JS Level 3

при поддержке

open (js)
school

REACT FORMS

open (js)
school

React

На прошлой лекции мы научились добавлять элементы, сегодня продолжим этот разговор.

Объединяем

Мы остановились на вот такой структуре:

```
const handleChange = (ev) => {
  const {name, value} = ev.target;
  if (name === 'tags') {
    const parsed = value.split(' ');
    setPost((prevState) => ({...prevState, [name]: parsed}));
    return;
  }

  setPost((prevState) => ({...prevState, [name]: value}));
};

return (
  <form onSubmit={handleSubmit}>
    <textarea name="content" value={post.content} onChange={handleChange}></textarea>
    <input name="tags" value={post.tags?.join(' ')} onChange={handleChange}></input>
    <button>Ok</button>
  </form>
)
```

Типичная ошибка

Если мы посмотрим в консоль, то увидим:

```
▸Warning: A component is changing an uncontrolled input of type undefined  index.js:1
to be controlled. Input elements should not switch from uncontrolled to controlled (or
vice versa). Decide between using a controlled or uncontrolled input element for the
lifetime of the component. More info: https://fb.me/react-controlled-components
    in input (at PostForm.js:47)
    in form (at PostForm.js:45)
    in PostForm (at Wall.js:75)
    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)
```

В чём дело? Дело в том, что post.tags?.join(' ') возвращает undefined, если post.tags будет null. Но значением поля value не может быть undefined. Поэтому обычно добавляют оператор ||, который работает следующим образом: возвращает первый операнд, который truthy или последний, если оба falsy.

Очищение ввода

Теперь ключевой вопрос: как очистить поля ввода. Ведь после нажатия на кнопке OK у нас остаётся введённым текст.

Всё достаточно просто: нужно в handleSubmit после onSave выставить в state "чистый" объект (см. следующий слайд).

Очищение ввода

```
const handleSubmit = (ev) => {
  ev.preventDefault();
  const parsed = post.tags?.map(o => o.replace('#', '')).filter(o => o.trim() !== '') || [];
  const tags = parsed.length !== 0 ? parsed : null;
  onSave({...post, id: Date.now(), created: Date.now(), tags, photo: post.photo?.url ? {alt: '', ...post.photo} : null});
  setPost({
    id: 0,
    author: {
      avatar: 'https://lms.openjs.io/logo_js.svg',
      name: 'OpenJS',
    },
    content: '',
    photo: null,
    hit: false,
    likes: 0,
    likedByMe: false,
    hidden: false,
    tags: null,
    created: 0,
  });
};
```

Обратите внимание, мы заменили id и created на 0 и только в handleSubmit выставляем их. Так же, как и с предыдущей ошибкой вы должны были догадаться по сообщениям в консоли, что иначе у вас будут одинаковые id для разных элементов. Конечно, после очистки они будут уже не одинаковые, но дата создания всё равно будет неверной.

React

Конечно, глядя на этот код возникает желание вынести чистый объект один раз и не делать одно и то же в useState и в handleSubmit (см. следующий слайд).

```
const [post , setPost] = useState({
    id: 0,
    author: {
      avatar: 'https://lms.openjs.io/logo_js.svg',
      name: 'OpenJS',
    },
    content: '',
    photo: null,
    hit: false,
    likes: 0,
    likedByMe: false,
    hidden: false,
    tags: null,
    created: 0,
});

const handleSubmit = (ev) => {
  ev.preventDefault();
  const parsed = post.tags?.map(o => o.replace('#', '')).filter(o => o.trim() !== '') || [];
  const tags = parsed.length !== 0 ? parsed : null;
  onSave({...post, id: Date.now(), created: Date.now(), tags, photo: post.photo?.url ? {alt: '', ...post.photo} : null});
  setPost({
    id: 0,
    author: {
      avatar: 'https://lms.openjs.io/logo_js.svg',
      name: 'OpenJS',
    },
    content: '',
    photo: null,
    hit: false,
    likes: 0,
    likedByMe: false,
    hidden: false,
    tags: null,
    created: 0,
  });
};
```

React

Давайте попробуем это сделать и посмотрим, к чему это приведёт:

```
const empty = {
  id: 0,
  author: {
    avatar: 'https://lms.openjs.io/logo_js.svg',
    name: 'OpenJS',
  },
  content: '',
  photo: null,
  hit: false,
  likes: 0,
  likedByMe: false,
  hidden: false,
  tags: null,
  created: 0,
};

export default function PostForm({onSave}) {
  const [post , setPost] = useState(empty);

  const handleSubmit = (ev) => {
    ev.preventDefault();
    const parsed = post.tags?.map(o => o.replace('#', '')).filter(o => o.trim() !== '') || [];
    const tags = parsed.length !== 0 ? parsed : null;
    onSave({...post, id: Date.now(), created: Date.now(), tags, photo: post.photo?.url ? {alt: '', ...post.photo} : null});
    setPost(empty);
  };
```

React

Это работает, но нужно быть достаточно аккуратным, поскольку если вы случайно поменяете сам объект empty - то ничего хорошего не будет.

Для этого обычно используют создание копии через {...empty}, но здесь вас подстерегает опасность: таким образом создаётся поверхностная копия, которая не копирует объекты, хранящиеся в полях. Например, поле author у всех скопированных подобным образом объектов будет указывать на один и тот же объект.

Соответственно, некоторые используют либо JSON.parse(JSON.stringify()) (плохое решение), либо используют собственные/библиотечные функции полного копирования.

useRef

С нашей формой добавления вроде всё достаточно неплохо, за одним исключением: после добавления фокус (то, где расположен курсор) остаётся в последнем поле, а неплохо бы переставить его на первое (обязательно следите за такими "мелочами").

Фокус - это функциональность из мира DOM, а не React. Поэтому в React для этого придумали специальный хук, который позволяет в том числе "добираться" до элементов.

useRef

Идея useRef достаточно проста - он позволяет хранить ссылку на объект (не обязательно DOM Element), и изменение этой ссылки не приводит к перерисовке компонента (перерендерингу).

Как использовать (см. следующий слайд).

useRef

```
export default function PostForm({onSave}) {
  const [post, setPost] = useState(empty);
  const firstFocusEl = useRef(null); // начальное значение

  const handleSubmit = (ev) => {
    ev.preventDefault();
    const parsed = post.tags?.map(o => o.replace('#', '')).filter(o => o.trim() !== '') || [];
    const tags = parsed.length !== 0 ? parsed : null;
    onSave({...post, id: Date.now(), created: Date.now(), tags, photo: post.photo?.url ? {alt: '', ...post.photo} : null});
    setPost(empty);
    firstFocusEl.current.focus(); // через свойство current получаем доступ и устанавливаем focus
  };

  const handleChange = (ev) => {…
  };

  return (
    // через ref = устанавливаем ссылку на элемент (React сам установит ссылку)
    <form onSubmit={handleSubmit}>
      <textarea ref={firstFocusEl} name="content" placeholder="content" value={post.content} onChange={handleChange}></textarea>
      <input name="tags" placeholder="tags" value={post.tags?.join(' ') || ''} onChange={handleChange}></input>
      <button>Ok</button>
    </form>
  )
```

Читабельность кода

Наш код работает, но вы могли заметить, что строки очень длинные и их тяжело читать.

Поэтому не ленитесь и форматируйте свой код так, чтобы его удобно было читать,

например:

```
  onSave({
    ...post,
    id: Date.now(),
    created: Date.now(),
    tags,
    photo: post.photo?.url ? {alt: '', ...post.photo} : null
  });
  setPost(empty);
  firstFocusEl.current.focus(); // через свойство current получаем доступ и устанавливаем focus
};
```

Читабельность кода

Кроме того, закрывайте пустые теги:

```
return (
  // через ref = устанавливаем ссылку на элемент (React сам установит ссылку)
  <form onSubmit={handleSubmit}>
    <textarea
      ref={firstFocusEl}
      name="content"
      placeholder="content"
      value={post.content}
      onChange={handleChange}/>
    <input
      name="tags"
      placeholder="tags"
      value={post.tags?.join(' ') || ''}
      onChange={handleChange}
    />
    <button>Ok</button>
  </form>
)
```

Читабельность кода

Обратите внимание: мы не сразу написали идеальный код (такого не бывает), мы пришли к нему в процессе и навели "красоту" тогда, когда код стал работать.

React

Теперь давайте поговорим о редактировании. С редактированием есть два подхода:

1. На редактирование вы делаете отдельную форму
2. Вы выполняете редактирование в той же форме, что и добавление

Мы рассмотрим с вами второй вариант, т.к. первый будет лишь его вариацией.

React

Итак поехали: чтобы отредактировать пост, нам нужно каким-то образом в форму редактирования передать тот пост, который мы собираемся редактировать. Конечно же, пока мы это можем сделать только через props. Но до этого нам нужно добавить кнопку "Изменить" в пост и соответствующие функции в Wall (см. следующий слайд):

```
return (
  <article>
    <header>
      <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>
      <button onClick={handleEdit}>изменить</button>
      <div>{post.created}</div>
      {post.hit && <span>HIT</span>}
    </header>
```

```
const handleEdit = () => {
  onEdit(post.id);
};
```

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

  const handlePostLike = (id) => {…
  };

  const handlePostRemove = (id) => {…
  };

  const handleTogglePostVisibility = (id) => {…
  };

  const handlePostEdit = (id) => {
    const post = posts.find(o => o.id === id);
    if (post === undefined) {
      return;
    }

    setEdited(post);
  };

  const handlePostSave = (post) => {…
  };

  return (
    <>
      <PostForm edited={edited} onSave={handlePostSave} />
      <div>
        {posts.map(o => <Post
                          key={o.id}
                          post={o}
                          onLike={handlePostLike}
                          onRemove={handlePostRemove}
                          onHide={handleTogglePostVisibility}
                          onEdit={handlePostEdit}
                          onShow={handleTogglePostVisibility} />)}
      </div>
    </>
  );
}

export default Wall;
```

React

JS PostForm.js ×

src > components > PostForm > JS PostForm.js > ...

```
1   import React, { useState, useRef } from 'react';
2
3   const empty = {…
17  };
18
19  export default function PostForm({edited = empty, onSave}) {
20    const [post, setPost] = useState(edited);
21    const firstFocusEl = useRef(null);
22
23    const handleSubmit = (ev) => {…
36    };
37
38    const handleChange = (ev) => {…
53    };
54
55    return (
56      <form onSubmit={handleSubmit}>
57        <textarea
58          ref={firstFocusEl}…
62          onChange={handleChange}/>
63        <input
64          name="tags"…
67          onChange={handleChange}
68        />
69        <button>Ok</button>
70      </form>
71    )
72  };
```

React

Здесь нам потребуются некоторые знания JS. В Wall, если мы ничего не устанавливаем в качестве начального значения useState, значит значение будет undefined.

```
const [edited, setEdited] = useState();
```

В props PostForm, если значение edited undefined, то будет использоваться значение по умолчанию, которое выставлено как empty:

```
export default function PostForm({edited = empty, onSave}) {
```

Таким образом, мы реализуем логику:

- если в edited в Wall undefined, то используем empty (добавление)
- если в edited в Wall какой-то пост, то используем его (редактирование)

Конечно же, логику с empty можно сразу разместить в Wall, но это лишь один из вариантов.

React

Несмотря на то, что можно увидеть в React Dev Tools как props меняется, отрисовываться в форме редактируемый пост не будет. Почему?

Дело в том, что useState лишь один раз устанавливает начальное значение. Изменение props (в нашем случае) не ведёт к установке начального значения.

Здесь используется немного другая механика: вместо того, чтобы пытаться установить начальное значение, мы можем на изменение props устанавливать новое значение state.

useEffect

Хук useEffect используется для того, чтобы реализовывать различные побочные действия (в том числе установку state) в ответ на изменение зависимостей. Что подразумевается под зависимостью? Давайте смотреть:

```
export default function PostForm({edited = empty, onSave}) {
  const [post, setPost] = useState(edited);
  const firstFocusEl = useRef(null);
  useEffect(() => {
    setPost(edited);
  }, [edited]);
```

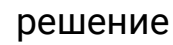
Каждый раз, когда edited будет меняться, мы будем менять state.

useEffect

А теперь попробуйте отредактировать самый первый пост, у которого content = null:

```
▶Warning: `value` prop on `textarea` should not be null. Consider using an empty string to clear the
component or `undefined` for uncontrolled components.
    in textarea (at PostForm.js:60)
    in form (at PostForm.js:59)
    in PostForm (at Wall.js:85)
    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)
```

```
return (
  <form onSubmit={handleSubmit}>
    <textarea
      ref={firstFocusEl}
      name="content"
      placeholder="content"
      value={post.content || ''}
      onChange={handleChange}/>
    <input
      name="tags"
      placeholder="tags"
      value={post.tags?.join(' ') || ''}
      onChange={handleChange}
    />
    <button>Ok</button>
  </form>
)
```



решение

"Чистота данных"

Почему мы так много внимания обращаем на "неудобные" данные? Ведь можно было сразу договориться в content хранить не null, а пустую строку. В tags - пустой массив, а в photo - объект с пустыми значениями alt и url.

Всё зависит от того, кто разрабатывает серверную часть вашего приложения. Если вы сами или тот, с кем вы можете согласовать удобный для себя формат данных - тогда нет проблем, вы используете его (и большая часть обсуждаемых нами вопросов снимается).

Но не всегда это возможно, именно поэтому мы вам показываем эти проблемы, чтобы вы знали, как их решать.

Редактирование

Ок, редактирование работает, но достаточно странно. Если отредактировать пост, то он вместо обновления существующего, добавит новый.

Один из вариантов решения этой проблемы: прямо в Wall проверять, это обновление или создание нового:

```
const handlePostSave = (post) => {
  if (edited !== undefined) {
    setPosts((prevState) => prevState.map((o) => {
      if (o.id !== post.id) {
        return o;
      }

      return {...post};
    }))
    setEdited(undefined);
    return;
  }
  setPosts((prevState) => [{...post}, ...prevState]);
};
```

Редактирование

Но так тоже не работает. Почему? Вы можете посмотреть в отладчике (поставить инструкцию debugger) и увидеть, что поста с таким id просто нет, даже если мы редактируем. А почему? Потому, что в PostForm мы каждый раз при сохранении меняем id:

```
onSave({
  ...post,
  id: Date.now(),
  created: Date.now(),
  tags,
  photo: post.photo?.url ? {alt: '', ...post.photo} : null
});
setPost(empty);
firstFocusEl.current.focus();
```

Редактирование

Соответственно, мы можем просто поменять логику, чтобы id не назначался, если он уже назначен (помните, мы в empty выставили 0? мы это сделали не случайно):

```
onSave({
  ...post,
  id: post.id || Date.now(),
  created: post.created || Date.now(),
  tags,
  photo: post.photo?.url ? {alt: '', ...post.photo} : null
});
setPost(empty);
firstFocusEl.current.focus();
```

Теперь всё работает, как нужно.

useEffect

Единственный вопрос, а нужно ли нам теперь в handleSubmit делать setPost(empty)? Ведь мы после редактирования делаем setEdited(undefined) в Wall?

На самом деле, нужно, потому что иначе после добавления не будет вычищаться форма (именно после добавления), либо можно вызывать setEdited(undefined) в Wall после добавления:

```
const handlePostSave = (post) => {
  if (edited !== undefined) {
    setPosts((prevState) => prevState.map((o) => {
      if (o.id !== post.id) {
        return o;
      }

      return {...post};
    }))
    setEdited(undefined);
    return;
  }
  setPosts((prevState) => [{...post}, ...prevState]);
  setEdited(undefined);
};
```

Важно

Мы специально провели вас полностью через этап создания приложения и показали, как на самом деле происходит работа, когда вы делаете что-то впервые (никто не делает сразу идеально, если до этого уже раз 5 не делал того же самого).

Поэтому будьте готовы редактировать свой код, проверять его на различные случаи и т.д.

ИТОГИ

Итоги

Сегодня мы рассмотрели вопросы редактирования. Фактически, мы завершили создание CRUD-приложения (базовый навык) с достаточно сложными объектами.

У той модели, что мы использовали, есть ряд недостатков: компоненты достаточно тесно связаны друг с другом в плане того, что нам приходится при внесении изменений постоянно что-то прокидывать в props, менять в state одного компонента, другого и т.д.

Для небольших приложений вроде нашего - это вполне нормально. Но для более сложных - нет. Поэтому на следующем занятии мы рассмотрим Context API и Redux, которые позволяют решить эту проблему.

ДОМАШНЕЕ ЗАДАНИЕ

ДЗ: Отмена

Несмотря на то, что кажется, что наше приложение работает, на самом деле оно не работает. Почему? Просто потому, что в нём нельзя отменить редактирование. Т.е. если мы нажали на каком-то посту "изменить", мы не можем "отменить" это (перезагрузка страницы не считается).

Что вам нужно сделать? Сделайте кнопку "Отменить" на форме, которая:

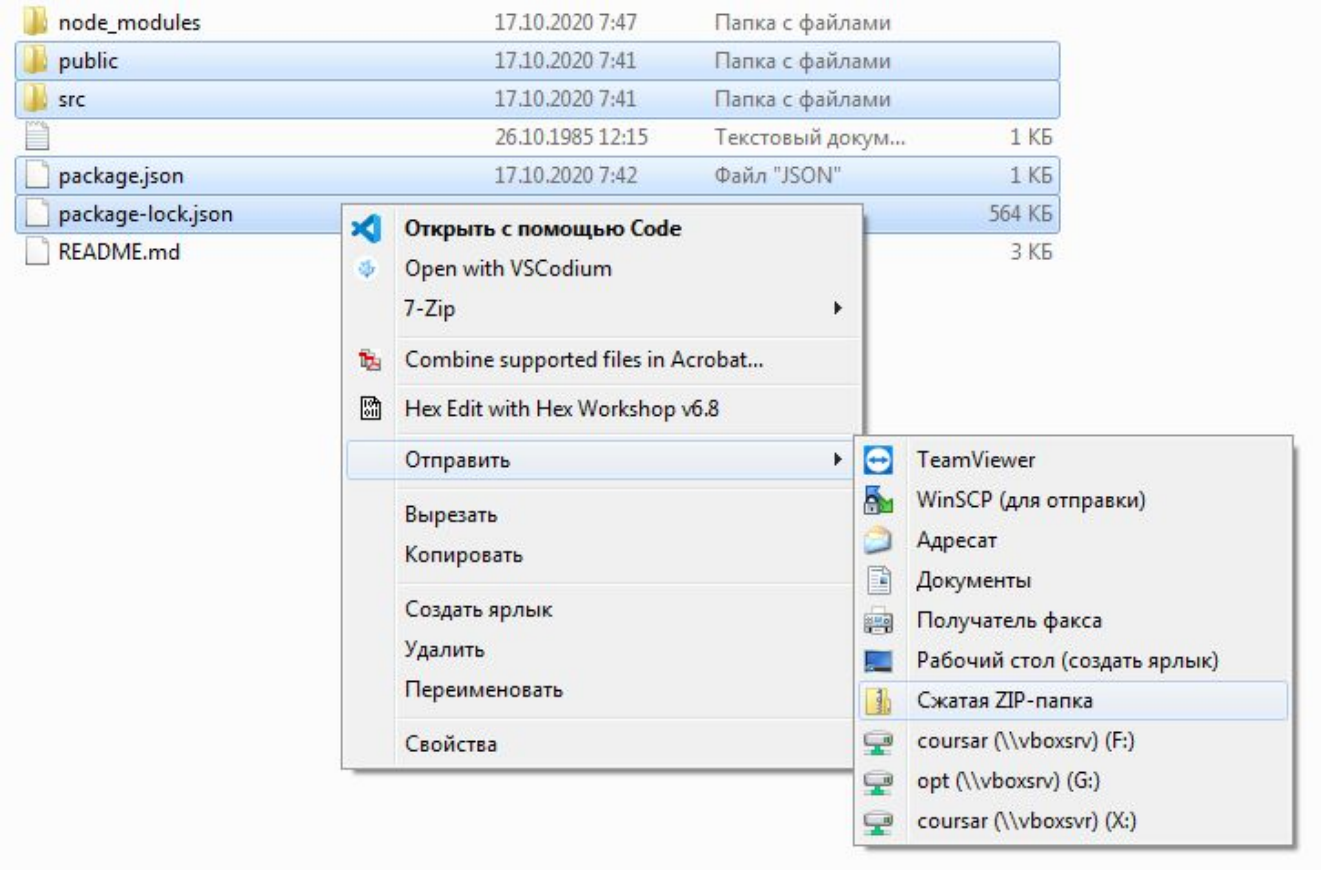
1. Появляется только тогда, когда пользователь редактирует существующий пост, а не создаёт новый
2. Отменяет редактирование (т.е. выставляет state в empty)
3. И не забудьте, что ещё стоит как-то Wall сообщить о том, что пользователь нажал на отмену (сделайте с помощью props onCancel)
4. **Важно**: устанавливайте обработчик именно на кнопку (onClick), а не на форму (onReset)

ДЗ: Отмена

В коде лекции показано решение с фото (хотя самих полей не показано). Это значит, что если вы не сделали на прошлой лекции задачу с фото, то самое время её сделать (это всегда так - недоделанное рано или поздно приходится доделывать).

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.

open (js)
school