Создание компонентов в React

В этой лабораторной мы попрактикуемся создавать компоненты - строительные кирпичики при построении пользовательского интерфейса

Упражнение 0: Выбор рабочей папки

0. **Если вы не выполняли предыдущие лабораторные**, под правами администратора откройте в консоли папку `02mod-my-app`. Например, это может выглядеть так

```
Microsoft Windows [Version 10.0.17763.2686]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd C:\Users\Administrator\Desktop\react1\02mod-my-app\my-app
```

и выполните установку пакетов внутри приложения

```
npm install
```

Примечание: основная папка проекта в VSCode по-прежнему должна называться *my_app*

Упражнение 1: Создание функционального компонента

1.0. Если CRA не запущен, перейдите в папку **my_app** и запустите CRA командой `npm start` 1.1. В папке **src** создайте папку **components** с файлами **Button.js** и **Button.css**.

```
src
  ├── components
        ├── Button.js
        └── Button.css
  ├── App.css
  ├── App.js
  ├── App.test.js
  ├── index.css
  ├── index.js
  ├── logo.svg
  ├── serviceWorker.js
  └── setupTests.js
```

1.2. Задайте стили в **Button.css**.

```
.btn{
    padding: 5px 10px;
    background: rgb(0, 87, 250);
    border: 1px solid rgb(255, 255, 255);
    border-radius: 5px;
    color: rgb(255, 255, 255);
}
.btn:hover{
    background: rgb(58, 127, 255);
    cursor: pointer;
}
```

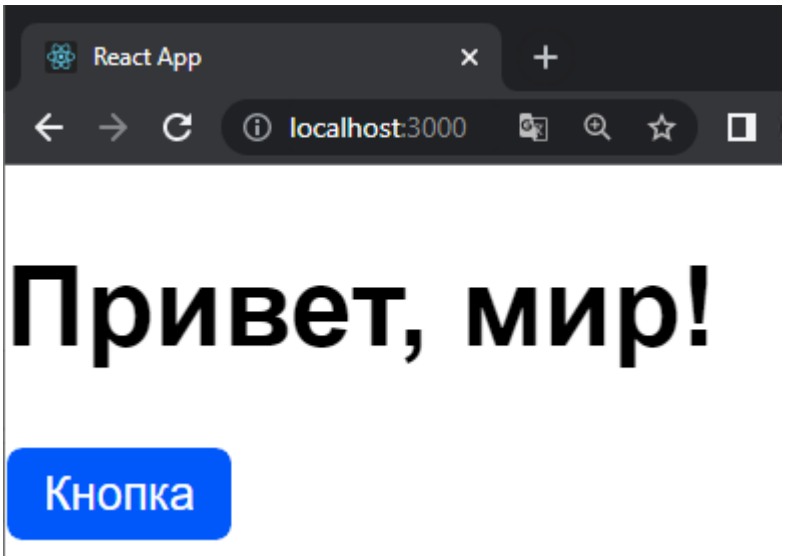
1.3. Опишите функциональный компонент в **Button.js**

```
import './Button.css';

const Button = function (){
    return <button className='btn'>Кнопка</button>
}

export default Button
```

1.4. Внесите изменения в **App.js**

```
import './App.css';
import Button from './components/Button.js'

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h1>Привет, мир!</h1>
        <div>
          <Button />
        </div>
      </header>
    </div>
  );
}

export default App;
```

1.5. Посмотрите результат в браузере на странице `http://localhost:3000/`, вид изображения будет приблизительно таким



1.6. Познакомьтесь с утверждениями и убедитесь, что все их них понятны

- обычно компоненты помещаются в папку **components**
- каждый компонент представляет собой отдельный JS-файл
- в файле компонента происходит импорт нужных стилей и других JS-файлов
- функциональный компонент представляет собой JavaScript-функцию, которая возвращает JSX или `null`
- для использования в **App.js** компонент нужно импортировать
- компонент можно описывать стрелочной функцией

```
const Button = () => <button className='btn'>Кнопка</button>
```

Упражнение 2: Свойства/пропсы функционального компонента

2.1. Выполним переработку кода (рефакторинг). Измените код кнопки в **App.js**

```
...
<Button value={'Моя кнопка!'} onClickHandler={()=>alert('Ура!')} />
...
```

- мы передаём в компонент **props** (свойства): `value` и `onClickHandler`
- значение свойства `value` задаётся как выражение, чтобы потом можно было менять
- значение свойства `onClickHandler` задаётся как функция
- переданные свойства `value` и `onClickHandler` попадают в функцию как свойства `props.value` и `props.onClickHandler`
- `props` - это первый аргумент любой функции-компонента
- если нужно передать набор свойств, можно использовать синтаксис `<Button {... obj } />`, где `obj`, свойства которого будут переданы в аргумент `props` функционального компонента

2.2. Внесите изменения в **Button.js**

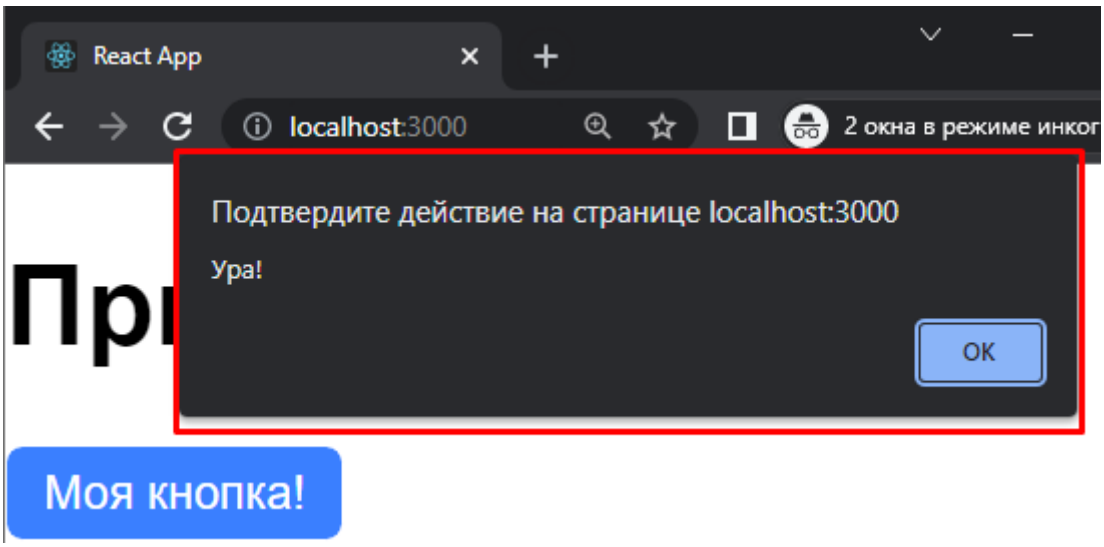
```
import './Button.css';

const Button = function (props){

    const {onClickHandler, value} = props;

    return <button className='btn' onClick={onClickHandler}>
        {value}
    </button>
}

export default Button
```

- первый аргумент функции - `props`
- `const {onClickHandler, value} = props` - извлечение значений, эквивалентно коду

```
const onClickHandler = props.onClickHandler;
const value = props.value;
```

- `onClick` - обработчик события `click`
- `onClick={onClickHandler}` - назначение обработчика нажатия

2.3. Нажмите на кнопку и убедитесь, что вызывается модальное окно



2.4. Создайте ещё пару кнопок внутри **App.js** с произвольными значениями в `value` и `onClickHandler` (тут должна передаваться произвольная функция)

2.5* Постройте набор кнопок, на основе массива объектов (разместите массив внутри компонента **App**)

```
const buttons = [
  {value: 'Кнопка1', fn: ()=> console.log(1)},
  {value: 'Кнопка2', fn: ()=> console.log(2)},
  {value: 'Кнопка3', fn: ()=> console.log(3)},
```

```
  {value: 'Кнопка4', fn: ()=> console.log(4)},
]
```

Примечание: используйте метод массивов `map`, чтобы вернуть массив JSX-выражений и поместите новый массив в `{}` внутри возвращаемого результата компонента `App`

2.6.* Создайте произвольный собственный функциональный компонент и используйте его в **App.js**

2.7. Выводы

- создали функциональный компонент React - **Button**
- передали в вызов компонента свойства/пропсы `<Button value={''} />`
- научили обращаться к переданным значениями через `props` функции: **props.value**
- подсмотрели документацию о props

Упражнение 3: Классовый компонент

Примечание: проще всего работать с компонентами как с функциями

3.1. Измените содержимое **Button.js**

```
import './Button.css';
import { Component } from 'react'

// const Button = function (props){

//     const {onClickHandler, value} = props;

//     return <button className='btn' onClick={onClickHandler}>
//         {value}
//     </button>
// }

class Button extends Component {
  render(){
    const {onClickHandler, value} = this.props;
    return <button className='btn' onClick={onClickHandler}>
         {value}
    </button>
  }
}

export default Button
```

- при создании компонента на классах ES6, мы должны расширять встроенный `React.Component`
- `render()` - метод класса, в жизненном цикле работы компонента отвечает за отрисовку компонента
- `render()` может возвращать JSX или `null` (если компонент не должен быть отрисован)

3.2. Убедитесь, что кнопки по прежнему работоспособны