Объекты в React

При любых изменениях объектов, эти изменения сразу же будут отображаться на экране.
Также, как и при работе с массивами, изменения объектов следует производить иммутабельно, то есть не изменяя исходного объекта.
Например: Пусть в стейте obj хранится объект:

```
function App() {
  const [obj, setObj] = useState({
    prop1: 'value1',
    prop2: 'value2',
    prop3: 'value3',
  });

  return <div>
    <span>{obj.prop1}</span>,
    <span>{obj.prop2}</span>,
    <span>{obj.prop3}</span>
  </div>;
}
```

Изменим значение какого-либо из свойств нашего объекта.
Для этого можно выполнить изменения в копии объекта:

```
const copy  = Object.assign({}, obj);
copy.prop1 = '!';
setObj(copy);
```

Или воспользоваться деструктуризацией

```
setObj({...obj, ...{prop1: '!'}});
```

Выведем каждое свойство объекта в отдельном инпуте

```
function App() {
  const [obj, setObj] = useState(initObj);

  return <div>
    <input value={obj.prop1} />
    <input value={obj.prop2} />
    <input value={obj.prop3} />

    <br />
    {obj.prop1}-{obj.prop2}-{obj.prop3}
  </div>;
}
```

К каждому инпуту привяжем событие onChange. В качестве обработчика назначим одну общую функцию

```
function App() {
  const [obj, setObj] = useState(initObj);

  return <div>
    <input value={obj.prop1} onChange={event => handleChange('prop1', event)} />
    <input value={obj.prop2} onChange={event => handleChange('prop2', event)} />
    <input value={obj.prop3} onChange={event => handleChange('prop3', event)} />

    <br />
    {obj.prop1}-{obj.prop2}-{obj.prop3}
  </div>;
}
```

```
function handleChange(prop, event) {
  const copy = Object.assign({}, date);
  copy[prop] = event.target.value;
  setObj(copy);
}
```

или

```
function handleChange(prop, event) {
  setObj({...obj, ...{[prop]: event.target.value}});
}
```

Рассмотрим реактивность массива объектов. Как правило, вы будете работать с такой структурой достаточно часто. Поэтому вам необходимо знать, как иммутабельно осуществлять добавление, изменение и удаление элементов такого массива.
Проблема здесь в том, что все изменения следует проводить по id, которые хранятся внутри самих объектов. Из-за этого нельзя просто взять и получить элемент по его id как по ключу массива.
Придется перебирать массив циклом и в цикле проверять каждый из объектов на то, равен ли его id тому, который нам нужен. Если равен, то выполним с ним нужную нам операцию, а если не равен - то оставим элемент без изменения.
Например: Пусть имеется следующий массив объектов:

```
const initNotes = [
  {
    id: 'GYi9G_uC4gBF1e2SixDvu',
    prop1: 'value11',
    prop2: 'value12',
    prop3: 'value13',
  },
  {
    id: 'IWSpfBPSV3SXgRF87uO74',
    prop1: 'value21',
    prop2: 'value22',
    prop3: 'value23',
  },
  {
    id: 'JAmjRlfQT8rLTm5tG2m1L',
    prop1: 'value31',
    prop2: 'value32',
    prop3: 'value33',
  },
];
```

выведем каждый элемент массива в отдельном абзаце, а значения свойств каждого объекта - в своем span внутри абзаца

```
function App() {
  const [notes, setNotes] = useState(initNotes);

  const result = notes.map(note => {
    return <p key={note.id}>
      <span>{note.prop1}</span>,
      <span>{note.prop2}</span>,
      <span>{note.prop3}</span>
    </p>;
  });

  return <div>
    {result}
  </div>;
}
```

Удаление

Пусть в переменной хранится id элемента массива:

```
const id = 'IWSpfBPSV3SXgRF87uO74';
```

Удалим элемент с таким id. Используем для этого метод filter:

```
setNotes(notes.filter(note => {
  if (note.id !== id) {
    return note;
  }
}));
```

Код можно упростить

```
setNotes(notes.filter(note => note.id !== id));
```

Добавление

Пусть в переменной хранится объект, который мы хотим сделать новым элементом нашего массива. Для этого можно добавить элемент в копию массива

```
const newElem = {
  id: 'GMNCZnFT4rbBP6cirA0Ha',
  prop1: 'value41',
  prop2: 'value42',
  prop3: 'value43',
};
```

```
const copy = Object.assign([], notes);
copy.push(newElem);
setNotes(copy);
```

Или воспользоваться деструтуризацией

```
setNotes([...notes, newElem]);
```

Изменение

Изменим какой-нибудь элемент массива. Пусть новые данные хранятся в переменной

```
const data = {
  id: 'IWSpfBPSV3SXgRF87uO74',
  prop1: 'value21 !',
  prop2: 'value22 !',
  prop3: 'value23 !',
};
```

В приведенном объекте id совпадает с id второго элемента массива, а значения свойств - другие. Говоря другими словами в data в свойстве id у нас хранится id того элемента массива, который мы хотим изменить.

Давайте выполним это изменение. Для этого будем перебирать элементы массива циклом и, если id совпадает с искомым, выполним замену элемента, а если не совпадает, оставим элемент без изменений:

```
setNotes(notes.map(note => {
  if (note.id === data.id) {
    return data;
  } else {
    return note;
  }
}));
```

Можно сократить код, воспользовавшись тернарным оператором

```
setNotes(notes.map(note => note.id === data.id ? data : note));
```

Может потребоваться изменять не весь объект, а конкретное свойство.
Пусть в переменных хранятся id элемента, имя свойства для изменения и новое значение свойства:

```
const id = 'IWSpfBPSV3SXgRF87uO74';
const prop = 'prop1';
const value = '!!!';
```

Для решения задачи удобно использовать деструктуризацию и вычисляемые имена свойств:

```
setNotes(notes.map(note => {
  if (note.id === id) {
    return {...note, [prop]: value};
  } else {
    return note;
  }
}));
```

может потребоваться получить элемент массива по его id.

```
const id = 'IWSpfBPSV3SXgRF87uO74';
```

получим элемент с таким id. Используем для этого метод reduce:

```
const result = notes.reduce((res, note) => {
  if (note.id === id) {
    return note;
  } else {
    return res;
  }
}, {});
```

```
const result = notes.reduce((res, note) => note.id === id ? note: res, {});
```

Иногда может потребоваться получить элемент по id, а затем извлечь из этого элемента значение определенного свойства.
Пусть id элемента и необходимое свойство хранятся в переменных:

```
const id = 'IWSpfBPSV3SXgRF87uO74';
const prop = 'prop1';
```

Для решения задачи нужно просто модифицировать полученный ранее код с reduce:

```
const result = notes.reduce((res, note) => {
  if (note.id === id) {
    return note[prop]; // получаем заданное свойство
  } else {
    return res;
  }
}, '');
```

```
const result = notes.reduce((res, note) => note.id === id ? note[prop] : res, '');
```

Удаления из массива объектов

```
function App() {
  const [notes, setNotes] = useState(initNotes);

  function remItem(id) {
    setNotes(notes.filter(note => note.id !== id));
  }

  const result = notes.map(note => {
    return <p key={note.id}>
      <span>{note.prop1}</span>,
      <span>{note.prop2}</span>,
      <span>{note.prop3}</span>
      <button onClick={() => remItem(note.id)}>remove</button>
    </p>;
  });

  return <div>
    {result}
  </div>;
}
```

Форма для добавления в массив объектов

Пусть у нас есть массив объектов initNotes, элементы которого выводятся в виде абзацев:

```
function App() {
  const [notes, setNotes] = useState(initNotes);

  const result = notes.map(note => {
    return <p key={note.id}>
      <span>{note.prop1}</span>,
      <span>{note.prop2}</span>,
      <span>{note.prop3}</span>
    </p>;
  });

  return <div>
    {result}
  </div>;
}
```

Сделаем инпуты для добавления новых элементов в наш массив.
Для начала сделаем три инпута и кнопку, по нажатию на которую будет происходить добавление:

```
return <div>
  {result}

  <br />

  <input />
  <input />
  <input />
  <button>save</button>
</div>;
```

Каждому инпуту сделаем свой отдельный стейт

```
const [value1, setValue1] = useState('');
const [value2, setValue2] = useState('');
const [value3, setValue3] = useState('');
```

Добавим обработчики события onChange:

```
return <div>
  {result}

  <br />

  <input value={value1} onChange={event => setValue1(event.target.value)} />
  <input value={value2} onChange={event => setValue2(event.target.value)} />
  <input value={value3} onChange={event => setValue3(event.target.value)} />

  <button onClick={addItem}>save</button>
</div>;
```

по нажатию на кнопку вызывается функция addItem

```
function addItem() {
  let obj = {
    prop1: value1,
    prop2: value2,
    prop3: value3,
  };

  setNotes([...notes, obj]);
}
```

Данная функция берет значение каждого инпута из соответствующего стейта, делает из этих значений объект с новым элементом, а затем добавляет этот элемент в массив

Кроме того, добавляемый элемент должен иметь уникальный id. Этот id можно генерировать с помощью функции id():

```
function id() {
  // тут генерация id
}
```

Есть готовые библиотеки для генерации. Например, библиотека nanoid, генерирующая случайные строки, либо библиотека react-uuid, генерирующая UUID.

Изменим код

```
function addItem() {
  let obj = {
    id: id(),
    prop1: value1,
    prop2: value2,
    prop3: value3,
  };

  setNotes([...notes, obj]);
}
```

```
function App() {
  const [notes, setNotes] = useState(initNotes);

  const [value1, setValue1] = useState('');
  const [value2, setValue2] = useState('');
  const [value3, setValue3] = useState('');

  const result = notes.map(note => {
    return <p key={note.id}>
      <span>{note.prop1}</span>,
      <span>{note.prop2}</span>,
      <span>{note.prop3}</span>
    </p>;
  });

  function addItem() {
    let obj = {
      id: id(),
      prop1: value1,
      prop2: value2,
      prop3: value3,
    };

    setNotes([...notes, obj]);
  }

  return <div>
    {result}

    <br />

    <input value={value1} onChange={event => setValue1(event.target.value)} />
    <input value={value2} onChange={event => setValue2(event.target.value)} />
    <input value={value3} onChange={event => setValue3(event.target.value)} />

    <button onClick={addItem}>save</button>
  </div>;
}
```

Форма для редактирования массива объектов

Пусть есть массив объектов initNotes, элементы которого выводятся в виде абзацев. Под абзацами сделаем инпуты для редактирования данных наших абзацев. Пусть в конце каждого абзаца будет кнопка для редактирования.
По нажатию на кнопку данные абзаца должны попасть в инпуты. При редактировании инпутов реактивно будет изменяться текст абзаце.

Для начала сделаем стейт editId, хранящий в себе id элемента, который редактируется в настоящий момент. Если ничего не редактируется (например, по умолчанию), пусть этот стейт имеет значение null

```
const [editId, setEditId] = useState(null);
```

Добавим в конец абзаца кнопку, которая будет записывать id абзаца в editId:

```
const result = notes.map(note => {
  return <p key={note.id}>
    <span>{note.prop1}</span>,
    <span>{note.prop2}</span>,
    <span>{note.prop3}</span>

    <button onClick={() => setEditId(note.id)}>edit</button>
  </p>;
});
```

Сделаем так, чтобы в инпутах выводился текст редактируемого абзаца.
Для этого нужно из массива получить редактируемый объект по его id и в каждый инпут записать соответствующее свойство этого объекта.
Пусть это значение извлекает специальная функция getValue:

```
<input value={getValue('prop1')} />
<input value={getValue('prop2')} />
<input value={getValue('prop3')} />
```

```
function getValue(prop) {
  return notes.reduce((res, note) => {
    if (note.id === editId) {
      return note[prop];
    } else {
      return res;
    },
  }, '');
}
```

```
function getValue(prop) {
  return notes.reduce((res, note) => note.id === editId ? note[prop] : res, '');
}
```

Сделаем так, чтобы при изменении любого инпута изменялось значение соответствующего свойства соответствующего элемента массива.
Для этого каждому инпуту в качестве обработчика события onChange привяжем функцию:

```
<input
  value={getValue('prop1')}
  onChange={event => changeItem('prop1', event)}
/>
<input
  value={getValue('prop2')}
  onChange={event => changeItem('prop2', event)}
/>
<input
  value={getValue('prop3')}
  onChange={event => changeItem('prop3', event)}
/>
```

```
function changeItem(prop, event) {
  setNotes(notes.map(note => {
    if (note.id === editId) {
      return {...note, [prop]: event.target.value};
    } else {
      return note;
    }
  }));
}
```

```
function changeItem(prop, event) {
  setNotes(notes.map(note =>
    note.id === editId ? {...note, [prop]: event.target.value} : note
  ));
}
```

После инпутов добавим кнопку, нажатие на которую будет завершать редактирование. Т.е. просто очистку инпутов. Для этого нужно установить стейт editId в null:

```
<button onClick={() => setEditId(null)}>save</button>
```

Полное решение задачи

```
function App() {
  const [notes, setNotes] = useState(initNotes);
  const [editId, setEditId] = useState(null);

  const result = notes.map(note => {
    return <p key={note.id}>
      <span>{note.prop1}</span>,
      <span>{note.prop2}</span>,
      <span>{note.prop3}</span>

      <button onClick={() => setEditId(note.id)}>edit</button>
    </p>;
  });

  function getValue(prop) {
    return notes.reduce((res, note) => note.id === editId ? note[prop] : res, '');
  }

  function changeItem(prop, event) {
    setNotes(notes.map(note =>
      note.id === editId ? {...note, [prop]: event.target.value} : note
    ));
  }

  return <div>
    {result}
    <br />

    <input
      value={getValue('prop1')}
      onChange={event => changeItem('prop1', event)}
    />
    <input
      value={getValue('prop2')}
      onChange={event => changeItem('prop2', event)}
    />
    <input
      value={getValue('prop3')}
      onChange={event => changeItem('prop3', event)}
    />

    <button onClick={() => setEditId(null)}>save</button>
  </div>;
}
```

Универсальная форма для массива объектов

```
function App() {
  const [notes, setNotes] = useState(initNotes);
  const [obj, setObj] = useState(getInitObj());
  const [editId, setEditId] = useState(null);

  const result = notes.map(note => {
    return <p key={note.id}>
      <span>{note.prop1}</span>,
      <span>{note.prop2}</span>,
      <span>{note.prop3}</span>

      <button onClick={() => setEditId(note.id)}>edit</button>
    </p>;
  });

  function getValue(prop) {
    if (editId) {
      return notes.reduce((res, note) => note.id === editId ? note[prop] : res, '');
    } else {
      return obj[prop];
    }
  }

  function changeItem(prop, event) {
    if (editId) {
      setNotes(notes.map(note =>
        note.id === editId ? {...note, [prop]: event.target.value} : note
      ));
    } else {
      setObj({...obj, [prop]: event.target.value});
    }
  }
```

```
  function saveItem() {
    if (editId) {
      setEditId(null);
    } else {
      setNotes([...notes, obj]);
      setObj(getInitObj());
    }
  }

  return <div>
    {result}

    <br />

    <input
      value={getValue('prop1')}
      onChange={event => changeItem('prop1', event)}
    />
    <input
      value={getValue('prop2')}
      onChange={event => changeItem('prop2', event)}
    />
    <input
      value={getValue('prop3')}
      onChange={event => changeItem('prop3', event)}
    />

    <button onClick={saveItem}>save</button>
  </div>;
}

function getInitObj() {
  return {
    id: id(),
    prop1: '',
    prop2: '',
    prop3: ''
  }
}
```

Практические задания

1. Сделайте 3 кнопки. Пусть первая кнопка изменяет значение свойства prop1, вторая - prop2, а третья - prop3.
2. Возьмите массив с продуктами `initProds` и выведите его в виде HTML таблицы.

```
const initProds = [
  {id: id(), name: 'prod1', catg: 'catg1', cost: 100},
  {id: id(), name: 'prod2', catg: 'catg2', cost: 200},
  {id: id(), name: 'prod3', catg: 'catg3', cost: 300},
];
```

3. Возьмите таблицу с продуктами initProds. В конце каждого ряда сделайте ячейку, в которой будет кнопка для удаления продукта.
4. Сделайте под таблицей инпуты для добавления нового продукта.
5. Сделайте под таблицей инпуты для добавления нового продукта.
6. Сделайте под таблицей форму для редактирования продукта. Добавьте в таблицу еще одну колонку, в которой будут кнопки для редактирования продуктов.
7. Сделайте под таблицей универсальную форму для добавления и редактирования продукта.