**Введение**

**Как строилось приложение? Лишние операции**

React - это JS библиотека для построения пользовательских интерфейсов

React JS это отличное современное решение для создания мощных динамичных веб сайтов.

При помощи React JS вы можете создавать приложение, на которые ранее могли уходить не просто часы, а целые недели. React очень упрощает процесс создания сайта, а также делает его более отзывчивым и легким для доработки.

Кроме того, многие современные компании обязательно требуют от новичков знаний React JS, поэтому овладев этой библиотекой вам не будет равных на рынке труда!

Преимущества React:

1. Самая быстрая библиотека

2. Очень понятный код. Расширение JavaScript-a HTML-eм (JSX)

3. Компонентный подход (компонента содержит несколько нод)

4. Не нужно учить специальный синтаксис

5. Очень удобная обработка ошибок

**Сравнение по скорости с другими JS фреймворками:**







Детали сравнения с примерами кода <https://www.codementor.io/chrisharrington/react-vs-angularjs-vs-knockoutjs-a-performance-comparison-85hwzepbz>

**За счет чего React такой быстрый?**

Самая затратная операция в JS приложении - это работа с DOM. Для упрощения работы с DOM был создан Virtual DOM, в котором хранятся все состояния реального дома. И как только REACT увидит, что изменился виртуальный DOM, только тогда React поменяет что-то в реальном DOM. React хранит два состояния виртуального DOM. Он сравнивает предыдущее состояние с текущим и меняет только лишь отличие этих двух состояний в реальном DOM.





**Hello World!**

Для начала работы с React необходимо перейти на оф. сайт <https://facebook.github.io/react/docs/installation.html> и скачать React.



После выполнения комманд по адресу <http://localhost:3000/> будет доступна наша заготовка проекта.

Итак, разберем структуру проекта



Папка public содержит favicon, index.html, в котором будет загружаться наше приложение и manifest.json (детали <https://habrahabr.ru/post/275849/>).

С примером манифеста можно ознакомиться по этой ссылке . Кратко пройдемся по параметрам:

name – имя, которое будет отображаться под иконкой, ну и вообще везде, где будет отображаться ваше «приложение»

short\_name – будет использоваться в тех случаях, когда места для отображения полного имени недостаточно

icons – набор иконок разных размеров

start\_url – определяет url, которые открывается при нажатии иконки (можно использовать, чтобы зафиксировать пользователей, которые открывают сайт через иконку на рабочем столе, добавив параметр, допустим, ?src=homescreen в url)

display – отвечает за то, как будет отображаться ваш сайт (с адресной строкой без нее и т.п.)

background\_color – устанавливает цвет страницы до того как она загрузилась. Пока страница не загрузилась пользователь видит перед собой белое пустое поле. Чтобы как-то разукрасить его серые будни, можно изменить этот цвет. Например, поставить цвет фона сайта.

Больше всего нас будет интересовать папка src.

App.css, index.css - это наши стили

App.js - файл нашего главного компонента

index.js - файл, отвечающий за добавление нашего приложения в index.html

registerServiceWorker.js - это файл, который регистрирует воркер для определения работы приложения в оффлайн режиме.

package.json - файл в котором прописаны базовые настройки приложения, его имя, версия, зависимости и т.д.

**JSX**

Как вы думаете, что это такое? HTML в JS?

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

Этот синтаксис называется JSX и представляет собой расширение языка в JavaScript. JSX может напоминать вам HTML разметку, но он полноценно работает в JavaScript. JSX производит «элементы» React. Благодаря JSX мы намного упрощаем синтаксис

Вы можете вставить любой JavaScript-код в JSX, завернув его в {фигурные скобки}. Например, 2 + 2, user.name, и formatName(user) все эти выражения допустимы:

**function** *formatName*(user) {  
 **return** user.**firstName** + **' '** + user.**lastName**;  
}  
**const** user = {  
 **firstName**: **'Harper'**,  
 **lastName**: **'Perez'**};  
**const** element = (  
 <**h1**>  
 Hello, {*formatName*(user)}!  
 </**h1**>  
);  
**ReactDOM**.**render**(  
 element,  
 **document**.getElementById(**'root'**)  
);

[**https://codepen.io/gaearon/pen/PGEjdG?editors=0010**](https://codepen.io/gaearon/pen/PGEjdG?editors=0010)

Также JSX теги могут содержать дочерние модули:

**const** element = (  
 <**div**>  
 <**h1**>Hello!</**h1**>  
 <**h2**>Good to see you here.</**h2**>  
 </**div**>  
);

**Элементы React**

Элементы являются самыми мелкими компонентами приложения. Элемент описывает то, что вы хотите увидеть на экране.

Давайте предположим, что в вашем HTML файле находится div:

<**div id="root"**></**div**>

Мы назовем это корневым узлом DOM, поскольку все, что находится внутри него, будет управляться React DOM. Приложения, созданные только с помощью React, обычно имеют один корневой узел DOM. Для отображения элемента React на корневом узле DOM, выполните ReactDOM.render():

**const** element = <**h1**>Hello, world</**h1**>;  
**ReactDOM**.**render**(  
 element,  
 **document**.getElementById(**'root'**)  
);

Элементы React являются неизменными (immutable). После того, как вы создали элемент, вы не можете изменить его дочерние модули или атрибуты.

Обладая самыми новейшими знаниями программирования, единственным способом обновить UI является создание нового элемента и его передача в ReactDOM.render().

**function** *tick*() {  
 **const** element = (  
 <**div**>  
 <**h1**>Hello, world!</**h1**>  
 <**h2**>It is {**new** Date().toLocaleTimeString()}.</**h2**>  
 </**div**>  
 );  
 **ReactDOM**.**render**(  
 element,  
 **document**.getElementById(**'root'**)  
 );  
}  
setInterval(*tick*, 1000);

[**https://codepen.io/gaearon/pen/gwoJZk?editors=0010**](https://codepen.io/gaearon/pen/gwoJZk?editors=0010)

React обновляет только то, что необходимо

React DOM сравнивает элементы и их дочерние модули с предыдущими и позволяет обновлять DOM только тогда, когда его необходимо обновить до нужного состояния.

**Даже если мы создадим элемент, описывающий все дерево UI на каждом тике, только текстовый узел, чье содержание изменилось, будет обновляться React DOMом.**

**Создание первого компонента**

Компоненты позволяют разделять интерфейс на самостоятельные повторно применяемые части и анализировать каждую из них в отдельности. По идее, компоненты это функции JavaScript. Они принимают произвольное свойство и отдают обратно React элементы, описывающие то, что должно отображаться на экране.

**Функциональные и классовые компоненты**

Самый простой способ определить компонент это написать JavaScript функцию:

**function** *Welcome*(props) {  
 **return** <**h1**>Hello, {props.**name**}</**h1**>;  
}

Также компонент можно создать при помощи класса, для этого в классе обязательно должен быть реализован метод render()!!!!!!!!!:

**class** Welcome **extends** React.**Component** {  
 render() {  
 **return** <**h1**>Hello, {**this**.**props**.**name**}</**h1**>;  
 }  
}

После создания компоненты мы можем его отрендерить у нас на странице:

**function** *Welcome*(props) {  
 **return** <**h1**>Hello, {props.**name**}</**h1**>;  
}  
**const** element = <**Welcome name="Sara"** />;  
**ReactDOM**.**render**(  
 element,  
 **document**.getElementById(**'root'**)  
);

[**https://codepen.io/gaearon/pen/YGYmEG?editors=0010**](https://codepen.io/gaearon/pen/YGYmEG?editors=0010)

Далее мы можем использовать созданный компонент сколько угодно раз. Единственное условие - это все компоненты должны быть обернуты одним блоком:

**function** *Welcome*(props) {  
 **return** <**h1**>Hello, {props.**name**}</**h1**>;  
}  
**function** *App*() {  
 **return** (  
 <**div**>  
 <**Welcome name="Sara"** />  
 <**Welcome name="Cahal"** />  
 <**Welcome name="Edite"** />  
 </**div**>  
 );  
}  
**ReactDOM**.**render**(  
 <**App** />,  
 **document**.getElementById(**'root'**)  
);

[**https://codepen.io/gaearon/pen/KgQKPr?editors=0010**](https://codepen.io/gaearon/pen/KgQKPr?editors=0010)

Теперь давайте посмотрим на содержимое нашего проекта. Пока нас интересует только файл index.js в папке src.

**import** React **from 'react'**;  
**import** ReactDOM **from 'react-dom'**;  
**import './index.css'**;  
**import** App **from './App'**;  
**import** *registerServiceWorker* **from './registerServiceWorker'**;  
  
ReactDOM.**render**(<**App** />, **document**.getElementById(**'root'**));  
*registerServiceWorker*();

Что же происходит в этом файле, чтобы страничка, которую вы видели у нас зарендерилась? В этом файле мы импортируем react, react-dom из node\_modules. Импортируем компонент App. Это как раз компонент, который мы будем с вами писать. Импортируем css. И вызываем метод render у пакета ReactDOM, куда передаем App компонент и дом елемент, куда мы хотим, чтобы он отрендерился.

**import** React, { **Component** } **from 'react'**;  
**import** logo **from './logo.svg'**;  
**import './App.css'**;  
**class** App **extends Component** {  
 render() {  
 **return** (  
 <**div className="App"**>  
 <**div className="App-header"**>  
 <**img src=**{logo} **className="App-logo" alt="logo"** />  
 <**h2**>Welcome to React</**h2**>  
 </**div**>  
 <**p className="App-intro"**>  
 To get started, edit <**code**>src/App.js</**code**> and save to reload.  
 </**p**>  
 </**div**>  
 );  
 }  
}  
**export default** App;

То, что вы видите здесь - это базовая заготовка абсолютно любого react приложения. Дальше просто создается компонент App, в него вкладываются другие компоненты, и так далее.

**Props, типы компонентов**

**Props -** это свойства, которые мы передаем нашему компоненту. Компоненты бывают умные и простые. Хорошей практикой считается создавать один умный компонент на странице и влаживать в него простые компоненты. При условии, что уровень вложености не очень глубокий. Умный компонент может получать данные например от бекенда, а простые получают данные от парента, которые им пробросили и что-то рендерят. Таким образом данные пробрасываются вниз по уровню вложенности. Давайте рассмотрим на примере:

Создадим в нашем проекте новый компонент Header



**import** React, { **Component** } **from 'react'**;  
**class** Header **extends Component**{  
 render(){  
 **console**.log(**'items'**, **this**.**props**.**items**);  
 **return**(  
 <**div**>  
 <**ul**>  
 {**this**.**props**.**items**.**map**((item,key) => {  
 **return** (  
 <**li key =** {key}>  
 <**a href=**{item.**link**}>{item.**label**}</**a**>  
 </**li**>  
 )  
 })}  
 </**ul**>  
 </**div**>  
 )  
 }  
}  
**export default** Header;

И поменяем компонент App.js:

**import** React, { **Component** } **from 'react'**;  
**import** Header **from './Header.js'**;  
**import** logo **from './logo.svg'**;  
**import './App.css'**;  
**const** menu = [  
 {  
 **link**: **'/articles'**,  
 **label**: **'Articles'** },  
 {  
 **link**: **'/contacts'**,  
 **label**: **'Contacts'** },  
 {  
 **link**: **'/posts'**,  
 **label**: **'Posts'** }  
]  
**class** App **extends Component** {  
 render() {  
 **return** (  
 <**div className="App"**>  
 <**Header items =** {menu}>vcbgfhgfnfgnfgnfg</**Header**>  
 </**div**>  
 );  
 }  
}  
**export default** App;  
  
Предположим, что нам пришел массив объектов нашего меню с сервера. В каждом объекте есть свойства link (ссылка) и label (название пункта меню). Мы создаем компонент Header, который будет нам рендерить меню. Далее мы влаживаем его в родительский компонент App и пробрасываем наш массив menu в props.

При использовании компонента все "атрибуты" (такие,как items) попадают во встроенный объект this.props и вытащить определенное свойство можно через точку this.props.items.

**Задания:**

**1. Создать пользовательский компонент для отображения шапки и меню из простых элементов**

**2. Создать футер сайта и продублировать на нем меню**

**3. Наполнить страницу публикациями нескольких статей (заголовок, краткое описание). Создать компонент одной статьи и отобразить на странице список таких компонентов с разным содержимым.**

**4\* Сделать тоже самое, что в 3 задании, используя объект, в котором будут все заголовки, описания, картинки всех статей.**

<https://abraxabra.ru/react.js/bystryy-start/ustanovka-react-js/>

<https://facebook.github.io/react/docs/installation.html>

<https://abraxabra.ru/blog/react-js/course-on-react-js-1-pilot-hello-react/>

<https://abraxabra.ru/blog/react-js/virtual-dom/>

<https://abraxabra.ru/blog/react-js/react-js-components/>

<https://abraxabra.ru/blog/react-js/course-on-react-js-3-jsx/>