

# JS Level 3



при поддержке



# REACT LISTS

# React

На прошлой лекции мы научились с вами создавать компоненты, поговорили, как всё устроено в React.

Сегодня наша задача - реализовать базовое приложение, которое умеет отображать список элементов. А в следующей лекции научимся не только отображать, а ещё добавлять и удалять элементы.

# Wall

Начнём мы с того, что создадим компонент, отвечающий за отображение списка. В социальных сетях его обычно называют либо "Стена" (Wall) либо "Лента" (Feed).

Мы назовём `Wall` и разместим в нём список наших постов для отображения.

# App.js

JS App.js ×

src > JS App.js > ...

```
1  import React from 'react';
2  import Wall from './components/Wall/Wall';
3
4  function App() {
5    return (
6      <div className="App">
7        <Wall />
8      </div>
9    );
10 }
11
12 export default App;
```

src &gt; components &gt; Wall &gt; JS Wall.js &gt; ...

```
1  import React from 'react';
2
3  function Wall() {
4    const posts = [
5      {
6        id: 2,
7        author: {
8          id: 1,
9          avatar: 'https://lms.openjs.io/logo_js.svg',
10         name: 'OpenJS',
11       },
12       content: 'Ну как, вы справились с домашкой?',
13       photo: null,
14       hit: true,
15       likes: 222,
16       likedByMe: true,
17       created: 1603774800,
18     },
19     {
20       id: 1,
21       author: {
22         id: 1,
23         avatar: 'https://lms.openjs.io/logo_js.svg',
24         name: 'OpenJS',
25       },
26       content: null,
27       photo: 'https://lms.openjs.io/openjs.jpg',
28       hit: true,
29       likes: 10,
30       likedByMe: true,
31       created: 1603501200,
32     },
33   ];
34
35   return (
36     <div>
37       ← будем разбираться, что писать
38     </div>
39   );
40 }
41
42 export default Wall;
```

# Wall.js

6

# Посты

Итак, пока поста у нас всего два, мы бы могли взять и прямо руками написать их (как делали на первой лекции):

```
> function Wall() {  
  const posts = [...]  
  ;  
  
  return (  
    <div>  
      <Post post={posts[0]} />  
      <Post post={posts[1]} />  
    </div>  
  );  
}  
  
export default Wall;
```

Но что, если их будет не 2, а 100 или 999? Мы же не знаем точное количество.

Тем более в современных социальных сетях они на самом деле подгружаются каждый раз снизу, когда вы листаете страницу.

# Циклы

В JS, как и в других языках программирования, существует специальная конструкция - цикл, которая позволяет делать повторяющиеся действия:

## Failed to compile

```
./src/components/Wall/Wall.js
Line 38:8:  Parsing error: Unexpected token

36 |   return (
37 |     <div>
> 38 |       {for (const post : posts) {
    |         ^
39 |         <Post post={posts[0]} />
40 |       }}
41 |     </div>
```

This error occurred during the build time and cannot be dismissed.



# Циклы

Но почему не работает? Всё дело в том, что JSX - это вызов функций и вы не можете туда вставить цикл. Просто не можете, это запрещено правилами языка.

Что же делать? Можно посмотреть на то, во что превращается JSX:

`React.createElement(element, props, ...children)`

... означает, что через запятую можно передать дочерние элементы.

Т.е. `<div> xxx </div>` это будет `React.createElement('div', {}, xxx);`

# Циклы

Т.е. можно сделать как-то так:

```
function Wall() {  
  React.createElement()  
>  const posts = [...]  
    ];  
  
  const postsEl = [];  
  for (const post of posts) {  
    postsEl.push(<Post post={post} />)  
  }  
  
  return (  
    <div>  
      {postsEl}  
    </div>  
  );  
}  
  
export default Wall;
```

Конечно, это будет работать. Но как-то это уж не совсем красиво. Какие-то циклы и т.д.

Давайте подумаем, что на самом деле мы делаем: мы хотим из массива постов сделать массив React Element'ов.

Из первого курса вы должны знать про методы массива `map` и `filter`. Они будут ключевыми при работе с React.

# map

`map` - это метод, позволяющий из одного массива создать другой массив, в котором все элементы будут преобразованы с помощью переданной функции преобразования:

```
function Wall(props) {  
  console.log(props);  
> const posts = [...  
  ];  
  
  return (  
    <div>  
      {posts.map(o => <Post post={o} />)}  
    </div>  
  );  
}  
  
export default Wall;
```

Как это работает? Мы берём все посты и последовательно применяем к ним функцию

`o => <Post post={o} />`

Т.е. из каждого элемента массива делаем `React.Element` и уже это передаём в качестве дочерних элементов.

Полную документацию на метод `map` вы можете найти [на странице MDN](#).

# map

Важно: привыкайте работать именно с `map`, поскольку если вы будете делать иначе, то вас не поймут коллеги.

# image

Теперь всё работает, то в интерфейсе мы видим, что вместо photo показывается атрибут alt (именно для этого он и был нужен):

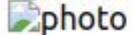


OpenJS

1603774800

HIT

Ну как, вы справились с домашкой?



# console

Давайте откроем консольку (F12) браузера и посмотрим, что происходит там:

```
✖ ▶ Warning: Each child in a list should have a unique "key" prop.    index.js:1

Check the render method of `Wall`. See https://fb.me/react-warning-keys for
more information.
  in Post (at Wall.js:39)
  in Wall (at App.js:7)
  in div (at App.js:6)
  in App (at src/index.js:9)
  in StrictMode (at src/index.js:8)

⚠ ▶ ./src/components/Post/Post.js    webpackHotDevClient.js:138
  Line 17:9:   Redundant alt attribute. Screen-readers already announce `img`
tags as an image. You don't need to use the words `image`, `photo`, or
`picture` (or any specified custom words) in the alt prop  jsx-ally/img-
redundant-alt
  Line 21:11:  img elements must have an alt prop, either with meaningful
text, or an empty string for decorative images
jsx-ally/alt-text
```

Там есть предупреждения. Более значимые подсвечиваются красным цветом, менее - жёлтым. Начнём с тех, что менее значимые.

# alt

Это предупреждение говорит нам о том, что на 17-ой строке нашего компонента Post, выставлен атрибут alt, который не имеет смысла ("photo"), а на 21-ой его вообще нет:

```
⚠ ./src/components/Post/Post.js webpackHotDevClient.js:138  
  Line 17:9:   Redundant alt attribute. Screen-readers already announce `img`  
tags as an image. You don't need to use the words `image`, `photo`, or  
`picture` (or any specified custom words) in the alt prop  jsx-ally/img-  
redundant-alt  
  Line 21:11:  img elements must have an alt prop, either with meaningful  
text, or an empty string for decorative images  
jsx-ally/alt-text
```

Бот будет интерпретировать подобные замечания как ошибки и начиная с сегодняшней лекции не будет принимать ДЗ с предупреждениями.

# alt

Давайте исправим:

```
{
  id: 1,
  author: {
    id: 1,
    avatar: 'https://lms.openjs.io/logo_js.svg',
    name: 'OpenJS',
  },
  content: null,
  photo: {
    url: 'https://lms.openjs.io/openjs.jpg',
    alt: 'openjs logo',
  },
  hit: true,
  likes: 10,
  likedByMe: true,
  created: 1603501200,
},
```



# alt

```
function Post({post}) {  
  const {author} = post;  
  const {photo} = post;  
  
  return (  
    <article>  
      <header>  
        <img src={author.avatar} className="Post-avatar" width="50" height="50" alt={author.name}/>  
        <h5>{author.name}</h5>  
        <div>{post.created}</div>  
        {post.hit && <span>HIT</span>}  
      </header>  
      <div>  
        <div className="Post-content">{post.content}</div>  
        <img src={photo.url} alt={photo.avatar} className="Post-photo"/>  
      </div>  
      <footer>  
        <span className="Post-likes">  
          <img  
            src={post.likedByMe ? 'https://lms.openjs.io/liked.svg' : 'https://lms.openjs.io/unliked.svg'}  
            alt="likes"  
            width="20"  
            height="20"  
          />  
          <span className="Post-likes-count">{post.likes}</span>  
        </span>  
      </footer>  
    </article>  
  );  
}
```

# null

Но теперь у нас проблемы посерьёзнее: ничего не работает, поскольку у одного поста photo - это объект, а другого - null:

**TypeError: Cannot read property 'url' of null**



Post

src/components/Post/Post.js:18

```
15 | </header>
16 | <div>
17 |   <div className="Post-content">{post.content}</div>
> 18 |   <img src={photo.url} alt={photo.avatar} className="Post-photo"/>
    |           ^
19 | </div>
20 | <footer>
21 |   <span className="Post-likes">
```

null говорит нам о том, что объекта нет. А значит, к его свойствам нельзя обращаться.

&amp;&amp;

В нашем случае решение достаточно простое:

```
<div>  
  <div className="Post-content">{post.content}</div>  
  {photo} && <img src={photo.url} alt={photo.avatar} className="Post-photo"/>  
</div>
```

# key

Теперь важное предупреждение:

```
✖ ▶ Warning: Each child in a list should have a unique "key" prop.    index.js:1  
  
Check the render method of `Wall`. See https://fb.me/react-warning-keys for  
more information.  
    in Post (at Wall.js:39)  
    in Wall (at App.js:7)  
    in div (at App.js:6)  
    in App (at src/index.js:9)  
    in StrictMode (at src/index.js:8)
```

О чём оно говорит? На прошлой лекции мы говорили о том, как React работает со своими элементами: что он сравнивает до изменения и после изменения два виртуальных DOM-дерева и применяет различия.

Со списками React'у тяжелее, потому что надо понимать, что будет меняться в списке (то ли новый элемент добавился, то ли старый изменился, то ли удалился, то ли просто поменял свою позицию). Поэтому, чтобы было легче, React просит нас указать уникальный ключ для этого элемента, чтобы понимать, что произошло.

# key

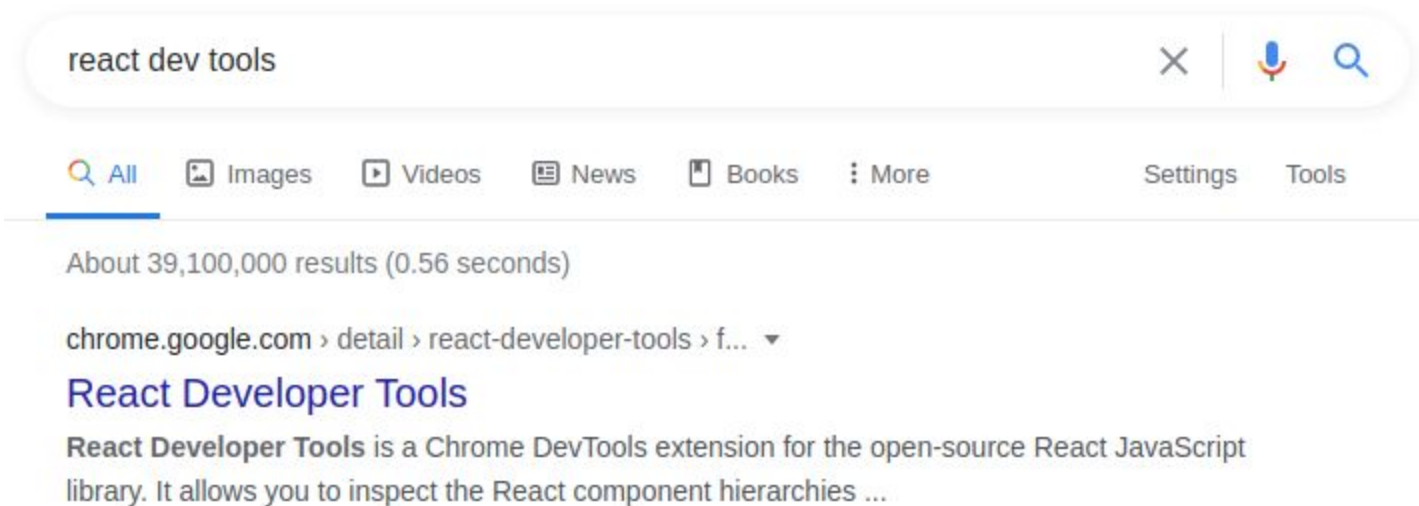
Как вы понимаете, на роль key отлично подходит id:

```
return (  
  <div>  
    {posts.map(o => <Post key={o.id} post={o} />)}  
  </div>  
);
```

Соответственно правило: всегда, когда работаете со списками, первым свойством передавайте **key**.

# key

Стоит отметить, что `key` - это специальное свойство, оно не передаётся в `props` вашего компонента. А как посмотреть, что на самом деле передаётся? Для этого вам необходимо зайти в поиске браузера React Dev Tools:



# React Dev Tools

[Home](#) > [Extensions](#) > React Developer Tools



## React Developer Tools

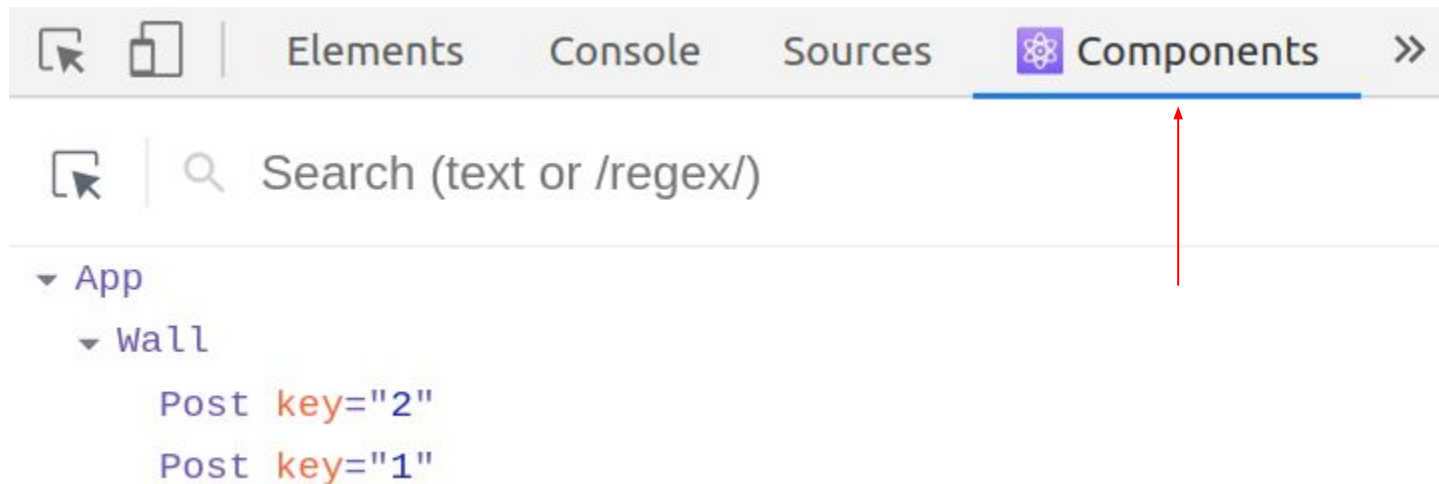
Offered by: Facebook

★★★★★ 1,287 | [Developer Tools](#) |  2,000,000+ users

Add to Chrome

# React Dev Tools

После этого нужно в Dev Tools (F12) перейти к панельке Components:

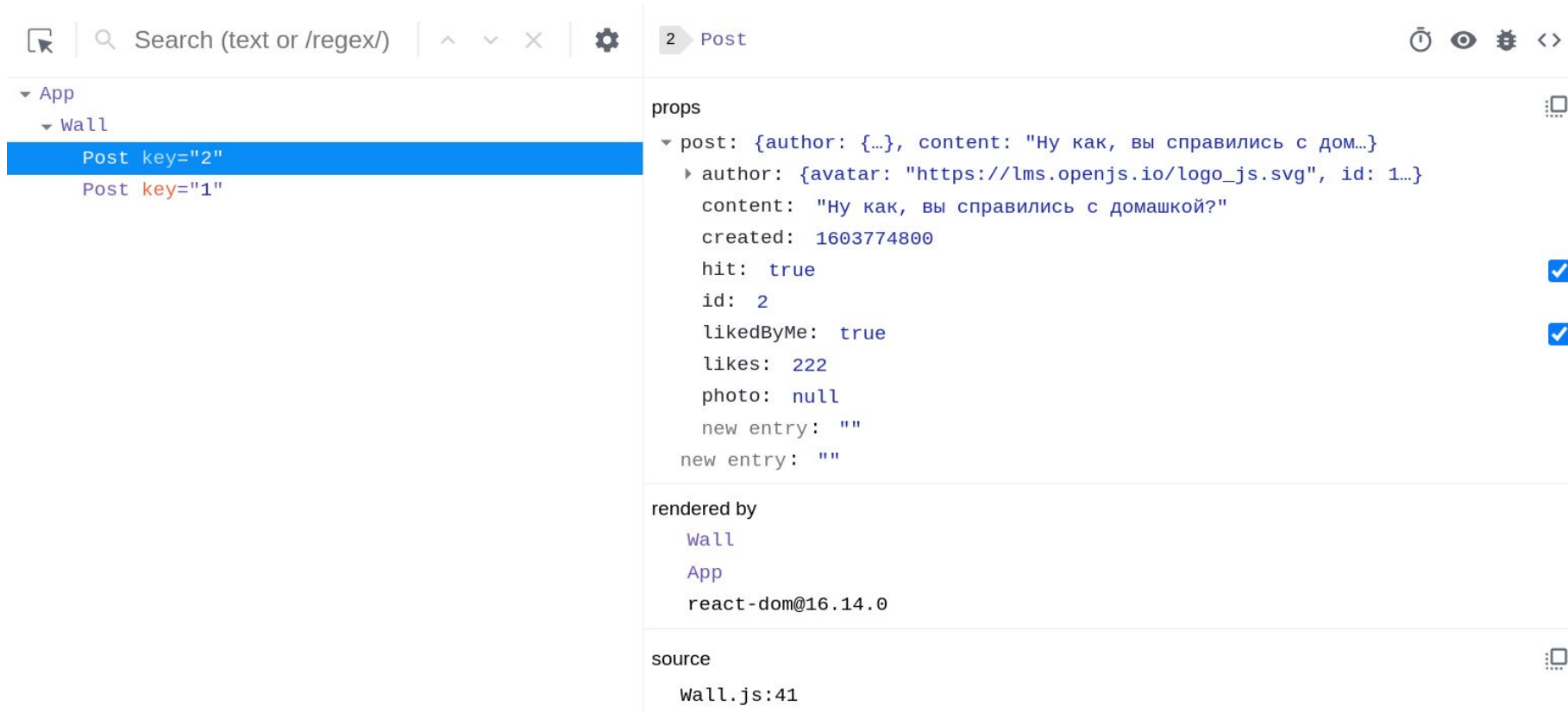


Если панельки нет, то нажмите на >> и найдите её там. Если и там нет, то полностью закройте и снова откройте браузер.



# React Dev Tools

Теперь вы можете выбрать любой компонент и посмотреть, какие props ему передаются:



The screenshot shows the React DevTools component inspector. On the left, the component tree shows 'App' containing 'Wall', which contains two 'Post' components. The 'Post' component with 'key="2"' is selected. The right pane shows the props for this component:

```
props
  post: {author: {...}, content: "Ну как, вы справились с дом..."}
    author: {avatar: "https://lms.openjs.io/logo_js.svg", id: 1...}
    content: "Ну как, вы справились с домашкой?"
    created: 1603774800
    hit: true
    id: 2
    likedByMe: true
    likes: 222
    photo: null
    new entry: ""
    new entry: ""
```

Below the props, it shows the component was rendered by 'Wall', 'App', and 'react-dom@16.14.0'. The source is 'Wall.js:41'.

Причём вы можете не только посмотреть, но и поменять их (при этом элемент перерисовывается).

# React Dev Tools

Обязательно используйте React Dev Tools в своей работе, это значительно облегчит вам жизнь.

# ИТОГИ

# Итоги

В этой лекции мы обсудили отображение списков. Списки - это один из ключевых элементов любого современного приложения. Поэтому мы выделили для них отдельную лекцию.

На следующей лекции мы поговорим о том, что такое состояние и обработка событий, а также познакомимся с хуком `useState` (да и вообще узнаем, что такое хуки).

# ДОМАШНЕЕ ЗАДАНИЕ

# ДЗ: Теги

Все любят теги (или хэш-теги): это такие специальные ссылки с # в названии, по которой пользователи могут кликать и искать подобные же записи:



OpenJS

1603774800

НIT

Ну как, вы справились с домашкой?

💖 222теги: [#deadline#homework](#)

Как вы видите, мы пока не заморачиваемся с оформлением, это не принципиально (но потом займёмся).

# ДЗ: Теги

Что нужно сделать:

1. Компонент `Tags`, который отображает список тегов и слово `теги`
2. Сам компонент `Tags` отображается только тогда, когда в объекте поста есть свойство `tags`:

```
{
  id: 2,
  author: {
    id: 1,
    avatar: 'https://lms.openjs.io/logo_js.svg',
    name: 'OpenJS',
  },
  content: 'Ну как, вы справились с домашкой?',
  photo: null,
  hit: true,
  likes: 222,
  likedByMe: true,
  tags: ['deadline', 'homework'],
  created: 1603774800,
},
```

```
▼ <footer> == $0
  ▼ <span class="Post-likes">
    
    <span class="Post-likes-count">222</span>
    "теги: "
  ▼ <button>
    "#"
    "deadline"
  </button>
  ▼ <button>
    "#"
    "homework"
  </button>
</span>
</footer>
```

# ДЗ: Теги

Что этот компонент принимает в качестве props:

	Tags
<div>▼ App</div> <div>▼ Wall</div> <div>▼ Post key="2"</div> <div>Tags</div>	<p>props</p> <ul style="list-style-type: none"><li>tags: ["deadline", "homework"]</li><li>new entry: ""</li></ul>
<div>Post key="1"</div>	<p>rendered by</p> <ul style="list-style-type: none"><li>Post</li><li>Wall</li><li>App</li><li>react-dom@16.14.0</li></ul>
	<p>source</p> <p>Post.js:30</p>



# ДЗ: Теги

Но что тогда использовать в качестве key? Небольшая подсказка: теги внутри массива уникальны.

# Подсказка

Иногда требуется, чтобы компонент генерировал код, но без лишних тегов (например, как в задаче выше) - т.е. без всяких `span`, `div` и т.д.

Мы можем написать просто вот так:

```
function Tags({tags}) {  
  return (  
    'теги'  
  );  
}
```

Но тогда непонятно, как дописывать сами теги. Почему? Потому что `return` из компонента должен возвращать всегда ровно один `React Element`.

# Подсказка

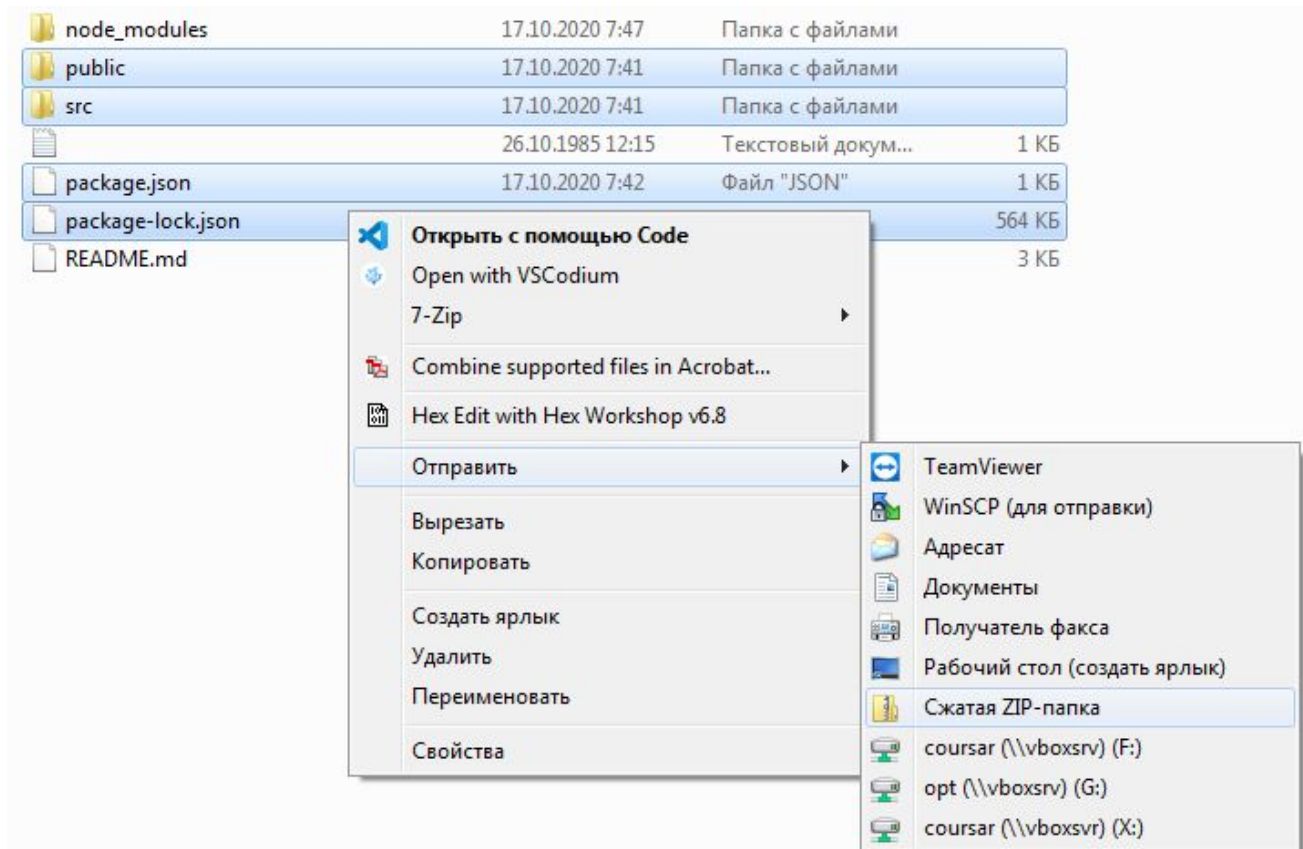
В этих случаях используется специальный пустой тег, который позволяет создать React Element, не генерирующий нового DOM-тега. Т.е.:

```
function Tags({tags}) {  
  return (  
    <>  
    теги:  
    </>  
  );  
}
```

А дальше внутри `<></>` вы можете спокойно размещать всё, что нужно.

# Как сдавать ДЗ

Вам нужно запаковать в zip-архив ваш проект те файлы и каталоги, которые указаны на скриншоте ниже. Для этого выберите их, нажмите правую кнопку мыши и выберите Отправить → Сжатая ZIP-папка:



# Как сдавать ДЗ

Полученный архив загружаете в личном кабинете пользователя.

**Важно:** учитывается только последняя отправленная попытка.

Спасибо за внимание

alif academy совместно с aims  
2020г.