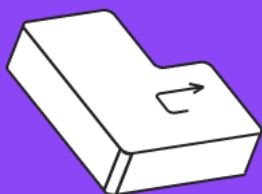


# Знакомство с ReactJS. Первые компоненты

React



# Оглавление

<b>Введение</b>	<b>2</b>
ReactJS. Основные принципы и возможности. Знакомство с JSX.	4
Основные принципы React	4
Первые компоненты. Знакомство с JSX	4
Чистые функции	6
<b>Create-react-app. Первое приложение</b>	<b>7</b>
Стилизация элементов	10
React Devtools	13
Недостатки create-react-app	13
Итоги урока	14
Что можно почитать еще?	14

## Введение

Первый вопрос который задают себе начинающие разработчики, это для чего мне изучать фреймворки, если я уже знаю javascript и с помощью данного языка программирования можно реализовать весь функционал на странице, поэтому первое с чего хочется начать, это понимание – а нужен ли нам фреймворк?

Сложные пользовательские интерфейсы включают в себя работу со списками, формами, управляющими элементами (такими как - кнопки слайдера и так далее). Как правило интерфейсы строятся из отдельных блоков, каждый из которых имеет какой-то интерактивный функционал, но сложность возникает при композиции нескольких элементов на странице или даже организации многостраничного приложения.

Пользовательские интерфейсы не сразу стали сложными. 10-15 лет назад достаточно было иметь статический сайт с формой для отправки заявки или заказа. С тех пор индустрия сильно развилась, а вместе с ней также выросли ожидания пользователей по возможностям, которые должна предоставлять веб-страница.

Часто бывает, что задачи, которые должно решать веб-приложение, основываются на сложной логике. Учитывая это, реализация такого сложного функционала требует применения нескольких парадигм для управления данными.

Какие именно проблемы возникают:

- отображение на странице переменных JS, которые могут меняться по какой-то логике
- действия, которые выполняют пользователи - клики по кнопкам, ввод данных в форме, - все это должно обрабатываться логикой скриптов
- необходимость упростить создание сложных приложений путем их разбиения на логические и функциональные блоки
- обеспечение воспроизводимости поведения - при выполнении одних и тех же действий мы должны получать тот же самый результат - и визуально на странице, и в модели данных
- отработка асинхронных процессов, которые могут выполняться параллельно, например загрузка данных и обработка других действий пользователя.

Учитывая озвученные проблемы, разрабатывать сложное приложение, которое будет обладать большим функционалом, может быть достаточно сложно. Чтобы упростить себе жизнь, можем воспользоваться специальной платформой, которая позволит облегчить разработку отдельных программных компонентов и объединить их в один проект.

По своей сути, мы сейчас почти что дали определение понятию “фреймворк”. В JavaScript существует множество фреймворков различной степени функциональности и, соответственно, популярности. В рамках нашего курса мы с вами рассмотрим фреймворк React.JS

На конец 2022 года самые популярные JavaScript-фреймворки – Angular, React и Vue. У каждого из этих фреймворков есть плюсы и минусы, но в целом все они подходят для создания приложений. Мы подробно остановимся на React.

1. AngularJS - “Java в мире фронтенда” - относительно сложный, тяжелый и многословный, но весьма мощный фреймворк, предоставляющий многие инструменты “из коробки”.
2. ReactJS - самый популярный на сегодня инструмент. Относительно прост в изучении, достаточно быстр, однако для многих задач требует установки дополнительных модулей (нехватки в которых, впрочем, нет, т.к. имеет одно из самых активных и обширных сообществ).
3. VueJS - относительно молодой фреймворк, создававшийся как “объединяющий лучшие качества React & Angular”.

Следует отметить, что официальная документация определяет React не как фреймворк, а как библиотеку для построения пользовательских интерфейсов (с этим связано, в частности, то, что для многих задач требуется установить дополнительные модули). Для многих разработчиков этот момент, впрочем, является спорным.

## **ReactJS. Основные принципы и возможности. Знакомство с JSX.**

### **Основные принципы React**

Одной из ключевых концепций приложений на React является декларативность. Сама библиотека построена таким образом, что разработчику удобно использовать декларативный подход - то есть мы указываем, какое состояние должен иметь интерфейс, а React сам позаботится о том, как этого состояния достичь. Этот подход противопоставляется императивному программированию, в котором разработчику требуется указать, какие действия необходимо совершить, чтобы достичь нужного состояния.

Кроме того, важной особенностью React является возможность переиспользования кода, что достигается за счет использования компонентного подхода. Разработчик может создать компонент React, использовать его в нескольких местах, объединять с другими, кастомизировать его и т.д.

Также важно отметить, что, хотя сам React и обладает меньшим количеством встроенных возможностей по сравнению с другими фреймворками, это позволяет разработчику практически полностью кастомизировать стек, на котором разрабатывается приложение, с помощью установки любого из множества сторонних модулей, подходящих для конкретной задачи.

### **Первые компоненты. Знакомство с JSX**

Как было указано выше, приложение на React строится на компонентах. Компонент - по сути, просто функция или класс javascript (класс должен наследовать от `React.Component` и реализовывать метод `render`). Соответственно, компоненты делятся на функциональные и классовые.

Конечно тут без примера не обойтись, установка будет чуть позже, нам необходима нотка вдохновения, чтобы понять что данная идея отлично реализуется и уже после этого можно будет всё попробовать на практике

### index.js

```
import React from 'react';

function AppHello() {
  return <h1>Hello World</h1>;
}

ReactDOM.render(<AppHello />, document.getElementById('root'));
```

Здесь создается функциональный компонент, который будет добавлять в DOM (“рендерить”) div с надписью “Hello world!”. Добавление корневого компонента в DOM осуществляется с помощью метода ReactDOM.render - этот метод принимает компонент, который необходимо добавить, и элемент DOM, в который следует добавить компонент. Как правило, в приложении вызов этого метода происходит лишь однажды - для корневого компонента, а он, в свою очередь, рендерит остальные компоненты. Важно понимать, что, за исключением единственного вызова этого метода, манипулирования DOM напрямую при использовании React происходить не должно.

Обратите внимание, как происходит создание элементов в компоненте AppHello - синтаксис напоминает HTML, хотя написан в .js-файле. С помощью такой же записи компонент AppHello передается в ReactDOM.render. Такой синтаксис называется JSX - он был создан разработчиками React для упрощения написания декларативного, лаконичного кода. JSX позволяет использовать не только элементы DOM (как div в примере), но и другие компоненты (AppHello в примере). Элементам DOM в JSX можно ставить атрибуты аналогично тому, как это делается в HTML (за исключением атрибута class - так как это слово является в js ключевым, вместо него указывается атрибут className).

Компоненты всегда должны называться с большой буквы, например, AppHello - корректное название, а appHello - некорректное. Названия с маленькой буквы допустимы только для элементов DOM.

Казалось бы а зачем нам еще и атрибуты у компонента, а тут всё просто, с их помощью мы сможем передавать данные и использовать внутри компонента.

```
function AppHello(props) {
```

```

    return <h1>Hello, {props.name}</h1>;
  }

ReactDOM.render(<AppHello name="Alex" />,
  document.getElementById('root'));

```

Атрибуты можно передавать не только элементам DOM, но и компонентам. В этом случае они будут доступны внутри функции-компонента в специальном аргументе props - объекте, содержащем значения этих атрибутов (а в классовой компоненте в свойстве this.props). Такие атрибуты называются пропсами (сокращение от properties). С помощью пропсов мы можем передавать данные от родительских компонентов дочерним.

Обратите внимание, внутри JSX обращение к пропсам (а также любым другим переменным) осуществляется с помощью фигурных скобок: `<span>{variable}</span>`.

```

class AppHello extends React.Component {
  render() {
    return <h1>Hello, Alex</h1>
  }
}

ReactDOM.render(<AppHello />, document.getElementById('root'));

```

В данном примере аналогичный функционал реализован на классовых компонентах.

**Важно:** До недавнего времени функциональные компоненты обладали ограниченной функциональностью по сравнению с классовыми, поэтому на сегодняшний день можно встретить большое количество кода, написанного на классах. Сейчас большая часть приложений разрабатывается на функциональных компонентах, и в данном курсе мы будем в основном рассматривать именно их.

## Чистые функции

Одним из важных принципов построения приложения на React является разделение компонентов на презентационные компоненты (также называемые глупыми, dummy) и контейнеры. Контейнеры - компоненты, которые содержат некоторую логику, могут, например, отправлять запросы или обрабатывать переданные им

данные. Презентационные компоненты служат только для отображения переданных им данных, как правило, являются функциональными компонентами и чистыми функциями.

Чистая функция - функция, не имеющая побочных эффектов (т.е., не изменяющая переменные во внешней области видимости, не отправляющая запросы и т.п.), а также всегда зависящая только от переданных аргументов. Иными словами, для одних и тех же переданных аргументов такая функция всегда будет возвращать один и тот же результат.

```
function isBefore(timestamp) {  
  return timestamp < Date.now();  
}
```

Данная функция не является чистой, так как результат ее вызова зависит от текущего момента времени. Однако, можно преобразовать ее следующим образом:

```
function isBefore(timestamp, date) {  
  return timestamp < new Date(date).getTime();  
}
```

Теперь эта функция зависит только от переданных ей аргументов.

Код, написанный с помощью чистых функций, легче тестировать и оптимизировать. При создании приложений на React, как правило, стараются разделить презентационную и логическую части и сделать презентационные компоненты - чистыми функциями.

В данном уроке рассмотрены только презентационные компоненты.

## Create-react-app. Первое приложение

Мы узнали основы, не хватает создания первого проекта, для этого нам необходимо сделать несколько простых шагов.

1. Установить (или убедиться, что установлены) node и npm
2. Создать новую папку и открыть ее в редакторе кода
3. Запустить в терминале команду **npm create-react-app myapp**, где myapp - название вашего приложения

После этого скрипт создаст папку с названием, которое вы указали, а также выполнит установку и настройку необходимых для работы React модулей. Кроме

того, будет автоматически создан первый компонент в файле App.js. Для того чтобы увидеть результат работы скрипта, перейдите в папку myapp:

**cd myapp**

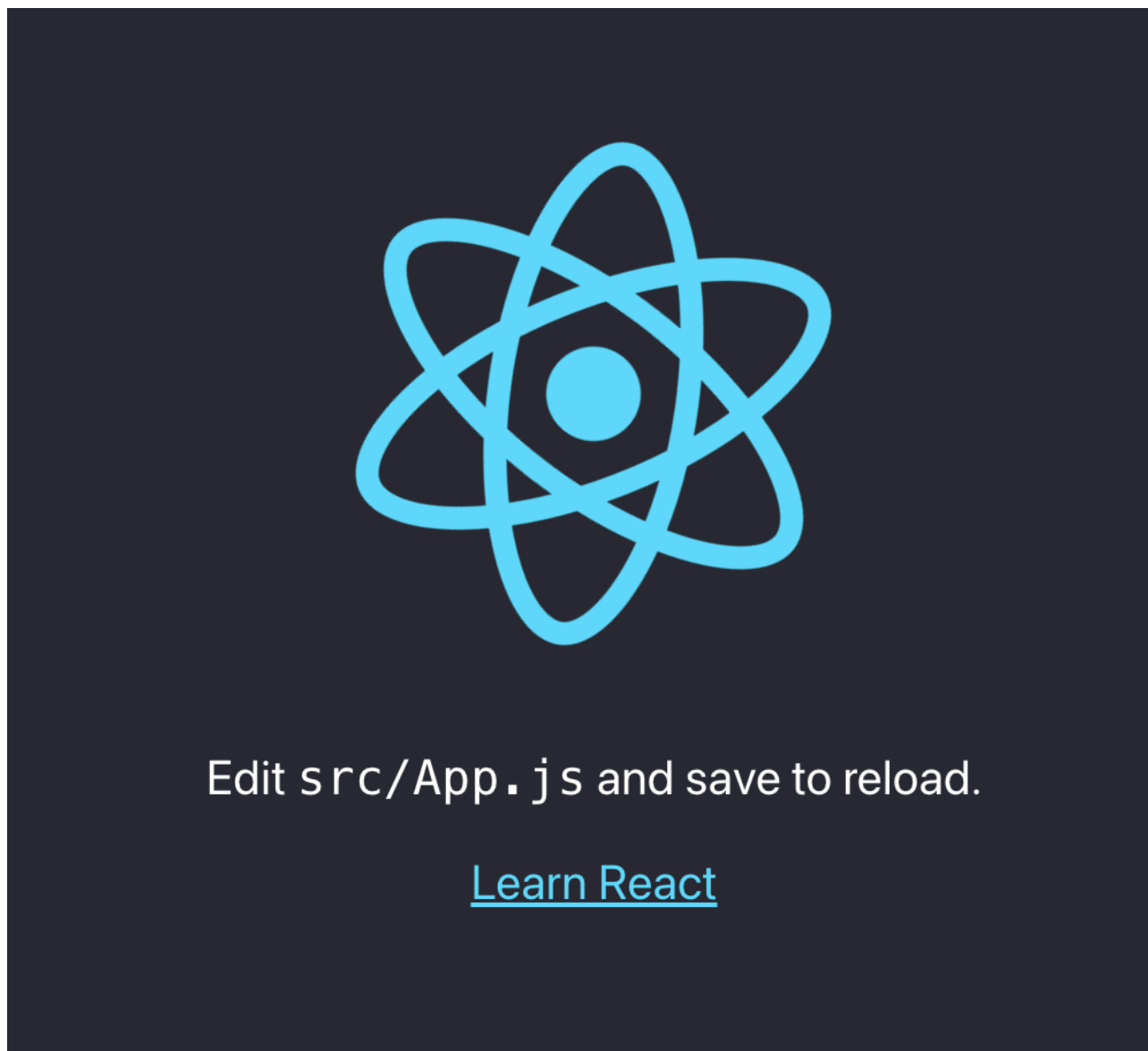
тут еще можно использовать

**cd .**

И запустите в терминале следующую команду:

**npm start**

В браузере откроется следующая вкладка:





Давайте разберем, что здесь происходит. В папке public находится файл index.html. Он является точкой входа в наше приложение. В его body находится только один div с id "root". В файле index.js с помощью ReactDOM.render в этот div рендерится компонент App, находящийся в файле App.js. Внутри этого компонента рендерится несколько элементов DOM, отображение которых мы и можем увидеть на странице в браузере.

Тут есть несколько вариантов работы с проектом, чаще всего удаляют все лишнее, так будет проще разобраться с тем что есть за файлы и не удалите ли вы что-то лишнее, так что тут я рекомендую разобраться как всё устроено внутри реакт, уделить буквально полчаса времени на анализ работы с созданным проектом, а затем уже переходить к чему-то новому.

Удалим существующий код компонента и создадим свои с использованием примеров из предыдущего раздела:

App.js

```
import './App.css';

function App(props) {
  return (
    <div className="App">
      <header className="App-header">
        My First React App
        <h3>Hello, {props.name}</h3>
      </header>
    </div>
  );
}

export default App;
```

Добавим передачу данных пропсами от родительского компонента компоненту App:

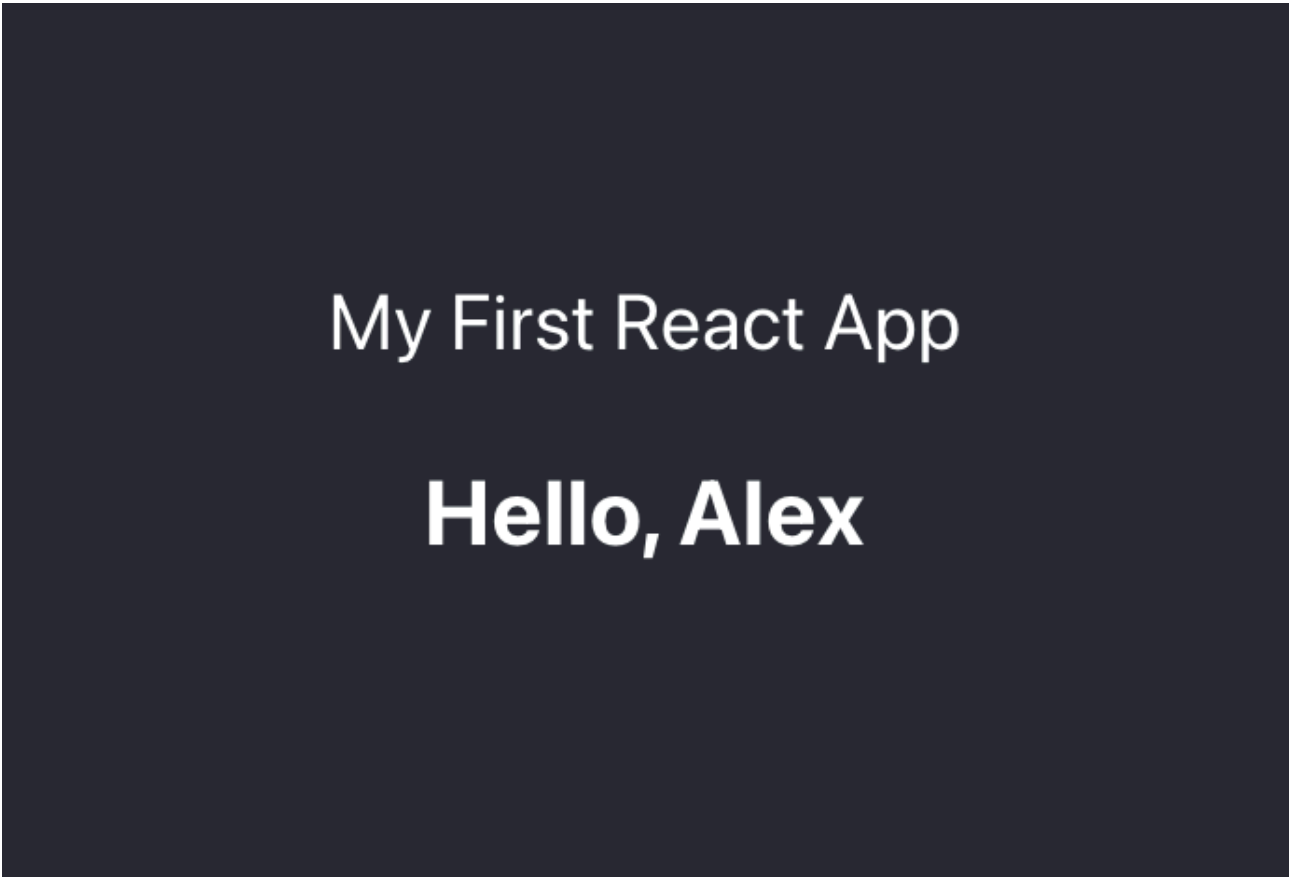
index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
```

```
import reportWebVitals from './reportWebVitals';
const myName = 'Alex';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App name={myName} />
  </React.StrictMode>
);

reportWebVitals();
```

Как итог

A screenshot of a web application with a dark blue background. The text "My First React App" is centered in a white, sans-serif font. Below it, the text "Hello, Alex" is also centered in a larger, bold, white, sans-serif font.

My First React App

**Hello, Alex**

Что хочется отметить, не так уж и сложно, но давайте думать чего нам не хватает, конечно же стилей, мы добавили произвольный контент, но если нам нужно добавить стили?

## Стилизация элементов

```
import './App.css';

function App(props) {
  return (
    <div style={{paddingTop: '25px'}}>
      Inline Styles
    </div>
  );
}

export default App;
```

Начнём сразу с примера, такой подход называется inline styles - обратите внимание, что здесь в свойство style передается объект. Его ключи похожи на значения свойств в CSS, но в них используется camelCase вместо kebab-case. Если значение свойства не является числом, его следует передать как строку.

Конечно нет задачи теперь всегда использовать инлайн стили, ведь у нас есть отличный файл App.css который уже подключен к проекту и может содержать стили, здесь вы можете писать уже знакомый нам CSS, плюс ко всему можно использовать препроцессоры (less, sass)

Давайте посмотрим сразу 2 варианта одновременно, создадим и класс и зададим inline стили

App.js

```
import './App.css';

function App(props) {
  const topPosition = '40px';
  return (
    <div>
      <header className="App-header" style={{ top: topPosition }}>
```

```

    My First React App
    <h3>Hello, {props.name}</h3>
  </header>
</div>
);
}

export default App;

```

Важный момент, на который хочется обратить внимание, это то что мы внутри `className` и `style` можем использовать вычисляемые свойства.

```

import './App.css';

function App(props) {
  return (
    <div>
      <header
        className={`App-header ${props.showRed ? 'header-red' :
'header-blue'}`}
        style={{ top: props.topPosition || '10px' }}
      >
        My First React App
        <h3>Hello, {props.name}</h3>
      </header>
    </div>
  );
}

export default App;

```

В данном примере элементу `header` будут установлены следующие стили:

1. Стили `css`-класса `App-header`
2. Если проп `showRed` передан, и его значение `truthy`, то будут установлены стили `css`-класса `header-red`, иначе - стили `css`-класса `header-blue`

3. Устанавливается значение свойства стиля top: если передан проп topPosition, то будет установлен он, в противном случае - значение 10px

**Важно:** При использовании CRA форматы css и less поддерживаются по умолчанию. Для использования sass необходимо установить node-sass:

```
npm install node-sass --save
```

## React Devtools

Для облегчения работы с приложением и его отладки существует специальное расширение для Chrome - React Devtools. Оно работает аналогично стандартным инструментам разработчика Chrome. Данное расширение устанавливается [из магазина расширений Chrome](#) и доступно для всех сайтов, на которых используется React.

После установки данного расширения запустите проект, в открывшейся вкладке откройте инструменты разработчика и перейдите на вкладку Components:

Тут нужно помнить что мы пока еще не создаем приложение, так что на данном этапе достаточно просто установить расширение и не требуется какая-то настройка или запуск, просто чтобы у нас был весь необходимый функционал.

## Недостатки create-react-app

Create-react-app - простой в использовании инструмент, позволяющий через один скрипт выполнить полную настройку React-проекта (включая настройку линтера, тестов, поддержку для файлов стилей и т.д.). Однако, такая простота достигается за счет сокрытия большинства настроек инструментов, требующихся для корректной работы проекта (в частности, Webpack, Babel, Jest). Файлы настроек для этих программ скрыты от разработчика (в проекте, созданном с помощью ручной настройки, эти файлы, как правило, находятся в корне проекта и доступны для редактирования).

В связи с этим выполнение тонкой конфигурации вышеуказанных инструментов представляет собой достаточно трудоемкую задачу - для этого требуется предварительно выполнить команду `npm run eject`.

Eject выполняет необратимую операцию “разборки” проекта. Не выполняйте ее, если не уверены в том, что делаете. На начальном этапе для изучения работы с самим React достаточно автоматической настройки через create-react-app.

В качестве упражнения самостоятельную настройку проекта можно выполнить в соответствии с этой статьей.

## Итоги урока

Как мы можем заметить ни установка, ни запуск, ни создание первых строчек кода не вызывает серьезных проблем, все это благодаря тому что React хорошо настроен на старте и не требует серьезных знаний. Конечно в дальнейшем вы будете писать более серьезный код и будете работать с созданием объемных проектов, но старт является фундаментом освоения проекта и поэтому тут необходима нотка практики, для этого следует попробовать создать свой первый проект и добавить первый код, главное не сильно бежать вперед, так как в обучении будет все необходимое.