

Цели занятия

- узнать как можно вкладывать компоненты друг в друга;
- научиться работать с коллекциями, избавляясь от дублирования кода;
- узнать для чего нужны ключи и как с ними работать;

Полезные материалы

- [Работа со списками и ключами](#)
- [Углубленное изучение JSX](#)
- [Методы массивов](#)

Специальное свойство children

Очень часто возникает потребность вкладывать компоненты друг в друга, как и простые теги в HTML. Но в React вы сами можете управлять местом, куда необходимо вставить дочерние элементы. Для таких случаев есть необязательное свойство children и давайте сразу рассмотрим его на примере:

```
function Form(props) {  
  const { title, children } = props;  
  
  return (  
    <form className='form'>  
      <div className='form__title'>{title}</div>  
  
      <div className='form__content'>  
        { /* Здесь будет вставлено все содержимое формы */ }  
        {children}  
      </div>  
    </form>  
  );  
}
```

```
    </form>
  )
}

function App() {
  return (
    <Form title='Моя форма'>
      {/* Всё что внутри компонента попадет в children */}
      <input name='phone' placeholder='+7 900 000-00-00' />
      <button type='submit'>Отправить</button>
    </Form>
  )
}
```

Именно эта возможность позволяет нам вкладывать компоненты друг в друга и получать приложения или веб-сайты любой сложности и гибкости.

Работа со списками

Практически на каждой странице сайта есть повторяющиеся элементы, которые с точки зрения разметки абсолютно одинаковы и различаются лишь своим содержимым или какими-то данными.

Например, в нашем с вами [макете](#) такие тоже есть, например, это кнопки для выбора цвета товара и список отзывов. И разумеется, что в реальном проекте список отзывов мы получаем от бэкенде в виде массива объектов, а во фронтенде у нас есть лишь один компонент.

Но прежде давайте вспомним как мы работали со списками в JavaScript:

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((number) => number * 2);
console.log(doubled);
```

В приведенном выше примере мы имеем список из чисел от 1 до 5, затем с помощью [метода map](#) мы проходимся по нему и получаем новый список, где каждое число было умножено на два, то есть [2, 4, 6, 8, 10]

Очень похожим образом это работает и в React, но, как правило, внутри map мы теперь будем возвращать какую-то разметку. Давайте представим, что нам каждое число нужно обернуть в отдельный div, вот как это может выглядеть:

```
function List() {  
  const numbers = [1, 2, 3, 4, 5];  
  
  return (  
    <div className="list">  
      {numbers.map((number) => (  
        <div className="number">{number}</div>  
      ))}  
    </div>  
  );  
}
```

Это возможно благодаря тому, что JSX позволяет встроить любое выражение внутри фигурных скобок. Рекомендую подробнее узнать про другие возможности [в этой статье](#) в документации.

Списки любой сложности

Отдельно стоит обратить внимание на то, что список может быть любой сложности и внутри списка вы можете вставлять как отдельные части разметки, так и целые компоненты.

Для примера давайте создадим компонент, который будет рендерить список из нескольких пользователей, причем каждый пользователь будет отдельным компонентом.

```

// Компонент ничего не знает откуда берутся данные
// Его задача лишь правильно их отобразить
function User(props) {
  const { user } = props;

  return (
    <div className="user">
      <div className="user__name">
        `${user.firstName} ${user.lastName}`
      </div>
      <div className="user__age">
        `Возраст: ${user.age}`
      </div>
    </div>
  );
}

function Users() {
  // Вот эти данные позднее можно вынести на бэкенд
  const users = [
    {
      firstName: "Yury",
      lastName: "Kundin",
      age: 27,
    },
    {
      firstName: "Yuliya",
      lastName: "Torseeva",
      age: 30,
    },
  ];

  return (
    <div className="users">
      {users.map((user) => (
        // Вставляем целый компонент
        <User user={user} />
      ))}
    </div>
  );
}

```

Ключи

Когда вы запустите любой из примеров выше, то увидите предупреждение, что у каждого элемента массива должен быть ключ (key). «Ключ» — это специальный строковый атрибут, который нужно указывать при создании **любого** списка элементов.

Ключи помогают React определять, какие элементы были изменены, добавлены или удалены. Их необходимо указывать, чтобы React мог сопоставлять элементы массива с течением времени:

```
function List() {
  const numbers = [1, 2, 3, 4, 5];

  return (
    <div className="list">
      {numbers.map((number) => (
        // Добавили ключ, свойство key
        <div className="number" key={number.toString()}>
          {number}
        </div>
      ))}
    </div>
  );
}
```

Лучший способ выбрать ключ — это использовать строку, которая будет явно отличать элемент списка от его соседей. Чаще всего вы будете использовать ID из ваших данных как ключи:

```
{users.map((user) => (
  <div key={user.id}>
    {user.firstName}
  </div>
))}
```

Когда у вас нет заданных ID для списка, то в крайнем случае можно использовать индекс элемента как ключ:

```
{users.map((user, index) => (  
  <div key={index}>  
    {user.firstName}  
  </div>  
)) }
```

Но в React не рекомендуется использовать индексы как ключи, если порядок элементов может меняться. Это негативно скажется на производительности и может вызвать проблемы с состоянием компонента. Если вы опустите ключ для элемента в списке, то React по умолчанию будет использовать индексы как ключи. Вот [подробное объяснение о том, почему ключи необходимы](#).

Извлечение компонентов с ключами

Ключи нужно определять непосредственно внутри массивов. Например, если вы извлекаете компонент `Listitem`, то нужно указывать ключ для `<Listitem />` в массиве, а не внутри самого `Listitem`.

Пример неправильного использования:

```
function ListItem(props) {  
  const { value } = props;  
  
  return (  
    // Неправильно! Нет необходимости задавать здесь ключ:  
    <li key={value.toString()}>{value}</li>  
  );  
}
```

```
function NumberList(props) {
  const { numbers } = props;

  return (
    <ul>
      {numbers.map((number) => (
        // Неправильно! Ключ необходимо определить здесь:
        <ListItem value={number} />
      ))}
    </ul>
  );
}
```

Пример правильного использования:

```
function ListItem(props) {
  // Правильно! Не нужно определять здесь ключ:
  return <li>{props.value}</li>;
}

function NumberList(props) {
  const { numbers } = props;

  return (
    <ul>
      {numbers.map((number) => (
        // Правильно! Ключ нужно определять внутри массива:
        <ListItem key={number.toString()} value={number} />
      ))}
    </ul>
  );
}
```

Ключи должны быть уникальными среди соседних элементов

Ключи внутри массива должны быть уникальными только среди своих соседних элементов. Им не нужно быть уникальными глобально. Можно использовать один и тот же ключ в двух разных массивах.

```
function Blog() {  
  const posts = [  
    {  
      id: 1,  
      title: "Привет, мир",  
      content: "Добро пожаловать в документацию React!",  
    },  
    { id: 2, title: "Установка", content: "React можно установить из npm." },  
  ];  
  
  return (  
    <div>  
      <ul className="sidebar">  
        {posts.map((post) => (  
          <li key={post.id}>{post.title}</li>  
        ))}  
      </ul>  
      <hr />  
      <div className="content">  
        {posts.map((post) => (  
          <div key={post.id}>  
            <h3>{post.title}</h3>  
            <p>{post.content}</p>  
          </div>  
        ))}  
      </div>  
    </div>  
  );  
}
```

Ключи служат подсказками для React, но они никогда не передаются в ваши компоненты. Если в компоненте нужно то же самое значение, то передайте его явно через проп с другим именем:

```
<div className="posts">
  {posts.map((post) => (
    <Post key={post.id} id={post.id} title={post.title} />
  ))}
</div>
```

В примере выше компонент `Post` может прочитать значение `props.id`, но не `props.key`.