**1. Управление состоянием UI (интерфейса)**

Отображение или скрытие элементов интерфейса\*\*

  Например, отображение модальных окон, выпадающих списков, подсказок и других элементов.

  Пример:

const [isModalVisible, setModalVisible] = useState(false);

return (

  <>

    <button onClick={() => setModalVisible(true)}>Открыть модальное окно</button>

    {isModalVisible && <div>Это модальное окно!</div>}

  </>

);

**2. Формы и ввод пользователей**

Отслеживание изменений в полях ввода\*\*

  useState позволяет управлять значениями полей ввода текстовых форм, чекбоксов, радио-кнопок и других элементов формы.

  Пример:

const [name, setName] = useState("");

return (

  <form>

    <label>

      Имя:

      <input

        type="text"

        value={name}

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

      />

    </label>

    <p>Вы ввели: {name}</p>

  </form>

);

**3. Интерактивные компоненты**

Счетчики, переключатели и подобные элементы\*\*

  Это одна из самых популярных задач для использования useState. Например, создание счетчика будет выглядеть так:

  Пример:

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

return (

  <div>

    <p>Текущий счет: {count}</p>

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

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

  </div>

);

**4. Управление списками данных**

Добавление, удаление и обновление элементов списка\*\*

  Часто используется для управления локальными данными, например, при создании списка задач.

  Пример:

const [tasks, setTasks] = useState([]);

const addTask = () => {

  setTasks([...tasks, `Задача ${tasks.length + 1}`]);

};

return (

  <>

    <ul>

      {tasks.map((task, index) => (

        <li key={index}>{task}</li>

      ))}

    </ul>

    <button onClick={addTask}>Добавить задачу</button>

  </>

);

**5. Переключатели (Toggle)**

Состояние включено/выключено\*\*

  Удобно для переключателей, кнопок, темы (светлая/темная), и других подобных случаев.

  Пример:

const [isDarkMode, setDarkMode] = useState(false);

return (

  <div style={{ background: isDarkMode ? "black" : "white", color: isDarkMode ? "white" : "black" }}>

    <p>Темный режим {isDarkMode ? "включен" : "выключен"}</p>

    <button onClick={() => setDarkMode(!isDarkMode)}>Переключить режим</button>

  </div>

);

**6. Состояние загрузки**

Индикация загрузки\*\*

  Используется для отображения состояний "загрузка", "успех", "ошибка" при работе с асинхронным кодом.

  Пример:

   const [isLoading, setLoading] = useState(false);

const fetchData = async () => {

  setLoading(true);

  await new Promise(resolve => setTimeout(resolve, 2000)); // Имитируем загрузку

  setLoading(false);

};

return (

  <div>

    <button onClick={fetchData}>Загрузить данные</button>

    {isLoading ? <p>Загрузка...</p> : <p>Данные загружены!</p>}

  </div>

);

**7. Состояние ошибок**

Обработка ошибок\*\*

  С помощью useState можно сохранить текст ошибки или статус для отображения пользователю.

  Пример:

const [error, setError] = useState("");

const handleSubmit = () => {

  const isValid = false; // Проверка

  if (!isValid) {

    setError("Произошла ошибка!");

  }

};

return (

  <div>

    <button onClick={handleSubmit}>Отправить</button>

    {error && <p style={{ color: "red" }}>{error}</p>}

  </div>

);

**8. Переключение страниц или вкладок**

Управление активной вкладкой или страницей\*\*

  Используется в интерфейсах с несколькими вкладками или страницами (например, переключение между разными секциями).

  Пример:

const [activeTab, setActiveTab] = useState("home");

return (

  <div>

    <button onClick={() => setActiveTab("home")}>Главная</button>

    <button onClick={() => setActiveTab("profile")}>Профиль</button>

    <button onClick={() => setActiveTab("settings")}>Настройки</button>

    {activeTab === "home" && <p>Вы на главной странице</p>}

    {activeTab === "profile" && <p>Это профиль</p>}

    {activeTab === "settings" && <p>Это настройки</p>}

  </div>

);

**9. Хранение временных данных**

С помощью useState удобно хранить временные данные, которые не нужно передавать между компонентами или сохранять в глобальном состоянии. Например, данные, которые не важны после закрытия страницы.