Урок 5.2. React. Работа с props

`props`

props - это параметры компонента. Необходимы для того, чтобы можно было один и тот же компонент использовать с разнымии данными.

Использование `props` в функциональном компоненте

```
// MyComponent.js
// props - объект, передаётся как аргумент функции
function MyComponent(props) {
    return (
        <div className="MyComponent">
            {/* Используются свойства объекта props */}
            Hello, {props.name}!
        </div>
    );
}

export default MyComponent;
```

Использование `props` в классовом компоненте

```
// MyClassComponent.js
import React from 'react';

export default class MyClassComponent extends React.Component {
    render() {
        return (<div>
            {/* В классовом компоненте props доступны через this */}
           Hello Class, {this.props.name}!
        </div>);
    }
}
```

Прокидывание `props` при использовании компонента

```
import Component from './components/Component/Component';

function App() {
  const firstName = 'Boris';
  return (
      <div>
          {/* props прокидываются в компонент */}
          {/* как атрибуты тега */}
         <Component name={'Andrew'}/>

         {/* Можно использовать переменные */}
         <Component name={firstName}/>
      </div>  );
}

export default App;
```

`defaultProps`

defaultProps - это пропсы по умолчанию. Те значения, которые будут использоваться, если в компонент эти пропсы переданы не будут.

`defaultProps` в функциональном компоненте

```
// Здесь происходит деструктуризация,
// Объект в фигурных скобках - это props, и мы
// сразу достаём из него name, и если там его не оказалось
// записываем default значение
function MyComponent({name = 'Default value for name'})
 {
    return (
        <div className="MyComponent">
            {/* Далее name используется уже как самостоятельная переменная */}
            Hello, {name}!
        </div>
    );
}

export default MyComponent;
```

`defaultProps` в классовом компоненте

```
import React from 'react';

export default class MyClassComponent extends React.Component {
    // Создаём статический объект defaultProps
    static defaultProps = {
        name: 'Default Value for name'
    }

    render() {
        return (<div>
            Hello Class, {this.props.name}!
        </div>);
    }
}
```

Деструктуризация

Деструктуризация - это "вытягивание" свойств объекта или элементов массива.

```
const object = {
    name: 'Andrew',
    lastName: 'Gulin'
}
// Объявили две переменные, деструкурировав
// объект, теперь name - это object.name,
 // lastName - object.lastName
// Названия переменных должны совпадать с названием свойств
const { name, lastName } = object;


const array = [
    'First Element',
    'Last Element'
]
// Объявили две переменные
// Теперь first - это array[0],
// last - array[1].
// Здесь называть переменные можно как удобно
const [ first, last ] = array;
```

Деструктуризация свойств в классовом объекте

```
import React from 'react';
```

```
export default class MyClassComponent extends React.Component {
    render() {
        // name будет равно this.props.name
        // lastName будет равно this.props.lastName
        const { name, lastName } = this.props;
        return (<div>
            {/* Используем созданные переменные */}
            Hello Class, {name}!
        </div>);
    }
}
```

`PropTypes`

Для пропсов можно задать типы, для этого есть специальная библиотека от Реакта.

Установка библиотеки

1. Открыть терминал (в папке с проектом - WebStorm делает автоматически)
2. `nvm use` - для использования нужной версии NodeJS
3. `npm install prop-types`

Использование в функциональном компоненте

```
// Импортируем библиотеку
import PropTypes from 'prop-types';

function MyComponent(props) {
    return (
        <div className="MyComponent">
            Hello, {props.name}!
        </div>
    );
}

// Добавляем в функцию объект propTypes
MyComponent.propTypes = {
    // Используем для необходимого пропса
    // тип из PropTypes
    name: PropTypes.string
}

export default MyComponent;
```

Использование в классовом компоненте

```
import React from 'react';
// Импортируем библиотеку
import PropTypes from 'prop-types';

export default class MyClassComponent extends React.Component {
    // Создаём статический объект propTypes
    static propTypes = {
        name: PropTypes.string,
        lastName: PropTypes.string
    }
    render() {
        const { name, lastName } = this.props;
        return (<div>
            Hello Class, {name}!
        </div>);
    }
}
```

Отрисовка дочерних компонентов

Дочерний компонент - тот, который располагается в теге внутри родительского компонента:

```
<ParentComponent>
    <ChildComponent/>
    <ChildComponent/>
    <ChildComponent/>
</ParentComponent>
```

Чтобы все `<ChildComponent/>` отобразились, необходимо использовать `this.props.children` (или без `this` для функциональных компонентов)

```
import React from 'react';

export default class ParentComponent extends React.Component {
    render() {
        return (<div>
            {/* Здесь будет отрисовано всё, что находится внутри тега <ParentComponent>
 */}
            {this.props.children}
```

```
        </div>);
    }
}
```