* Размонтирование: componentWillUnmount.

**Хуки:**

* **useState:** состояние компонента.

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

* **useEffect:** эффекты.

useEffect(() => {...}, [dependency]);

* **useRef:** доступ к DOM-узлам, сохраняет данные между рендерами.

### 4. Хуки и контекст

**useCallback:** мемоизирует функции.

const memoizedFn = useCallback(() => doSomething(a, b), [a, b]);

**useContext:** доступ к значению контекста.

const value = useContext(MyContext);

**useReducer:** управление сложным состоянием.

const [state, dispatch] = useReducer(reducer, initialState);

**Контекст:** передает данные без props.

const MyContext = React.createContext();

<MyContext.Provider value={value}><App /></MyContext.Provider>;

**5. Управляемые/неуправляемые компоненты и работа с событиями**

**Управляемые компоненты:** значения хранятся в state, изменения обрабатываются через события.

const [value, setValue] = useState('');

<input value={value} onChange={(e) => setValue(e.target.value)} />;

**Неуправляемые компоненты:** управляются напрямую через ref.

const inputRef = useRef(null);

<input ref={inputRef} />;

**События:** используются как атрибуты (onClick, onChange). Вмешательство — вызов event.preventDefault() или event.stopPropagation().

**6. Обработка разных типов контролов**

Обработка контролов через onChange или onClick:

* **Текстовые поля:** value и onChange.
* **Чекбоксы и радио:** checked и onChange.
* **Селект:** value и onChange.
* **Кнопки:** onClick.
* **Ползунки:** value и onChange.  
  Пример:

<input type="checkbox" checked={isChecked} onChange={(e) => setChecked(e.target.checked)} />;

**7. Валидация форм**

**Алгоритм работы:**

1. На вводе проверять данные (regex, длина и т.д.).
2. Отправка формы блокируется при ошибках.
3. Показывать сообщения об ошибках.

**Клиентская валидация:** выполняется в браузере (быстрее, не требует серверных запросов).  
**Серверная валидация:** проверка на сервере для надежности.  
Пример:

if (!email.includes('@')) setError('Invalid email');

**8. Библиотеки готовых компонентов и CSS-фреймворки**

**Примеры библиотек:**

* **MUI**, **Ant Design**, **React Bootstrap** — UI-компоненты.
* **react-table**, **react-datepicker** — для таблиц и дат.

**Установка и подключение:**

npm install @mui/material

import { Button } from '@mui/material';

<Button variant="contained">Click</Button>;

**Особенности:**

* **Таблицы:** react-table поддерживает сортировку и фильтрацию.
* **Даты:** react-datepicker предоставляет удобный выбор.

**CSS-фреймворки:**

* **Bootstrap** — традиционный.
* **Tailwind CSS** — утилитарный подход.

**9. SPA и работа с роутером**

**SPA:** приложение, где переход между страницами выполняется без полной перезагрузки.

**Роутеры:**

* **Типы:** BrowserRouter (URL с историей), HashRouter (URL с #).
* **react-router-dom:**

npm install react-router-dom

import { BrowserRouter, Routes, Route } from 'react-router-dom';

<BrowserRouter>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

</Routes>

</BrowserRouter>;

**Вложенные роуты и параметры:**

<Route path="/user/:id" element={<User />} />;

const { id } = useParams();

**10. Компонент Route, path, переход, хуки**

**Route:** определяет путь и компонент.

<Route path="/about" element={<About />} />;

**Программный переход:**

import { useNavigate } from 'react-router-dom';

const navigate = useNavigate();

navigate('/about');

**Лейаут:** общий шаблон для вложенных роутов.

<Route path="/" element={<Layout />}>

<Route path="about" element={<About />} />

</Route>;

**Хуки:**

* useNavigate: программные переходы.
* useParams: доступ к параметрам URL.
* useLocation: текущий маршрут.

**11. Запросы в сеть**

**GET-запрос:**

fetch('https://api.example.com/data')

.then((res) => res.json())

.then((data) => console.log(data));

**POST-запрос:**

fetch('https://api.example.com/data', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ key: 'value' }),

});

**Cookies:** отправляются автоматически, если указано credentials: 'include':

fetch(url, { credentials: 'include' });

**12. Node.js**

**Установка и запуск:**

1. Установить с [nodejs.org](https://nodejs.org).
2. Проверить: node -v, npm -v.

**Event Loop:** механизм обработки асинхронных операций. Стадии: таймеры, IO, коллбеки, проверки, закрытия.

**Встроенные модули:** fs, path, http, events.  
**Подключение модулей:**

* **CommonJS:** const fs = require('fs');
* **ES6:** import fs from 'fs';

**13. Сервер на Node.js**

**HTTP-запросы:**

* **GET:** получение данных.
* **POST:** отправка данных.
* **PUT:** обновление.
* **DELETE:** удаление.

**Пример сервера:**

const http = require('http');

const server = http.createServer((req, res) => {

if (req.method === 'GET') res.end('Hello, GET!');

if (req.method === 'POST') res.end('Hello, POST!');

});

server.listen(3000, () => console.log('Server running on port 3000'));

**Особенности:** парсинг тела запроса через body-parser или json().

**14. Express.js: установка и обработчики**

**Установка:**

npm install express

**Идеология обработчиков:** маршруты обрабатывают запросы:

const express = require('express');

const app = express();

app.get('/', (req, res) => res.send('Hello, World!'));

app.listen(3000);

**Объекты:**

* **req:** запрос (параметры, заголовки).
* **res:** ответ (методы send, json, status).

**15. POST-запросы, куки и CORS**

**Разбор тела запроса:**

app.use(express.json());

app.post('/data', (req, res) => res.send(req.body));

**Работа с куками:** через cookie-parser:

npm install cookie-parser

const cookieParser = require('cookie-parser');

app.use(cookieParser());

app.get('/', (req, res) => res.cookie('name', 'value').send('Cookie set'));

**CORS:** настройка через cors:

npm install cors

const cors = require('cors');

app.use(cors({ origin: 'https://example.com' }));

**16. Обработка ошибок и middleware**

**Обработка ошибок:**

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).send('Something broke!');

});

**Метод next():** передает управление следующему обработчику.

**Middleware:** функции, выполняющиеся между запросом и ответом. Пример:

app.use((req, res, next) => {

console.log('Request URL:', req.url);

next();

});

**17. RESTful API**

**Принципы:**

* **Stateless:** сервер не хранит состояние между запросами.
* **Коды статуса:** 200 OK, 201 Created, 400 Bad Request, 404 Not Found.

**Схема API:** определяет маршруты и действия (CRUD).

**Роуты и параметры:**

app.get('/users/:id', (req, res) => res.send(`User ID: ${req.params.id}`));

**18. Аутентификация**

**Алгоритм логина:**

1. Проверить учетные данные.
2. Сохранить сессию или JWT.

**HTTP-куки:** используются для сохранения информации сессии.

res.cookie('sessionId', 'value', { httpOnly: true });

**Проверка сессии:**

if (req.cookies.sessionId) res.send('Logged in');

else res.status(401).send('Unauthorized');

**19. SQLite**

**Установка:**

npm install sqlite3

**Подключение и работа:**

const sqlite3 = require('sqlite3').verbose();

const db = new sqlite3.Database(':memory:');

db.run('CREATE TABLE user (id INT, name TEXT)');

db.run('INSERT INTO user (id, name) VALUES (1, "John")');

db.get('SELECT \* FROM user', (err, row) => console.log(row));

**20. ORM Sequelize**

**Понятие ORM:**  
ORM (Object-Relational Mapping) связывает объекты с таблицами БД.

**Установка и подключение Sequelize:**

npm install sequelize sqlite3

const { Sequelize, DataTypes } = require('sequelize');

const sequelize = new Sequelize('sqlite::memory:');

const User = sequelize.define('User', { name: DataTypes.STRING });

sequelize.sync();

User.create({ name: 'John' });

**Связи между таблицами:**

User.hasMany(Posts);

Posts.belongsTo(User);

Задачи

1. Написать кликер с кнопками  
import React, { useState } from 'react';

function Counter() {

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

return (

<div>

<h1 style={{ color: count % 2 === 0 ? 'green' : 'red' }}>{count}</h1>

<button onClick={() => count > 0 && setCount(count - 1)}>-</button>

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

</div>

);

}

export default Counter;

2. Список чисел с фильтрацией  
import React, { useState } from 'react';

function NumberList() {

const [numbers, setNumbers] = useState([]);

const [filter, setFilter] = useState('');

const addNumber = (e) => {

e.preventDefault();

const num = parseInt(e.target.number.value);

setNumbers([...numbers, num]);

e.target.number.value = '';

};

const filteredNumbers = numbers.filter(n => n.toString().includes(filter));

return (

<div>

<form onSubmit={addNumber}>

<input name="number" />

<button type="submit">+</button>

</form>

<input

placeholder="Фильтр..."

onChange={(e) => setFilter(e.target.value)}

/>

<ul>

{filteredNumbers.map((n, i) => (

<li key={i}>{n}</li>

))}

</ul>

</div>

);

}

export default NumberList;

3. Калькулятор систем счисления.

import React, { useState } from 'react';

function Calculator() {

const [dec, setDec] = useState(0);

const [base, setBase] = useState(2);

const handleChange = (e) => {

const value = parseInt(e.target.value, base);

setDec(value);

};

return (

<div>

<input

type="number"

value={dec.toString(base)}

onChange={handleChange}

/>

<select onChange={(e) => setBase(parseInt(e.target.value))} value={base}>

<option value="2">Двоичная</option>

<option value="8">Восьмеричная</option>

<option value="10">Десятичная</option>

<option value="16">Шестнадцатеричная</option>

</select>

<h3>{dec.toString(base)}</h3>

</div>

);

}

export default Calculator;

4. Сделать калькулятор перевода температур

import React, { useState } from 'react';

function TempConverter() {

const [celsius, setCelsius] = useState(0);

const handleCelsius = (e) => setCelsius(e.target.value);

const fahrenheit = (celsius \* 9) / 5 + 32;

const handleFahrenheit = (e) => setCelsius(((e.target.value - 32) \* 5) / 9);

return (

<div>

<input type="number" value={celsius} onChange={handleCelsius} />

<h3>°C в °F: {fahrenheit.toFixed(2)}</h3>

<input type="number" value={fahrenheit.toFixed(2)} onChange={handleFahrenheit} />

<h3>°F в °C: {celsius.toFixed(2)}</h3>

</div>

);

}

export default TempConverter;

5. Обратный таймер.  
import React, { useState, useEffect } from 'react';

function Timer() {

const [time, setTime] = useState(10);

const [running, setRunning] = useState(false);

useEffect(() => {

let interval;

if (running && time > 0) {

interval = setInterval(() => setTime((prev) => prev - 1), 1000);

} else clearInterval(interval);

return () => clearInterval(interval);

}, [running, time]);

return (

<div>

<h3>{time}</h3>

<button onClick={() => setRunning(true)}>Старт</button>

<button onClick={() => setRunning(false)}>Стоп</button>

</div>

);

}

export default Timer;

6. Частотный словарь

import React, { useState } from 'react';

function WordCounter() {

const [text, setText] = useState('');

const getWordCount = (str) => {

const words = str.split(/\s+/).filter(Boolean);

const freq = {};

words.forEach((word) => (freq[word] = (freq[word] || 0) + 1));

return freq;

};

return (

<div>

<textarea

value={text}

onChange={(e) => setText(e.target.value)}

/>

<ul>

{Object.entries(getWordCount(text)).map(([word, count]) => (

<li key={word}>

{word}: {count}

</li>

))}

</ul>

</div>

);

}

export default WordCounter;

7. Сделать кнопку, которая при нажатии меняет цвет фона

import React, { useState } from 'react';

function ColorToggle() {

const [bg, setBg] = useState('red');

return (

<div style={{ background: bg, padding: '20px' }}>

<button onClick={() => setBg(bg === 'red' ? 'green' : 'red')}>Переключить</button>

</div>

);

}

export default ColorToggle;

8. Добавляем числа в список и показываем среднее арифметическое

import React, { useState } from 'react';

function AvgCalculator() {

const [numbers, setNumbers] = useState([]);

const addNumber = () => {

const num = parseInt(prompt('Введите число:'));

if (!isNaN(num)) {

const newNumbers = [...numbers, num];

setNumbers(newNumbers);

}

};

const avg = numbers.reduce((a, b) => a + b, 0) / (numbers.length || 1);

return (

<div>

<button onClick={addNumber}>Добавить число</button>

<h3>Среднее: {avg.toFixed(2)}</h3>

</div>

);

}

export default AvgCalculator;

9. Alert. Компоненту в пропсах передаётся строка для отображения и задержка в секундах.

import React, { useState, useEffect } from 'react';

function DelayedAlert({ message, delay }) {

const [visible, setVisible] = useState(false);

useEffect(() => {

setVisible(true);

const timer = setTimeout(() => setVisible(false), delay \* 1000);

return () => clearTimeout(timer);

}, [message, delay]);

return visible ? <div>{message}</div> : null;

}

export default DelayedAlert;

10. Обратная строка с кнопкой.

import React, { useState } from 'react';

function ReverseString({ text }) {

const [reversed, setReversed] = useState(text);

const reverseUnicode = (str) => Array.from(str).reverse().join('');

return (

<div>

<p>{reversed}</p>

<button onClick={() => setReversed(reverseUnicode(reversed))}>Перевернуть</button>

</div>

);

}

export default ReverseString;

11. Светофор.

import React, { useState, useEffect } from 'react';

function TrafficLight() {

const [color, setColor] = useState('🔴');

const colors = ['🔴', '🟡', '🟢'];

useEffect(() => {

const timer = setInterval(() => {

setColor((prev) => colors[(colors.indexOf(prev) + 1) % colors.length]);

}, 1000);

return () => clearInterval(timer);

}, []);

return <div style={{ fontSize: '50px' }}>{color}</div>;

}

export default TrafficLight;

12. Счётчик с переменным шагом

import React, { useState } from 'react';

function StepCounter() {

const [step, setStep] = useState(1);

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

return (

<div>

<input

type="range"

min="1"

max="10"

value={step}

onChange={(e) => setStep(+e.target.value)}

/>

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

<p>Текущее значение: {count}</p>

</div>

);

}

export default StepCounter;

13. Обращение к сети.

import React, { useState } from 'react';

function FetchTodo() {

const [id, setId] = useState('');

const [title, setTitle] = useState('');

const fetchData = async () => {

const res = await fetch(`https://jsonplaceholder.typicode.com/todos/${id}`);

const data = await res.json();

setTitle(data.title || 'Not found');

};

return (

<div>

<input value={id} onChange={(e) => setId(e.target.value)} />

<button onClick={fetchData}>Получить</button>

<p>{title}</p>

</div>

);

}

export default FetchTodo;

14. PIN-код.

import React, { useState } from 'react';

function PinInput() {

const [pin, setPin] = useState('');

const handleInput = (digit) => {

const updatedPin = (pin + digit).slice(-4);

setPin(updatedPin);

if (updatedPin === '9999') alert('Правильный PIN');

};

return (

<div>

<input value={pin} readOnly />

<div>

{[...Array(10).keys()].map((n) => (

<button key={n} onClick={() => handleInput(n)}>{n}</button>

))}

</div>

</div>

);

}

export default PinInput;

15. Калькулятор систем счисления.

import React, { useState } from 'react';

function BaseConverter() {

const [dec, setDec] = useState('');

const [base, setBase] = useState(2);

const handleChange = (e) => setDec(e.target.value);

return (

<div>

<input value={dec} onChange={handleChange} />

{[2, 8, 16].map((b) => (

<div key={b}>

<p>Основание {b}: {parseInt(dec || 0, 10).toString(b)}</p>

</div>

))}

</div>

);

}

export default BaseConverter;

16. Список чисел с фильтрацией.

import React, { useState } from 'react';

function NumberFilter() {

const [numbers, setNumbers] = useState([]);

const [filter, setFilter] = useState('all');

const [input, setInput] = useState('');

const addNumber = () => {

if (input) setNumbers([...numbers, parseInt(input)]);

setInput('');

};

const filteredNumbers = numbers.filter((n) =>

filter === 'all' ? true : filter === 'even' ? n % 2 === 0 : n % 2 !== 0

);

return (

<div>

<input value={input} onChange={(e) => setInput(e.target.value)} />

<button onClick={addNumber}>+</button>

<select onChange={(e) => setFilter(e.target.value)}>

<option value="all">Все</option>

<option value="even">Чётные</option>

<option value="odd">Нечётные</option>

</select>

<ul>

{filteredNumbers.map((n, i) => (

<li key={i}>{n}</li>

))}

</ul>

</div>

);

}

export default NumberFilter;

17. Лайк

import React, { useState } from 'react';

function LikeButton() {

const [liked, setLiked] = useState(false);

let clickTimer = null;

const handleClick = () => {

if (clickTimer) {

clearTimeout(clickTimer);

clickTimer = null;

setLiked(true);

} else {

clickTimer = setTimeout(() => {

clickTimer = null;

}, 800);

}

};

return (

<div onClick={handleClick} style={{ fontSize: '50px', color: liked ? 'red' : 'black' }}>

❤

</div>

);

}

export default LikeButton;

18. Градусник.

import React from 'react';

function Thermometer({ temp }) {

return (

<div style={{ color: temp < 0 ? 'blue' : 'red' }}>

Температура: {temp}°C

</div>

);

}

export default Thermometer;

19. Написать минимальный сервер на Express.js, отвечающий на GET-запрос

const express = require('express');

const app = express();

app.get('/', (req, res) => res.send('Hello, World!'));

app.listen(3000, () => console.log('Сервер запущен на http://localhost:3000'));

20. Написать минимальный сервер на Node.js (без использования каких-либо библиотек), отвечающий на GET-запрос

const http = require('http');

const server = http.createServer((req, res) => {

if (req.method === 'GET') {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('Hello, World!');

}

});

server.listen(3000, () => console.log('Сервер запущен на http://localhost:3000'));