JS Level 2

REACT EVENTS

React

На прошлой лекции мы научились отображать список элементов. Сегодня же наша задача – удалять элементы, а также частично модифицировать их (через кнопку like).

События

В рамках DOM API существует концепция событий: т.е. мы можем "повесить" на определённый элемент слушатель или обработчик события (функцию), которая будет срабатывать каждый раз, когда это событие произойдёт. Например, в index.js нашего приложения мы можем написать:

```
const rootEl = document.getElementById('root');
// event handler
rootEl.onclick = (evt) => {
  console.log('handler: onclick');
  console.log(evt);
};

// event listener
rootEl.addEventListener('click', (evt) => {
  console.log('listener: click');
  console.log(evt);
});
```

События

В DevTools мы увидим:

```
handler: clicked
▶ MouseEvent {isTrusted: true, screenX: 4734, screenY: 374, clientX: 202, clientY: 265, …}
listener: clicked
▶ MouseEvent {isTrusted: true, screenX: 4734, screenY: 374, clientX: 202, clientY: 265, …}
```

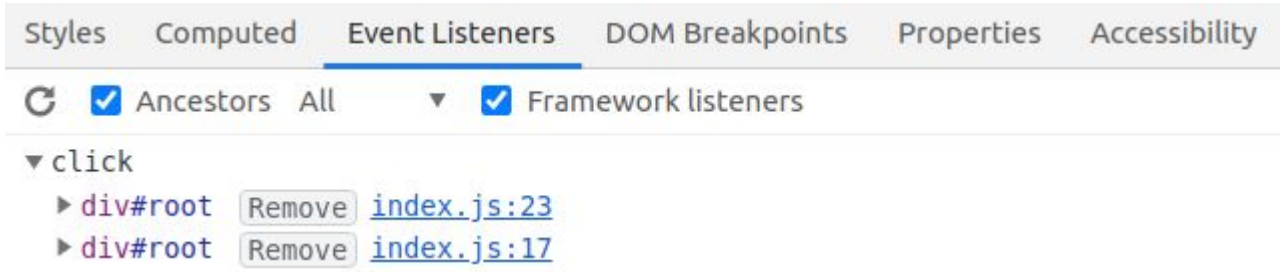
Handlers vs Listeners

Как вы видели на одном из предыдущих слайдов, назначать события можно как через свойство (onclick – это называется handler), так и через добавление (addEventListener – это называется listener).

Ключевая разница в том, что handler можно назначить только один, а вот listener'ов сколько угодно. В лекциях первого уровня детально описывается процесс обработки событий.

Handlers vs Listeners

В панельке Elements мы можем увидеть все назначенные на элемент handler'ы и listener'ы (для этого нужно выбрать соответствующий элемент в дереве):



Vanilla JS

В чистом JS допускается устанавливать обработчики только одним из указанных на предыдущих слайдов способов (при этом предпочтительным является подход с listener'ами).

Но существует и третий способ: мы можем прямо в HTML прописать handler (уберите код с назначением через js (в index.js), чтобы это заработало):

```
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root" onclick="console.log('html handler');"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
```

Handler in HTML

Несмотря на то, что данный подход работает, это считается дурным тоном - смешивать HTML и JS (поэтому старайтесь так не делать).

Но он вполне рабочий и даже используется в промышленных решениях, например в Vk:

```
▼<div id="post-149187028_369" class="_post post page_block all own post--with-likes deep_active"
data-post-id="-149187028_369" onclick="wall.postClick('-149187028_369', event);" post_view_hash=
"de50b82cf94bb14003">
```

React

Поскольку в React мы используем JSX, то было принято решение использовать подход, аналогичный подходу с назначением handler'ов в HTML, за рядом исключений:

1. Собственные названия handler'ов (в формате onClick, а не onclick)
2. Функции указываются в {}, а не в ""
3. Специальный объект события (не нативный DOM Event)
4. Немного отличающееся поведение (об этом чуть позже)

Полный список поддерживаемых событий указан в документации.

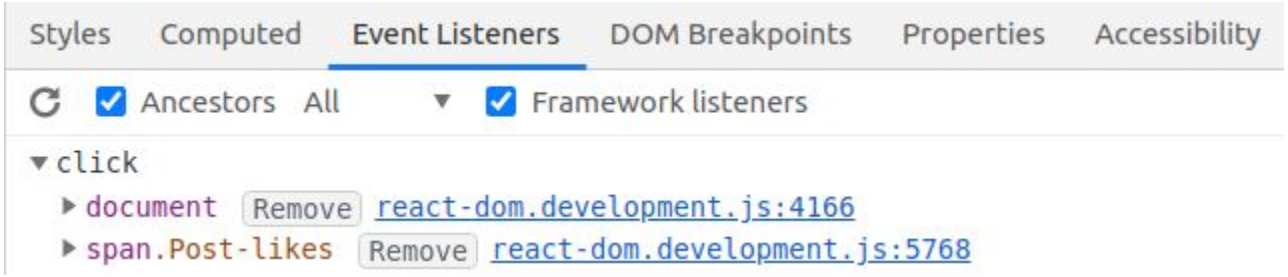
React

```
function Post({post}) {
  const {author} = post;
  const {photo} = post;

  const handleClick = (evt) => {
    console.log('like clicked');
  };

  return (
    <article>
      <header>…
      <div>…
      <footer>
        <span className="Post-likes" onClick={handleClick}>…
      </footer>
    </article>
  );
}

export default Post;
```

Т.е. всё достаточно просто: назначаем функцию, которую сами же определяем в том же компоненте.

React

Смотрим в панельку Listeners и видим, что React за нас сам назначил обработчик на событие click:



Плюс один есть на самом документе, но нас он пока не интересует.

Like

Это всё хорошо (что мы можем выводить что-то в консоль), но реальной пользы мало. При клике на лайк пост должен лайкаться (если до этого не был залайкан), либо лайк должен сниматься:

```
function Post({post}) {
  const {author} = post;
  const {photo} = post;

  const handleClick = (evt) => {
    if (post.likedByMe) {
      post.likedByMe = false;
      post.likes--;
      return;
    }

    post.likedByMe = true;
    post.likes++;
  };

  return (
    <article>…
  );
}

export default Post;
```

Но это не работает.

Q: почему?

A: потому что мы изменяем объект, за которым React не следит. Ему всё равно, что мы изменяем likes, likedByMe - их изменения не приводят к перерисовке компонента.

Debugger

Давайте убедимся, что свойства действительно меняются, установим специальную инструкцию debugger, которая откроет отладчик браузера в нужной точке:

```
const handleClick = (evt) => {
  debugger;
  if (post.likedByMe) {
    post.likedByMe = false;
    post.likes--;
    return;
  }

  post.likedByMe = true;
  post.likes++;
};
```

```
const handleClick = (evt) => {  evt = Class {dispatchConfig: {…},
  debugger;
  if (post.likedByMe) {
    post.likedByMe = false;
    post.likes--;
    return;
  }

  post.likedByMe = true;
  post.likes++;
};
```

Как пользоваться отладчиком вы можете прочитать из лекций первого уровня.

props

В предыдущем коде мы сделали достаточно нехорошую вещь, которую вы никогда не должны делать в коде React: мы попытались поменять props. В React всё, что передаётся в props – это только для чтения (т.е. не нужно его изменять).

state

Q: хорошо, если не props, то как? Как нам изменить данные и запустить перерисовку?

A: для этого в React есть термин state (состояние).

Состояние – это возможность компонента сохранять информацию между собственными перерисовками. Изменение состояния приводит к новой перерисовке (т.е. React запускает сравнение нового Virtual DOM со старым).

Functional Components

Компоненты на базе функций, которые мы с вами и используем, называются функциональными компонентами (functional components). Помимо них есть ещё компоненты на базе классов (class based components), но они сейчас потихоньку выходят из употребления, поэтому мы будем изучать только functional components.

Хуки

Так вот в functional components, чтобы получить доступ к функциональности, выходящей за рамки кода вашего компонента есть специальная возможность, которая называется хуки.

Хуки – это специальные функции, которые позволяют вам "вклиниться" в жизненный цикл React.

Хуки появились только в React 16.8, поэтому если вы попадёте на "старый" проект, то там хуков не будет.

useState

Сегодня нас будет интересовать специальный хук useState – он будет позволять использовать состояние в наших функциональных компонентах.

useState нужно импортировать из React:

Post.js

src > components > Post > Post.js > ...

```
1  import React, {useState} from 'react';
2  import './Post.css';
3  import Tags from '../Tags/Tags';
```

useState

```
function Post({post}) {
  const {author} = post;
  const {photo} = post;

  const [likedByMe, setLikedByMe] = useState(post.likedByMe);
  const [likes, setLikes] = useState(post.likes);

  const handleClick = (evt) => {
    if (post.likedByMe) {
      setLikedByMe(false);
      setLikes(post.likes - 1);
      return;
    }

    setLikedByMe(false);
    setLikes(post.likes - 1);
  };
```

useState принимает в качестве аргумента начальное значение состояния и возвращает массив, состоящий из двух элементов:

1. Текущее состояние
2. Функцию для изменения текущего состояния

Чтобы аккуратно "распаковать" этот массив в переменные, мы используем деструктуризацию.

Функция для изменения текущего состояния нужна для того, чтобы React видел, что вы меняете состояние и перерисовывал компонент.

useState

```
return (
  <article>
    <header>…
    <div>…
    <footer>
      <span className="Post-likes" onClick={handleClick}>
        <img
          src={post.likedByMe ? 'https://alif-skills.pro/media/liked.svg' : 'https://alif-skills.pro/media/unliked.svg'}
          alt="likes"
          width="20"
          height="20"
        />
        <span className="Post-likes-count">{likes}</span>
        {post.tags && <Tags tags={post.tags} />}
      </span>
    </footer>
  </article>
);
```

useState

Но так это сработает всего один раз.

Q: почему?

A: дело в том, что мы меняем состояние, но при этом сам пост не меняется.

Кстати, состояние вы можете посмотреть с помощью React DevTools:

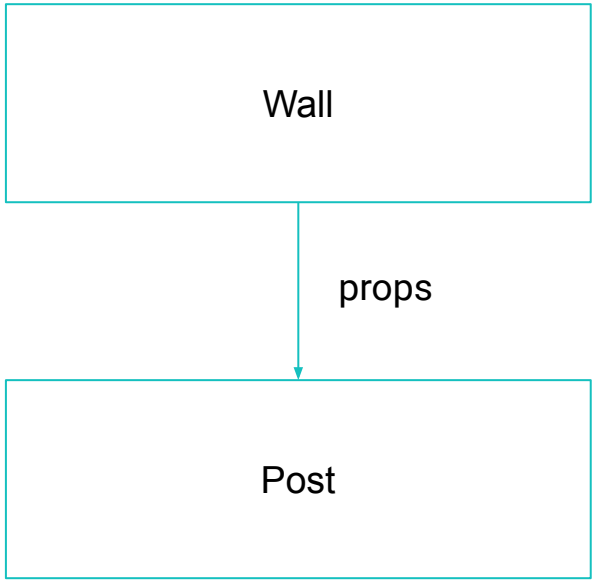
Владелец данных

И вот здесь возникает важный вопрос: а где мы должны поменять данные? Данные же хранятся не в самом компоненте Post, они хранятся в компоненте Wall. Именно компонент Wall "владелец данных", т.е. наш массив с данными принадлежит ему.

И если он владелец, то менять данные должен только он.

Схема взаимодействия

Давайте посмотрим на схему взаимодействия компонентов:



Данные всегда идут сверху-вниз: Post ничего не знает о Wall. Но как тогда Post сообщит Wall о том, что произошло какое-то событие? Например, клик на кнопке лайк?

Callback'и

Поскольку React компоненты в целом призваны имитировать поведение DOM элементов, то идея та же самая. Почему бы нам не передать свой handler или callback в дочерний компонент (так же, как мы делаем с onClick)? Тогда дочерний компонент сможет вызывать этот callback, когда событие произойдёт.

Callback'и

```
function Post({post, onLike}) {
  const {author} = post;
  const {photo} = post;

  const handleClick = () => {
    onLike(post.id);
  };

  return (
    <article>
      <header>…
      <div>…
      <footer>
        <span className="Post-likes" onClick={handleClick}>
          <img
            src={post.likedByMe ? 'https://alif-skills.pro/media/liked.svg' : 'https://alif-skills.pro/media/unliked.svg'}
            alt="likes"
            width="20"
            height="20"
          />
          <span className="Post-likes-count">{post.likes}</span>
          {post.tags && <Tags tags={post.tags} />}
        </span>
      </footer>
    </article>
  );
}

export default Post;
```

Wall

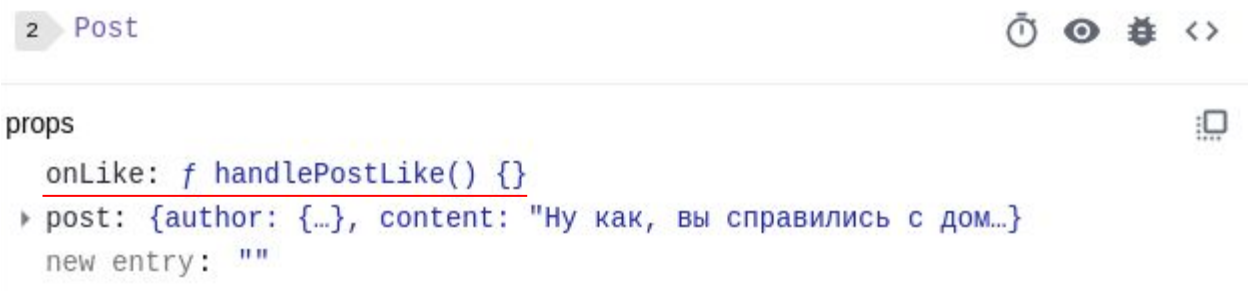
Wall.js

src > components > Wall > Wall.js > ...

```
1   import React, {useState} from 'react';
2   import Post from '../Post/Post';
3
4   function Wall(props) {
5     const [posts, setPosts] = useState([…
39
40    const handlePostLike = (id) => {
41      // TODO:
42    };
43
44    return (
45      <div>
46        {posts.map(o => <Post key={o.id} post={o} onLike={handlePostLike} />)}
47      </div>
48    );
49  }
50
51  export default Wall;
```

Обратите внимание: мы положили в state начальный список постов (элементы из него мы и будем изменять).

React Dev Tools



map

А теперь самые важные две вещи, которые нужно запомнить:

1. Любое "изменение объектов массива" производится с помощью функции map (конечно же, мы не изменяем на самом деле массив, а из старого массива делаем новый)
2. При использовании setState (setPosts в нашем случае), основанном на предыдущем состоянии (мы из старого состояния с неизменёнными лайками делаем новое) **всегда** используется форма (prevState) => nextState (в Post мы использовали не такую форму, потому что "пытались" жёстко задать значение).

map

```
function Wall(props) {
  const [posts, setPosts] = useState([…

  const handlePostLike = (id) => {
    setPosts((prevState) => prevState.map(o => {
        if (o.id !== id) {
          return o;
        }

        const likedByMe = !o.likedByMe;
        const likes = likedByMe ? o.likes + 1 : o.likes - 1;
        return {...o, likedByMe, likes};
    }));
  };

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

export default Wall;
```

map

Давайте попробуем разобраться:

setPosts((prevState) => prevState.map(...) - на базе предыдущего состояния (а там массив постов до нажатия на кнопке like) делаем новое состояние (его React и установит, после чего перерисует наш компонент).

```
if (o.id !== id) { // если id поста не равен тому, который лайкнули
    return o;       // то ничего с ним не делаем, просто возвращаем дальше
}
```

map

Если мы не вышли по return (см. предыдущий слайд), то значит id поста равен тому, который мы лайкнули, и надо его обновить. Но в React мы не обновляем объекты, мы создаём их копии с изменёнными значениями:

```
const likedByMe = !o.likedByMe;
const likes = likedByMe ? o.likes + 1 : o.likes - 1;
return {...o, likedByMe, likes};
```

Во-первых, создаём сами значения likedByMe и likes.

Во-вторых, копируем объект с помощью синтаксиса {...o} - в таком виде использование ... означает, что мы "как бы" пишем новый объект в виде: {id: o.id, author: o.author, и т.д.}.

При этом при копировании мы можем заменить часть свойств:

{id: o.id, ..., likedByMe: o.likedByMe, likedByMe: likedByMe, ...}

При такой записи в объекте останется именно то значение, которое записано последним (подчёркнуто красным).

map

Мы специально вам не показываем никаких других вариантов записи, потому что вы будете пользоваться именно этим форматом.

Вам нужно установить дебаггер и хорошо разобраться в том, что происходит, т.к. в дальнейшем выражения будут только усложняться.

filter

В React вы будете использовать два ключевых метода массива при работе с состояниями, содержащими набор объектов:

1. map;
2. filter.

map - это для "обновления", filter - для "удаления". Давайте попробуем его (удаление) реализовать (мы не будем делать кнопку, её вы сделаете в рамках ДЗ):

```
const handlePostRemove = (id) => {
  setPosts((prevState) => prevState.filter(o => o.id !== id));
}
```

Мы просто создаём новый массив постов без поста с переданным id и кладём его в state. О всём остальном React позаботиться сам.

ИТОГИ

Итоги

В этой лекции мы обсудили базовую обработку событий и посмотрели на состояние. Так же мы разобрали типичные ошибки (вроде изменения props) и важный вопрос владения данными.

ДОМАШНЕЕ ЗАДАНИЕ

Орг.моменты

Практикум состоит из 8 обязательных занятий. Мы выкладываем новые занятия каждый понедельник в 14:00 (по Душанбе).

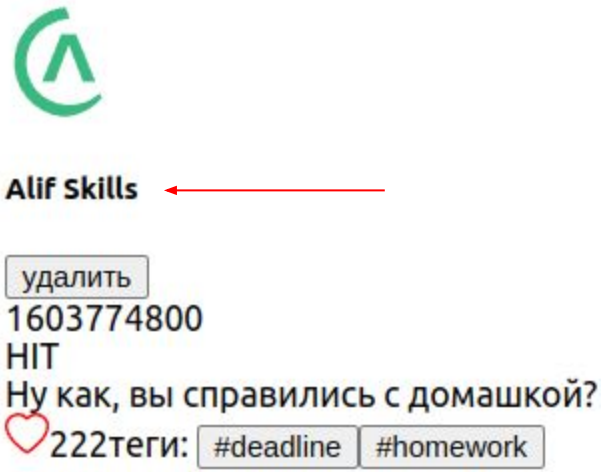
Каждое воскресенье в 23:59 (по Душанбе) дедлайн сдачи домашнего задания. Дедлайн – это предельный срок, до которого вы должны сдать ДЗ.

Если не успеете сдать в срок домашнее задания, тогда этот практикум будет для вас закончен и вы сможете зарегистрироваться на запуск следующего через несколько месяцев.

Все вопросы вы сможете задавать в Телеграм канале.

ДЗ: Удаление

Реализуйте функцию удаления на базе той, что описана в лекции:



ДЗ: Удаление


<article>
<header>
<img src="https://alif-skills.pro/media/logo_alif.svg" class="Post-avatar" width="50"
height="50" alt="Alif Skills">
<h5>Alif Skills</h5>
<button>удалить</button> == $0
1603774800
HIT
</header>
 ...
<footer> ... </footer>
</article>

ДЗ: Удаление


2 Post
props
onLike: ƒ handlePostLike() {}
onRemove: ƒ handlePostRemove() {}
post: {author: {…}, content: "Ну как, вы справились с дом…}
new entry: ""

ДЗ: Удаление

Используйте следующие данные:

```
{
  id: 2,
  author: {
    avatar: 'https://alif-skills.pro/media/logo_alif.svg',
    name: 'Alif Skills',
  },
  content: 'Ну как, вы справились с домашкой?',
  photo: null,
  hit: true,
  likes: 222,
  likedByMe: true,
  tags: ['deadline', 'homework'],
  created: 1603774800,
},
{
  id: 1,
  author: {
    id: 1,
    avatar: 'https://alif-skills.pro/media/logo_alif.svg',
    name: 'Alif Skills',
  },
  content: null,
  photo: {
    url: 'https://alif-skills.pro/media/meme.jpg',
    alt: 'Мем про дедлайн',
  },
  hit: true,
  likes: 10,
  likedByMe: true,
  created: 1603501200,
},
```

ДЗ: Скрытие

Мы хотим иметь возможность скрывать некоторые посты из ленты (не удалять, а именно скрывать):

```
<img src={author.avatar} className="Post-avatar" width="50" height="50" alt={author.name}/>
<h5>{author.name}</h5>
<button onClick={handleRemove}>удалить</button>
<button onClick={handleHide}>скрыть</button>
<div>{post.created}</div>
{post.hit && <span>HIT</span>}
```

ДЗ: Скрытие

При этом в самих постах появляется поле hidden:

```
const [posts, setPosts] = useState([
  {
    id: 2,
    author: {
      id: 1,
      avatar: 'https://alif-skills.pro/media/logo_alif.svg',
      name: 'Alif Skills',
    },
    content: 'Ну как, вы справились с домашкой?',
    photo: null,
    hit: true,
    likes: 222,
    likedByMe: true,
    hidden: true,
    tags: ['deadline', 'homework'],
    created: 1603774800,
  },
  {
    id: 1,
    author: {
      id: 1,
      avatar: 'https://alif-skills.pro/media/logo_alif.svg',
      name: 'Alif Skills',
    },
    content: null,
    photo: {
      url: 'https://alif-skills.pro/media/meme.jpg',
      alt: 'Мем про дедлайн',
    },
    hit: true,
    likes: 10,
    likedByMe: true,
    hidden: false,
    created: 1603501200,
  },
]);
```

ДЗ: Скрытие

Если оно (поле hidden) равно true, то мы видим следующее:

```
▼<article>
  ▼<header>
     <img src="https://alif-skills.pro/media/logo_alif.svg" class="Post-avatar" width="50"
     height="50" alt="Alif Skills">
     <h5>Alif Skills</h5>
     <button>показать</button> == $0
   </header>
 </article>
```

Сам компонент:

```
function Post({post, onLike, onRemove, onHide, onShow}) {
```


отвечает за показ поста

Фактически, вся задача сводится к изменению свойства hidden.

ДЗ: Скрытие

Подсказка: не обязательно "заморачиваться" с JSX. Вы вполне можете в компоненте делать:

```
if (условие) {
  return (
    <>Первый вариант "разметки"</>
  );
}

return (
  <>Второй вариант "разметки"</>
);
```

Спасибо за внимание

alif skills

2023г.