Лабораторная работа: Состояние в компонентах React

В этой лабораторной мы попрактикуемся создавать компоненты с состоянием. **Примечания:**

- мы не будем использовать файлы из работы с корзиной
- компоненты, которые будут разработаны не будут использоваться в дальнейших лабораторных
- основная цель - сосредоточиться на механизме работы с состояниями

Упражнение 1: Создание функционального компонента с состоянием

1.1. Создайте компонент **src/components/Clicker.js**

- это будет компонент с состоянием
- `Clicker` будет показывать количество нажатий по нему

```
import React, {useState} from 'react'

const Clicker = ({value}) => {
  return (
    <button
      className='btn'
      onClick={()=>{alert('Click!')}}
    >
        Clicker {value}
    </button>
  )
}

export default Clicker
```

Примечания:

- `import {useState} from 'react'` импорт функции-хука из библиотеки `react`
- можно было не импортировать, а использовать как `React.useState`
- `useState()` - принимает начально значение состояния (произвольный JS-тип)
- `useState()` - возвращает массив из двух элементов
  - первый - переменная/константа с текущим состоянием
  - второй - функция, которая устанавливает переменную-состояние
- `const [str, setStr] = useState('Привет')` - тут
  - `str` содержит 'Привет'
  - `setStr('мир!')` устанавливает значение `str` в 'мир!'

1.2. Откройте файл **App.js** и опишите импорт компонента

```
import Clicker from './components/Clicker'
```

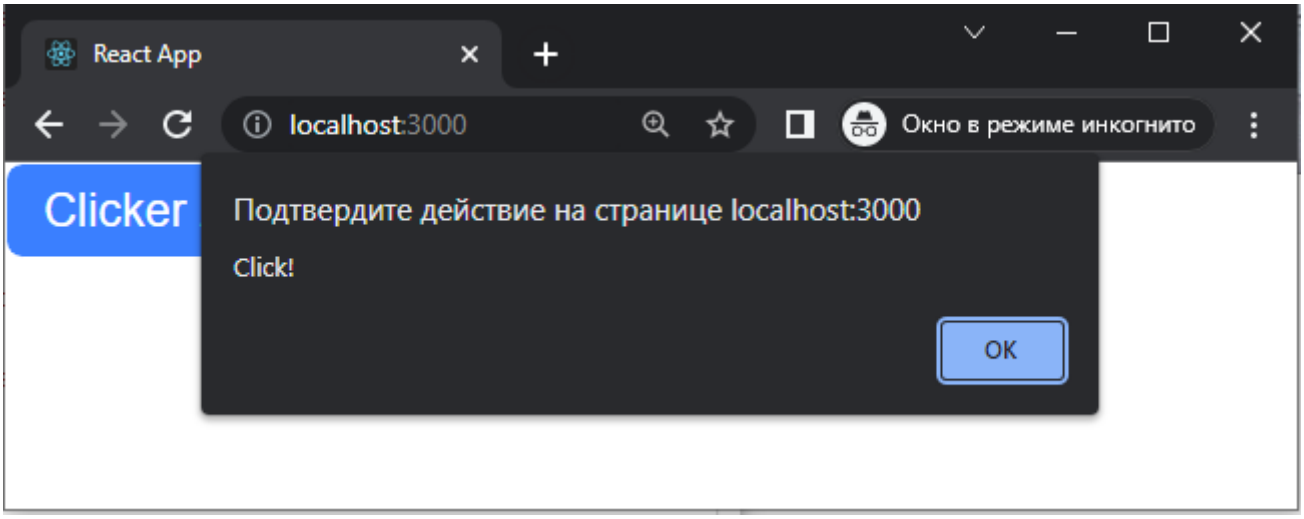
1.3. Измените его так, чтобы в `return` попал новый компонент `<Clicker value={2}/>`

```
return (
   <div className="App">
     <header className="App-header">
       {/* <h1>Корзина</h1> */}

       <Clicker value={2}/>

       {/* <Basket items={items} /> */}
     </header>
   </div>
 );
```

1.4. Посмотрите результат в браузере. При нажатии на кнопку, должно появляться модальное окно с текстом *Click!*



1.5. Измените функциональный компонент `Clicker` так, чтобы при нажатии число на кнопке изменялось на единицу в большую строну

```
import React, {useState} from 'react'

const Clicker = ({value}) => {

  const [num, setNum] = useState( value )

  return (
    <button
      className='btn'
      onClick={()=>{setNum(num + 1)}}
    >
        Clicker {num}
    </button>
  )
```

```
}

export default Clicker
```

Примечания:

- `useState( value )` - в качестве начального значения для состояния, берём значение из свойств/props - `value`
- `useState( value )` - возвращает массив из двух элементов
- `const [num, setNum] = useState( value )` - тут деструктуризация возвращаемого массива
  - `num` содержит '2' (при первом отображении)
  - `setNum(num + 1)` устанавливает значение `num` в значение 2 (при первом отображении)

1.6. Сделайте паузу, внимательно изучите полученный код. Поэкспериментируйте, создайте свои величины для хранения состояния и убедитесь, что вы полностью понимаете как хранит состояние функциональный компонент

1.7. Добавьте в `Clicker` состояние, которое при каждом нажатии будет менять оформление кнопки (например). Состояние должно быть логической величиной, а оформление кнопки менять за счет указания CSS-класса

```
import React, { useState } from "react";

const Clicker = ({ value }) => {
  const [num, setNum] = useState(value);
  const [isDark, setIsDark] = useState(false);

  const clickHandler = (ev) => {
    setNum(num + 1);
    setIsDark(!isDark);
  };

  return (
    <button
      className={"btn" + (isDark ? " btn-dark" : "")}
      onClick={clickHandler}
    >
      Clicker {num}
    </button>
  );
};

export default Clicker;
```

1.8. Убедитесь, что при каждом нажатии кнопки меняется не только счётчик нажатий, но и цвет фона кнопки

1.9. Обязательно познакомьтесь с правилами использования хуков

Упражнение 2: Создание классового компонента с состоянием

2.1. Закомментируйте `Ctrl + /` функциональное описание компонента `Clicker` в файле **Clicker.js** (но оставьте код с импортом и экспортом)

2.2. Опишите класс `Clicker` расширяющий `React.Component`.

```
class Clicker extends React.Component {

}
```

Примечание: классовые компоненты всегда должны расширять базовый класс `React.Component`

2.3. Опишите метод `constructor` в классе `Clicker`. Вызовите для реализации наследования в конструкторе родительский конструктор `super(props)`. Создайте начальное состояние компонента `this.state = {...}`. Метод-обработчик по нажатию на кнопку `clickHandler` привяжите к текущему `this`.

```
  constructor(props) {
    super(props);

    this.state = {
        num: props.value,
        isDark: false
    }

    this.clickHandler = this.clickHandler.bind(this);
  }
```

Примечания:

- если указывается конструктор, обязательно использовать `super()`
- начальное состояние задаётся в виде объекта
- метод `clickHandler` можно описать внутри `constructor`
- если метод `clickHandler` описан вне конструктора, то нужно выполнить связывание через `this.clickHandler.bind(this)`
- в начальное состояние может попасть значение из свойств/props

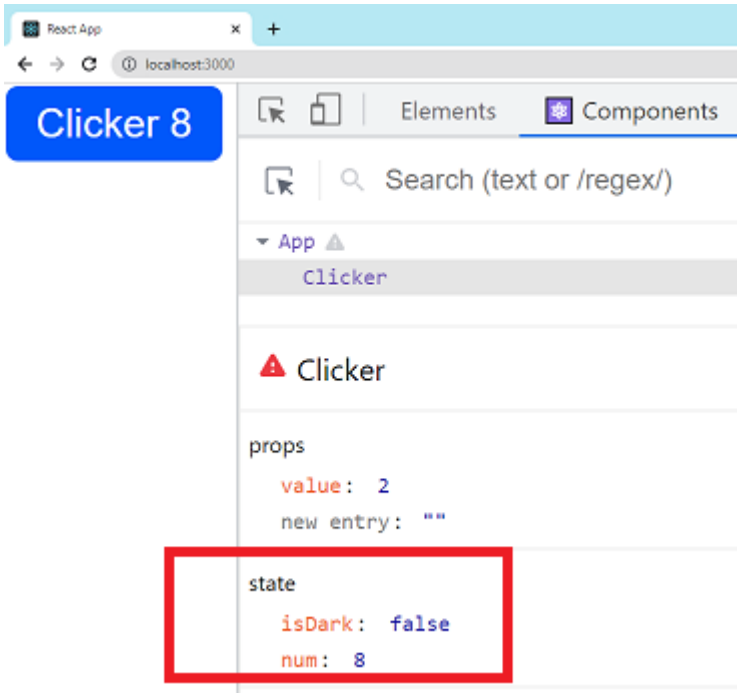
2.4. Опишите метод `render` в классе `Clicker`. `render` должен возвращать кнопку `<button></button>`, содержать ссылку в `onClick` на функцию обработчик нажатия `clickHandler` и выводить фрагменты состояния

```
  render() {
    return (
      <button
        className={"btn" + (this.state.isDark ? " btn-dark" : "")}
        onClick={this.clickHandler}
```

```
        >
          Clicker {this.state.num}
        </button>
      );
    }
```

Примечания:

- `this.state.isDark` - обращаемся к булеву свойству состояния
- `this.state.num` - обращаемся к числовому свойству состояния
- `className={...}` - содержание свойства формируем на ходу
- тут можно было бы использовать classnames
- `onClick={this.clickHandler}` - указываем ссылку. Обратите внимание, `this.clickHandler` тут без скобок. Со скобками было бы можно, если осознанно использовать замыкания
- Состояние компонента можно посмотреть в Devtools



2.5. Опишите метод `clickHandler` в классе `Clicker`. Метод должен при помощи `this.setState()` менять состояние компонента

```
clickHandler(ev){
    this.setState({
        num: this.state.num + 1,
        isDark: !this.state.isDark
    })
}
```

Примечания:

- `this.setState()` - метод, который принимает **новый** объект-состояние
- `.setState()` - может принять объект с частью состояния, например

```
this.setState({ num: this.state.num + 1 })
```

при этом остальные части состояния затронуты не будут

- если хотите посмотреть новое состояние, не пытайтесь смотреть в консоли `this.state`, потому что к моменту вывода состояние еще может быть не обновлено
- в классовых компонентах состояние можно менять только через `this.setState()`

2.6. Создайте свой собственный компонент с состоянием. Например (в порядке возрастания сложности),

- `<Like />` - рисует фразу типа "Мне нравится 2" или "Like 3", показывая количество лайков
- `<Time />` - выводит текущее время
- `<Game />` - игра Крестики-Нолики

##Выводы

- мы создали компонент с состоянием
- состояние - позволяет хранить набор характеристик компонента и менять их
- при изменении состояния React-компонент перерисовывается
- хуки - методы/функции React для работы с состояниями
- функциональные компоненты для работы с состоянием используют хуки (типа `useState`)
- классовые компоненты изменяют состояние через метод this.setState
- функциональные компоненты проще и получили большее распространение