Урок 3

Первое приложение на ReacJS

Задача написать и разобрать код react программы. Разбираем «Меню» блока

Что же такое ReactJS и как с ним работать

React – расширение javascript. Главное, что есть в react – компоненты.

Компоненты — это тело React. Обычно это части пользовательского интерфейса, которые содержат свою структуру и функциональность.

Элементы — это объекты JavaScript, которые представляют HTML-элементы. Их не существуют в браузере.они описывают DOM-элементы, такие как h1, div, или section. Входят в состав компонентов.

Virtual DOM — это дерево React элементов на JavaScript. React отрисовывает Virtual DOM в браузере, чтоб сделать интерфейс видимым. React следит за изменениями в Virtual DOM и автоматически изменяет DOM в браузере так, чтоб он соответствовал виртуальному.

Состояние — это специальный объект внутри компонента. Он хранит данные, которые могут изменяться с течением времени

Свойства - опции компонента. Они предоставляются в качестве аргументов компонента и выглядят так же, как атрибуты HTML.

Что такое JSX

JSX — это расширение синтаксиса JavaScript, разработанное командой ReactJS. Этот формат используется для упрощения написания компонентов ReactJS. Документацию по этому расширению можно найти на официальном сайте https://facebook.github.io/react/docs/jsx-in-depth.html. По своей сути это смесь JS и HTML, например:

```
const element = <h1>Hello, world!</h1>;
```

Создано для преобразования react в стандартный ECMAScript.

Преимущества:

1. Прост в написание.

2. Легко читается и упрощает обслуживание..

3. Запускается быстрее, чем такой же код на JS..

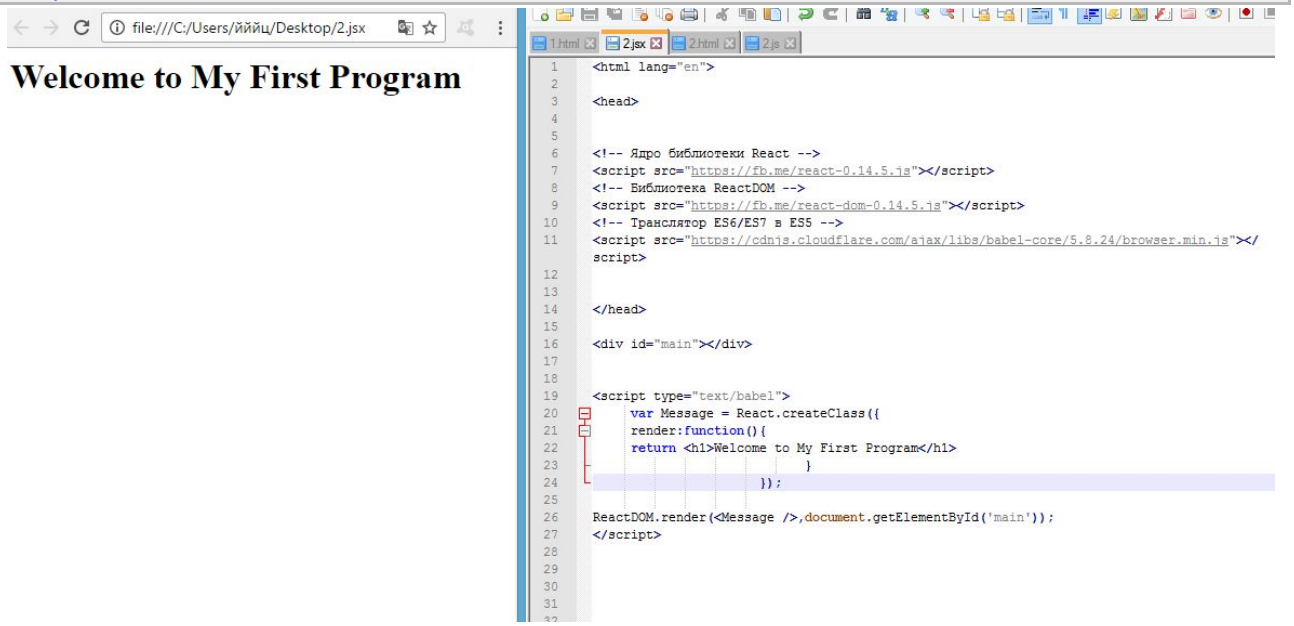
Для подключения для простоты можно воспользоваться шаблоном:

```
<head>
<scripttype="text/javascript"src="https://cdnjs.cloudflare.com/ajax/libs/react/15.3.2/react.js"></script>
<scripttype="text/javascript"src="https://cdnjs.cloudflare.com/ajax/libs/react/15.3.2/react-dom.js"></script>
<scriptsrc="https://unpkg.com/babel-core@5.8.38/browser.min.js"></script>
 <scripttype="text/babel">// JSX code would be here
</script>
</head>
```

где подключаются скрипты react и react-dom и скрипт babel, который будет переводить JSX код в вызовы react функций.

Пример JSX кода:

```
<scripttype="text/babel">
var Message = React.createClass({
render:function(){
return <h1>Welcome to My First Program</h1>
}
});
 ReactDOM.render(<Message/>,document.getElementById('main'));
</script>
```



Здесь появляется несколько интересных элементов, о которых речь будет идти далее:

```
varMessage

ReactDOM.render()
```

ReactComponent

В примере появляется компонент Message – компонент react, возвращающий JSX код выше, который затем при помощи Babel компилируется в функцию React.

Компоненты в react позволяют разделять интерфейс на отдельные части и анализировать каждую из них в отдельности.

Самый простой способ определить компонент это написать JavaScript функцию:

```
function Message(props) {

 return <h1>Hell</h1>;}
```

Другой вариант для определения компонента использование ES6 класса:

```
var Message = React.createClass{

 render() {

return<h1>Hello</h1>;}}
```

ReactDOM.render

ReactDOM.render является вложенным компонентом, используется для отрисовки элемента DOM.

Пусть существует элемент div –корневой узел DOM:

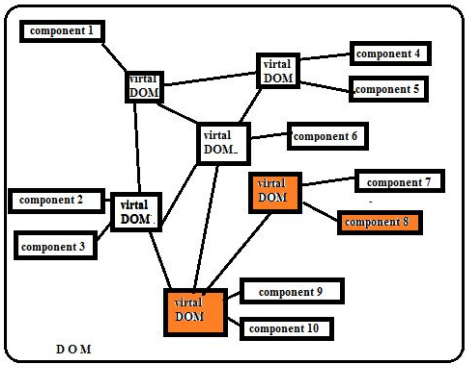
```
<divid="main"></div>
```

Таких узлов может быть бесконечное количество. Для отображения компонентов react на корневом узле используется команда ReactDOM.render. Пример синтаксиса:

```
const element = <h1>Hello, world</h1>;

ReactDOM.render(

 element,                              // название элемента

document.getElementById('main')  // вызов элементов корневом узле

);
```

Разделение на VirtalDom очень удобно и является не новым подходом. В каждом DOM создаются отдельные компоненты с отрисованными свойствами. В процессе отрисовки приложения необходимо выполнить изменения отдельного компонента. При этом отрисовка компонента выполняется по кратчайшему пути и не приводит к обновлению всего корневого документа. В результате получаем

увеличение производительности данного приложения. Является удобным способом отрисовки компонентов с большим количеством элементов.



Render function

Функция render определяет свойства компонентов. Может принимать два аргумента: виртуальный элемент и реального DOM узла. React принимает визуальный элемент и вставляет его в DOM узел..

Пример использования:

```
var Photo = React.createClass({
  render: function() {
return (
<divclassName='photo' />
<imgsrc= >
<span></span>
</div>
    )
  }
});
React.render(<PhotoimageURL='картинка 1'caption='New York!' />, document.body);
```

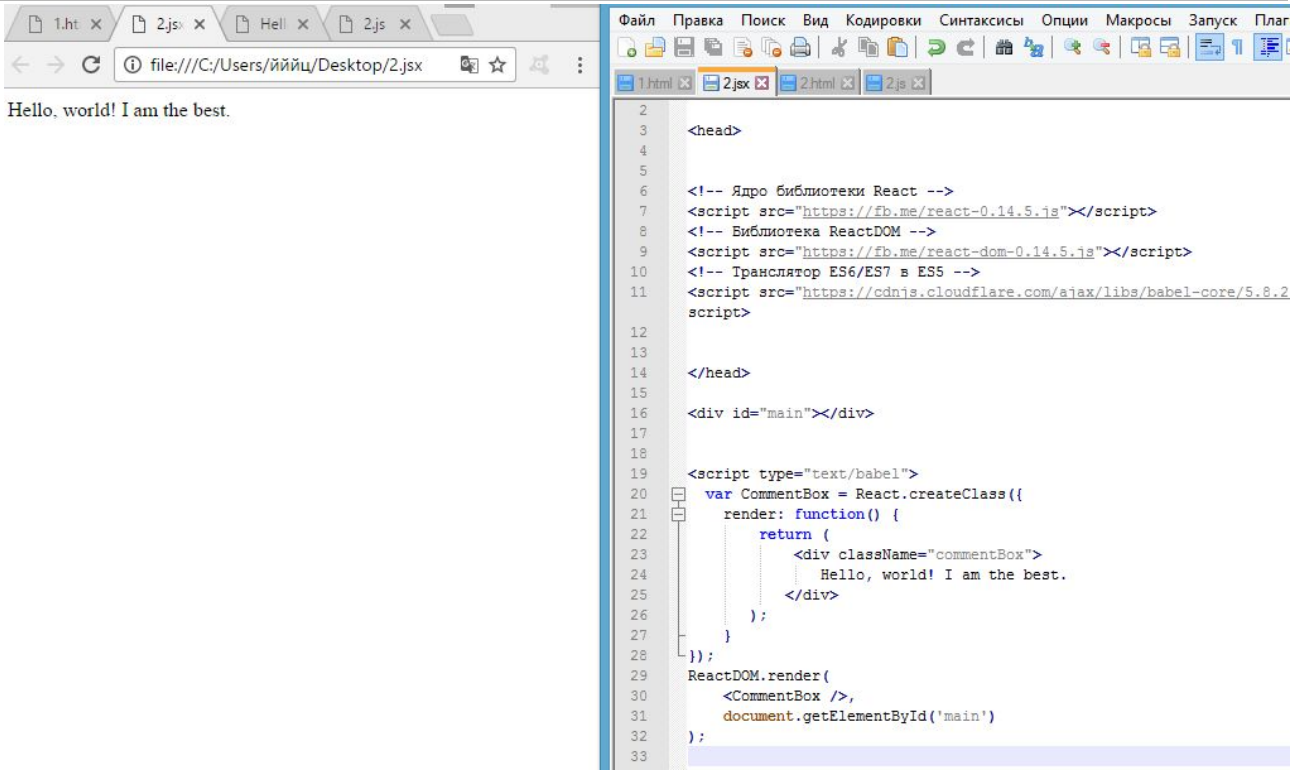
В данном примере функцией render переданы компоненту Photo 2 свойства: imageURL и caption.Внутри пользовательской функции render, свойство imageURL используется в качестве src для изображения. Свойство caption используется как текст элемента span. Свойства компонента неизменяемы.

Создаём и отрисовываем первый компонент

В основе React лежат модульные компоненты. Для блока комментариев у нас будет следующая структура компонентов:

Создаем простой и в нем компонент CommentBox

```
varCommentBox = React.createClass({
   render: function() {
            return (
<div className="commentBox">
                    Hello, world! I am the best.
     </div>
      );
   }
});
ReactDOM.render(
<CommentBox />,
document.getElementById('main')
);
```



Добавляем Layout компонент

Менеджер расположения (layoutmanager) определяет, каким образом на форме будут располагаться компоненты. Независимо от платформы, виртуальной машины, разрешения и размеров экрана менеджер расположения гарантирует, что компоненты будут иметь предпочтительный или близкий к нему размер и располагаться в том порядке, который был указан программистом при создании программы.

В стандартной библиотеках существует много готовых вариантов менеджеров расположения, и с их помощью можно реализовать абсолютно любое расположение.

Подключаем:

```
pmninstall react-layout-components
```

Эти компоненты удобнее всего прописать в отдельном файле и подставлять потом в код программы в <head>.

Пример компонента:

```
react-layout-components--box {

display: -webkit-box;

display: -moz-box;

display: -ms-flexbox;

display: -webkit-flex;

display: flex;

}
```

Определяет цвет, размер, шрифт, расположение и т.д.

Определяемся с набором компонентов нашего приложения

Создадим заготовки для компонентов CommentList и CommentForm, которые, опять же, будут простыми -ами. Добавим эти два компонента в наш файл рядом с компонентом CommentBox:

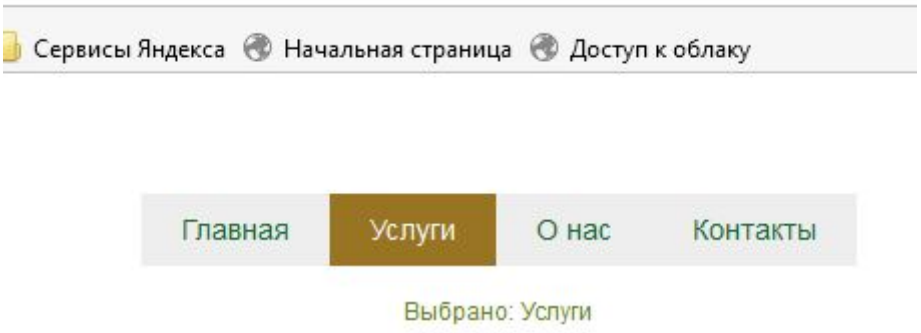
```
var CommentList = React.createClass({
render: function() {
return (
<divclassName="commentList">
      Hello, world! I am a CommentList.
</div>
   );
 }
});
varCommentForm = React.createClass({
render: function() {
return (
<divclassName="commentForm">
      Hello, world! I am a CommentForm.
</div>
   );
 }
});
```

Далее обновим компонент CommentBox так, чтобы он использовал новые компоненты:

```
varCommentBox = React.createClass({
  render: function() {
    return (
<div className="commentBox">
<h1>Comments</h1>
<CommentList />
<CommentForm />
</div>
    );
  }
});
```

Реализуем заготовки компонентов системы

Разберём пример «Меню»



Цель: Определить, какие компоненты использовались в данном примере?

```
<div id="main"></div>
<script type="text/babel">
var MenuExample = React.createClass({
        getInitialState: function(){
        return { focused: 0 };
        },
        clicked: function(index){
        // Обработчик click будет обновлять состояние с
         // с индексом нажатого элемента меню
        this.setState({focused: index});
        },
        render: function() {
        // Здесь мы будем читать свойство items, которое было
         // принято в качестве атрибута при создании компонента
        var self = this;
        // Метод map пройдется по массиву с пунктами меню,
        // и возвратит массив с элементами &lt;li&gt;.
        return (
        <div>
        <ul>{ this.props.items.map(function(m, index){
                var style = '';
                if(self.state.focused == index){
                style = 'focused';
                }
          // Обратите внимание на использование метода bind().
         // Он делает index доступным в функции clicked:
                return <li className={style} key={index} onClick={self.clicked.bind(self, index)}>{m}</li>;
        }) }
        </ul>
        <p>Выбрано: {this.props.items[this.state.focused]}</p>
        </div>
        );
        }
});
        // Рендер компонента меню на странице с передачей массива с пунктами меню
ReactDOM.render(
        <MenuExample items={ ['Главная', 'Услуги', 'О нас', 'Контакты'] } />,
        document.getElementById("main")
);
</script>
```

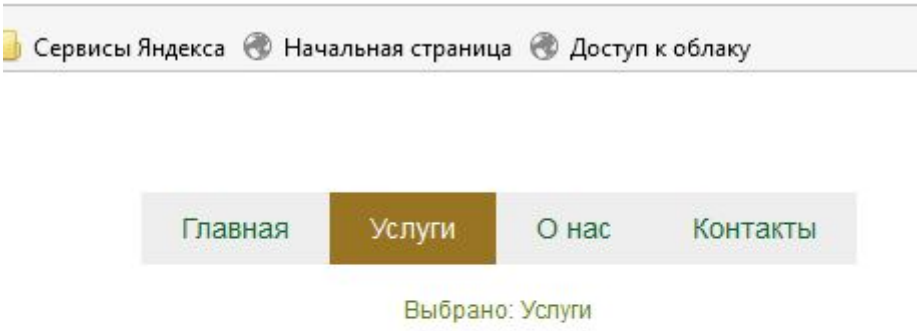
Пропишим свойства компонентов данной страницы:

```
<style>
* {
	padding:0;
	margin:0;
}
html{
	font:14px normal Arial, sans-serif;
	color:#658321;
	background-color:#fff;
}
body{
	padding:60px;
	text-align: center;
}
ul{
	list-style:none;
	display: inline-block;
}
ul li{
	display: inline-block;
	padding: 10px 20px;
	cursor:pointer;
	background-color:#eee;
	color:#156831;

	transition:0.3s;
}
ul li:hover{
	background-color:#beecea;
}
ul li.focused{
	color:#fff;
	background-color:#987423;
}
p{
	padding-top:15px;
	font-size:12px;
}
  </style>
```

Практика

Задача 1. Меню.

Разберём пример «Меню»



Цель: Определить, какие компоненты использовались в данном примере?

```
<div id="main"></div>
<script type="text/babel">
var MenuExample = React.createClass({
        getInitialState: function(){
        return { focused: 0 };
        },
        clicked: function(index){
    // Обработчик click будет обновлять состояние с
    // с индексом нажатого элемента меню
        this.setState({focused: index});
        },
        render: function() {
    // Здесь мы будем читать свойство items, которое было
    // принято в качестве атрибута при создании компонента
        var self = this;
    // Метод map пройдется по массиву с пунктами меню,
    // и возвратит массив с элементами &lt;li&gt;.
        return (
        <div>
        <ul>{ this.props.items.map(function(m, index){
                var style = '';
                if(self.state.focused == index){
                style = 'focused';
                }
    // Обратите внимание на использование метода bind().
    // Он делает index доступным в функции clicked:
                return <li className={style} key={index} onClick={self.clicked.bind(self, index)}>{m}</li>;
        }) }
        </ul>
        <p>Выбрано: {this.props.items[this.state.focused]}</p>
        </div>
        );
        }
});
     // Рендер компонента меню на странице с передачей массива с пунктами меню
ReactDOM.render(
        <MenuExample items={ ['Главная', 'Услуги', 'О нас', 'Контакты'] } />,
        document.getElementById("main")
);
</script>
```

Пропишим свойства компонентов данной страницы:

```
<style>
* {
	padding:0;
	margin:0;
}
html{
	font:14px normal Arial, sans-serif;
	color:#658321;
	background-color:#fff;
}
body{
	padding:60px;
	text-align: center;
}
ul{
	list-style:none;
	display: inline-block;
}
ul li{
	display: inline-block;
	padding: 10px 20px;
	cursor:pointer;
	background-color:#eee;
	color:#156831;
	transition:0.3s;
}
ul li:hover{
	background-color:#beecea;
}
ul li.focused{
	color:#fff;
	background-color:#987423;
}
p{
	padding-top:15px;
	font-size:12px;
}
  </style>
```

Домашнее задание

1. Реализовать шаблон главной страницы блога. Реализовать это необходимо в стиле react, то есть на выходе у Вас должен быть компонент Layout, который задаёт структуру страниц, главная страница (например, MainPage), на которой будет отображаться наша заглушка со статьями, а также элементы: menu и login.

   На странице должны присутствовать следующие элементы:

   1. Меню навигации
   2. Кнопка для логина
   3. Место для контента страницы
   4. *(Опционально) Боковое/нижнее меню
   5. *Используя bootstrap, реализуйте поведение для нажатие на кнопку login, а именно: при нажатии на кнопку login должно показываться модальное окно, где можно ввести логин и пароль для авторизации на сайте. При этом возможно использовать как react-bootstrap пакет, так и чистый bootstrap.

Задачи со * являются заданиями повышенной сложности.

Дополнительные материалы

1. starterkit
2. https://facebook.github.io/react/docs/installation.html
3. https://facebook.github.io/react/docs/introducing-jsx.html
4. https://facebook.github.io/react/docs/jsx-in-depth.html
5. https://habrahabr.ru/post/248799/
6. https://habrahabr.ru/post/249107/
7. https://facebook.github.io/react/docs/typechecking-with-proptypes.html
8. https://facebook.github.io/react/docs/components-and-props.html
9. https://facebook.github.io/react/docs/state-and-lifecycle.html

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. http://websketches.ru
2. https://github.com
3. http://webtoks.ru
4. http://www.pvsm.ru
5. https://habrahabr.ru