

Создание веб-приложений с использованием **Angular & React**

Урок № 5

React: основы

Содержание

React: основы	3
Что такое React?	3
Цели и задачи React.....	4
Почему стоит использовать React?	4
Кто использует React?	5
Особенности React	5
Где происходит разработка React приложений?	6
Онлайновые редакторы для работы с React.....	6
Настройка онлайн-окружения	9
Структура проекта	19
Компоненты	26
Props	42
Домашнее задание.....	48

React: основы

Что такое React?

React — это JavaScript библиотека от компании Facebook для построения пользовательских интерфейсов. Конечно, вы можете создавать пользовательские интерфейсы на чистом JavaScript, однако это занимает большое количество времени и усилий. Гораздо разумнее освоить один из самых популярных инструментов веб-разработки — React.

Первая общедоступная версия библиотеки появилась 29 мая 2013 года. Хотя её история началась значительно раньше. Создателем React является Джордан Валке (разработчик из Facebook). Первые упоминания о React появились в новостной ленте Facebook в 2011 году. Сейчас поддержкой библиотеки занимается Facebook и большое сообщество разработчиков. React является проектом с открытым исходным кодом.

Больше об истории создания React вы можете прочесть в [Википедии](#). Рекомендуем это сделать только после прочтения этого урока. Ведь вас ждёт много очень важной и полезной информации!

Имя React носят две библиотеки:

- **React** — для построения веб-интерфейсов;
- **React Native** — для построения кроссплатформенных приложений для платформ iOS, Android и др.

Целью данного курса является только React. Однако, многие из изученных концепций пригодятся при знакомстве с React Native.

Веб-сайт React расположен по адресу <https://reactjs.org/>. Мы уверен, что вы будете часто его посещать.

Цели и задачи React

React был создан для построения пользовательских интерфейсов. Вы можете сказать, что для этого даже в 2013 году существовало громадное количество инструментов. И вы будете абсолютно правы. При создании React пытались максимально упростить создание интерфейсов, так как инструменты того времени были достаточно сложны. Простота — главный лозунг React. Им пропитана вся философия библиотеки. В самое ближайшее время у вас будет возможность в этом убедиться.

В основе приложения React лежит понятие компонента. **Компонент** — это некоторая сущность для решения конкретной задачи в вашем приложении. Например, компонент может отображать регистрационную форму или изображение с информацией о нём. Ваше React приложение будет состоять из набора таких компонентов. Если говорить более формально, **React приложение** — это композиция компонентов.

Почему стоит использовать React?

Причин для использования React очень много. Проанализируем некоторые из них:

- **Популярность** — большой спрос на специалистов в области React по всему миру
- **Скорость** — у React приложений очень быстрая скорость обновления интерфейса. И эта скорость не теряется даже при разработке больших и сложных

систем. Это преимущество достигается с помощью *Virtual DOM*. Об этом мы поговорим позднее.

- **Лёгкая встраиваемость** — React относительно несложно подключить к уже существующим проектам.
- **Изоморфность** — React может рендериться, как на клиенте, так и на сервере.
- **Проверен в боях** — React создан Facebook. И Facebook его активно использует на страницах своей социальной сети. Это значит, что все появившиеся баги, максимально быстро устраняются. До тех пор, пока Facebook использует React, ему не грозит забвение.
- **Простота** — мы уже говорили о простоте структуры приложений. Кроме того, React очень прост в обучении. В нём небольшое количество концепций, которые необходимо изучить для старта проекта.

Кто использует React?

Многие компании по всему миру используют его. Приведем несколько звучных имен: Microsoft, Amazon, Apple, Twitter, Adobe, Salesforce, Netflix, Dropbox, Airbnb, PayPal и многие другие. Сегодня этот вопрос должен стоять другим образом: Кто в мире не использует React? ☺

Особенности React

Что же делает React таким особенным? Почему он так популярен? Причин много. Некоторые из них мы уже обсуждали выше. Коснемся его технических преимуществ.

- **JSX** — механизм для внедрения разметки в код на JavaScript

- **Virtual DOM** — позволяет эффективно вносить изменения только в те части DOM, которые были обновлены

С другими важными концепциями вы познакомитесь чуть позднее.

Где происходит разработка React приложений?

Мы можем разрабатывать приложения React, как на локальном компьютере, так и онлайн. Для работы на локальном компьютере вам придется выполнить последовательность шагов, чтобы построить базовое приложение. Такой путь может вызвать серьёзные трудности при начальном погружении в новую технологию, поэтому мы начнем наше знакомство с React, используя онлайновый редактор. Это позволит нам освоить азы библиотеки, не углубляясь в большое количество деталей. Когда у нас появится фундамент знаний, мы обязательно настроим React окружение на вашем компьютере.

Онлайновые редакторы для работы с React

В интернете доступно большое количество онлайн-редакторов для работы с React. У каждого из них свои плюсы и минусы, как и у любого программного продукта. Сделаем небольшой обзор вариантов:

- **repl.it** — настоящий монстр в мире онлайн-редакторов. Он содержит гигантское количество шаблонов для разных языков и технологий программирования. Большое количество полезных инструментов доступно для разработчиков (рис. 1, 2).

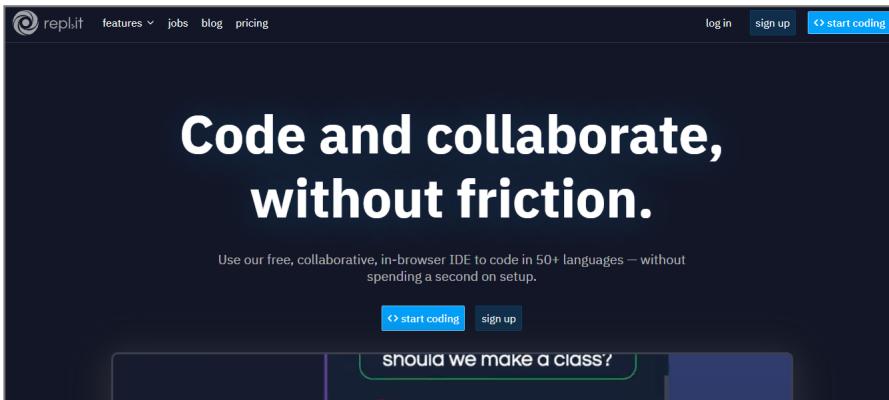


Рисунок 1

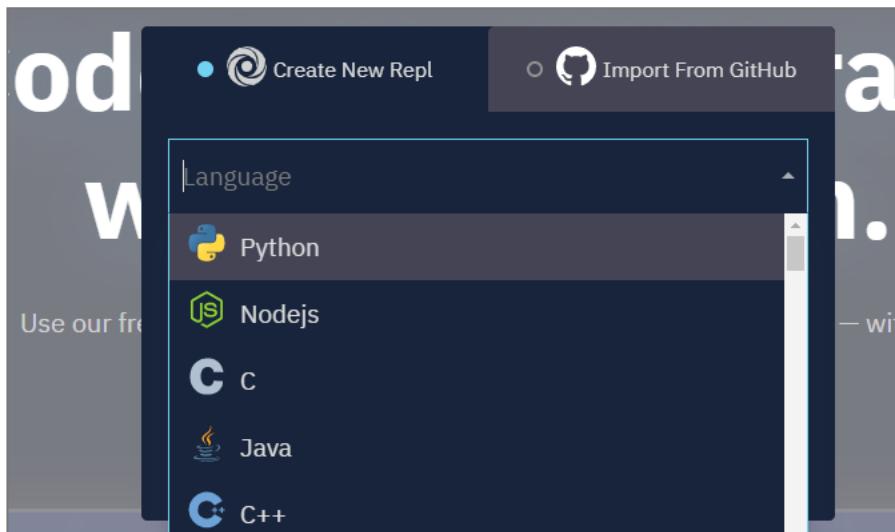


Рисунок 2

Ссылка на сайт: <https://repl.it/>.

- **jsfiddle.net** — один из самых старых и стабильных игроков на рынке онлайн-редакторов. Хорошая поддержка React и удобный дизайн (рис. 3).

Урок № 5

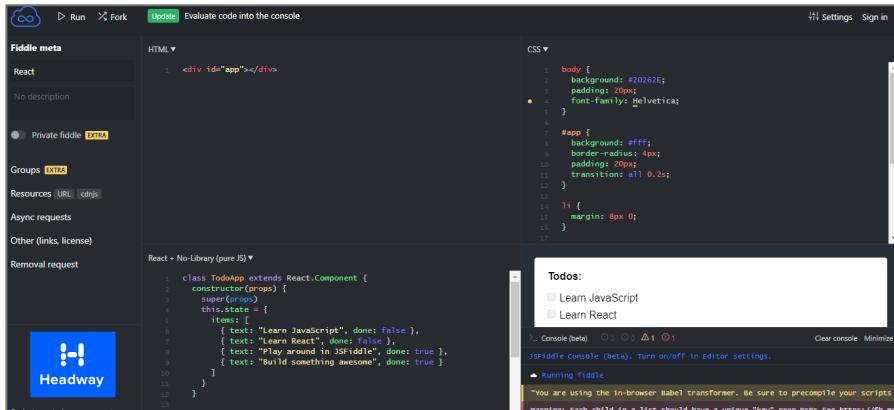


Рисунок 3

Ссылка на сайт <https://jsfiddle.net/>.

- **codepen.io** — один из самых популярных и известных онлайн-редакторов, большое количество пользователей, большое количество шаблонов проектов, поддержка React (рис. 4).

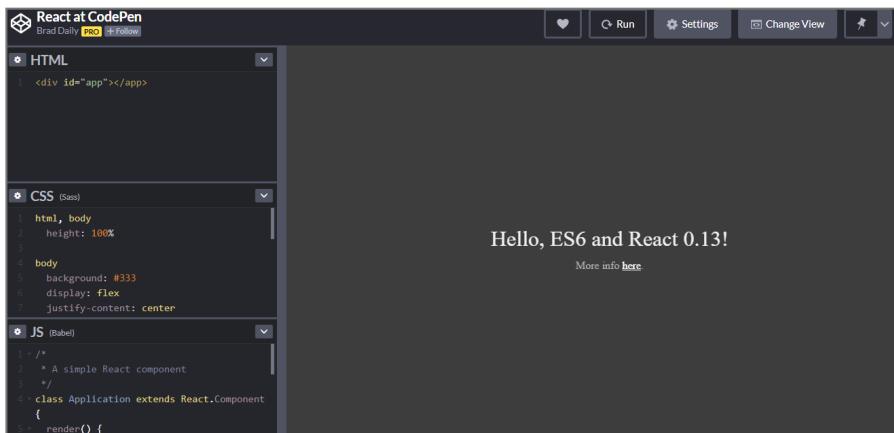


Рисунок 4

Ссылка на сайт <https://codepen.io/>.

- **codesandbox.io** — онлайн-редактор, который набирает всю большую популярность. Он предоставляет отличную функциональность и производительность, даже на бесплатном режиме. Именно его мы и будем использовать в наших уроках (рис. 5).

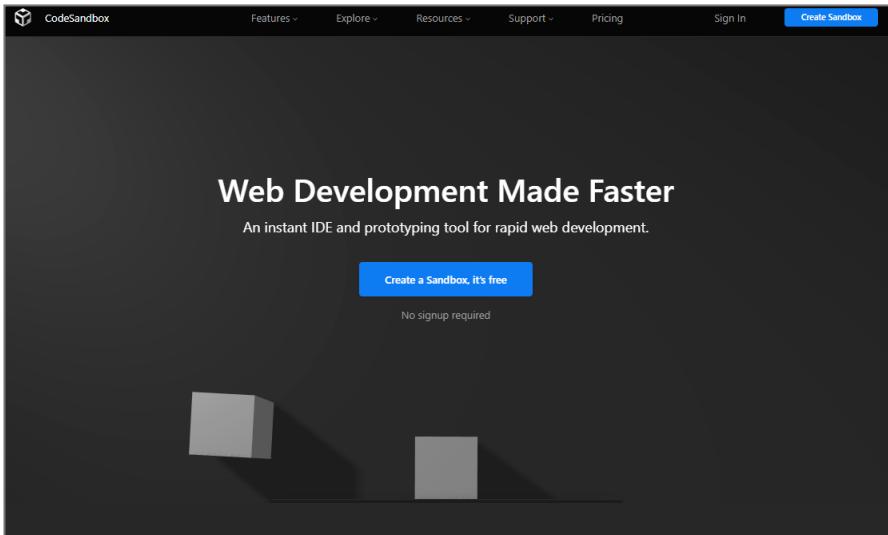


Рисунок 5

Ссылка на сайт <https://codesandbox.io/>.

Настройка онлайн-окружения

Для того, чтобы начать изучение React, нам нужно настроить наше онлайн-окружение. CodeSandbox использует аккаунт GitHub для логина, поэтому на первом шаге нам нужно создать аккаунт GitHub, если у вас его ещё нет. Вы можете лично убедиться в этом, если на сайте CodeSandbox кликнете по ссылке **Sign in** (рис. 6-7).

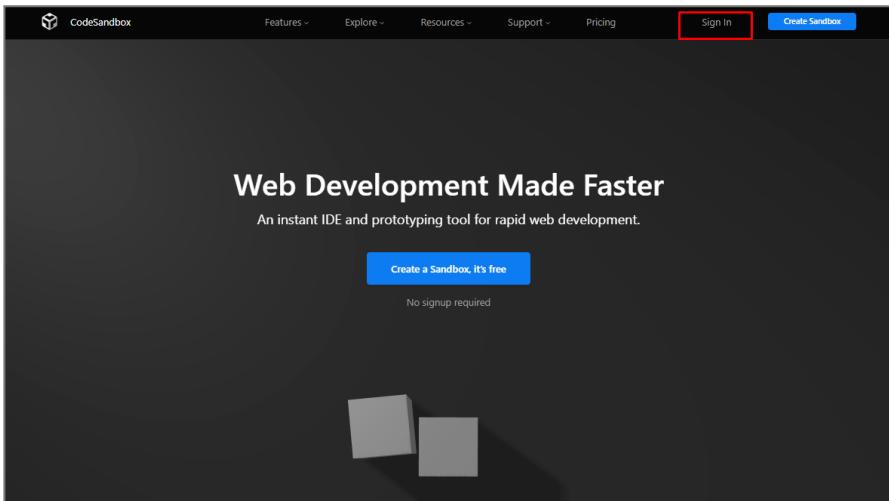


Рисунок 6

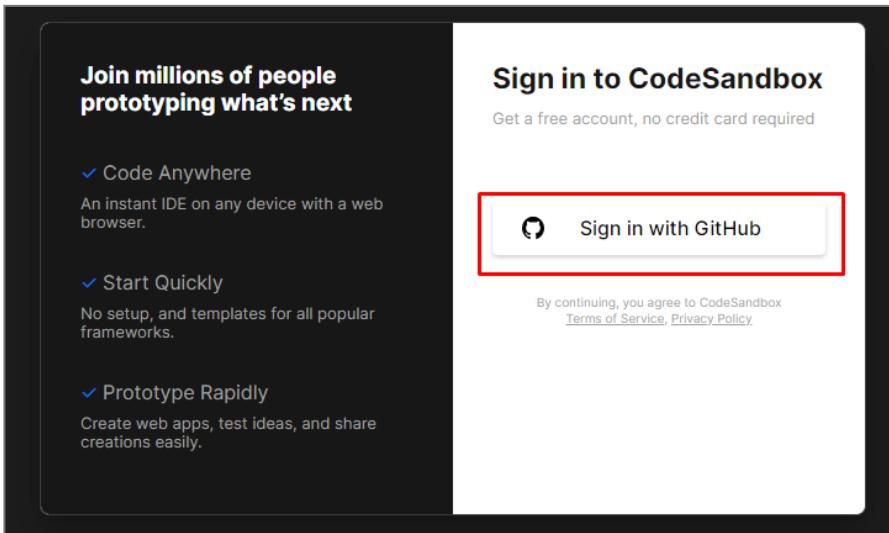


Рисунок 7

Для создания аккаунта зайдем на <https://github.com/> и заполним форму регистрации (рис. 8).

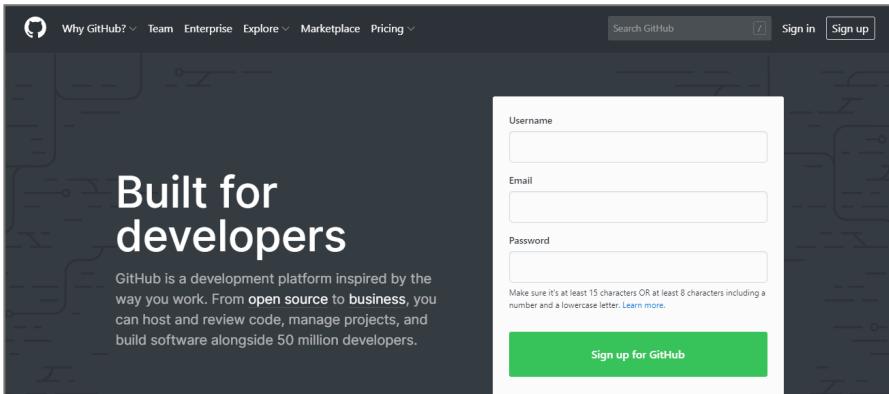


Рисунок 8

После регистрации аккаунта, вам на почту придет письмо для верификации созданного профиля. Не забудьте нажать на ссылку для подтверждения. После этого ваш аккаунт на GitHub станет полностью рабочим.

Теперь нажимаем на кнопку [Sign in with GitHub](#) на CodeSandbox и перед вами возникнет следующее окно (рис. 9):

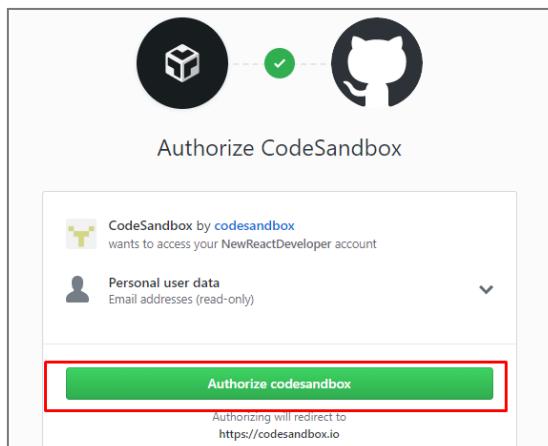


Рисунок 9

Нажмите кнопку [Authorize codesandbox](#) для завершения регистрации на CodeSandbox. Если всё прошло успешно вы увидите окно редактора (рис. 10).

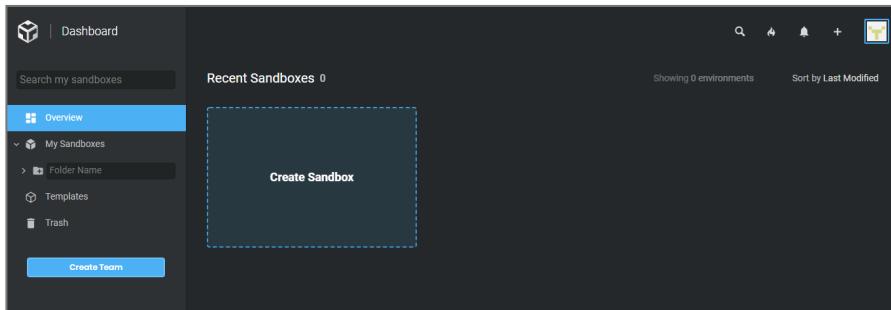


Рисунок 10

В основе этого онлайн-редактора лежит понятие песочницы (фактически, проект над которым мы работаем). Сейчас нам надо создать нашу первую React-песочницу.

Нажмите на кнопку [Create Sandbox](#) (рис. 11).

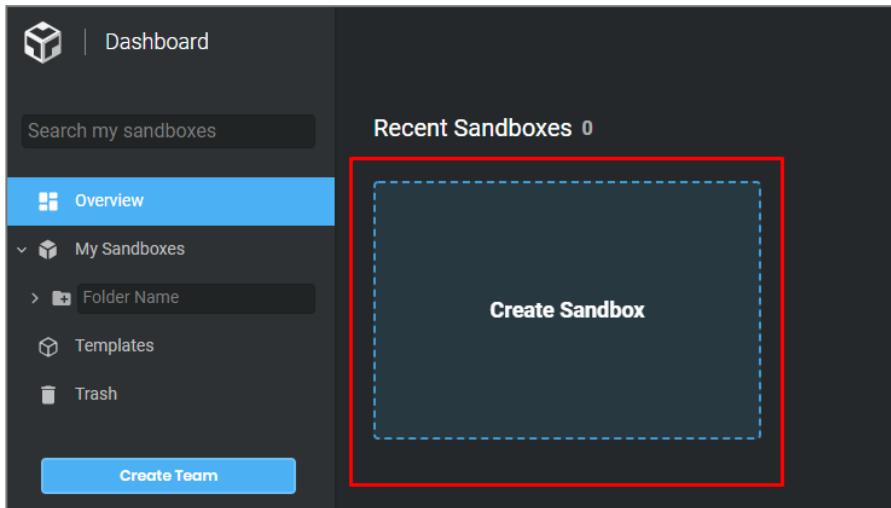


Рисунок 11

В новом окне выберите React. Это шаблон React приложения от команды CodeSandbox (рис. 12).

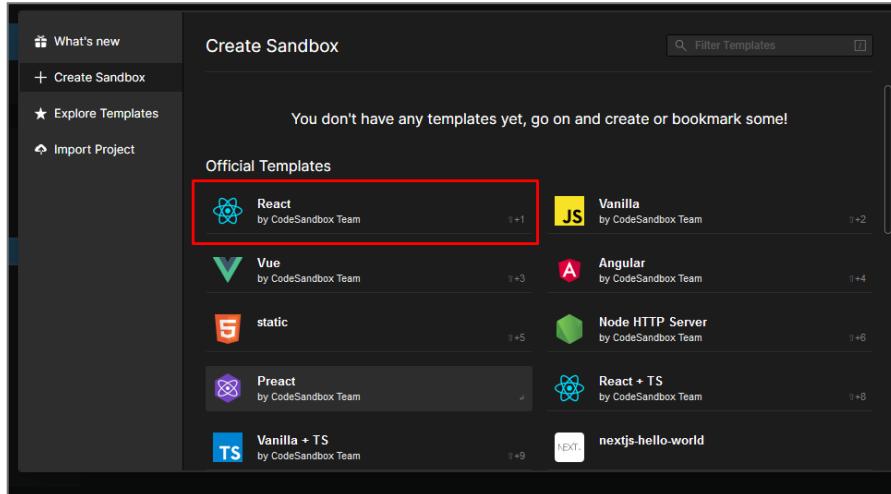


Рисунок 12

После клика на React у вас будет создан каркас приложения, который вы увидите в новом окне (рис. 13).

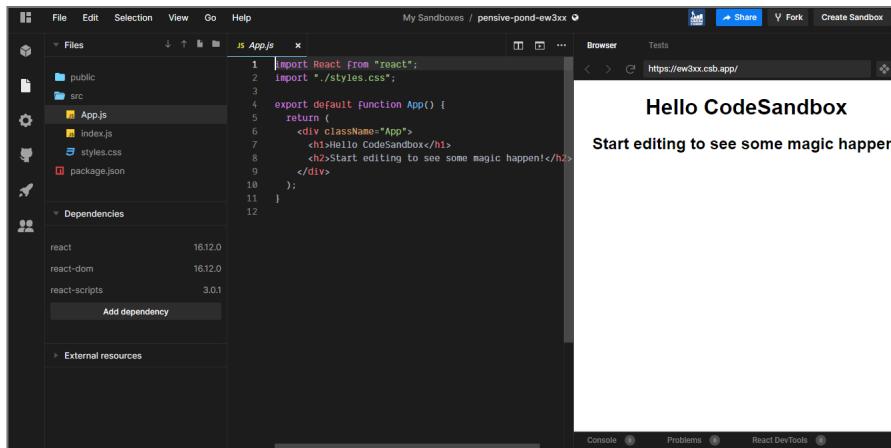


Рисунок 13

Ура! Мы создали каркас первого приложения! Самое сложное начать, дальше будет проще 😊.

Наступило время немного осмотреться внутри нашего редактора. Вся рабочая область по умолчанию поделена на три части. Слева находятся файлы нашего проекта (рис. 14).

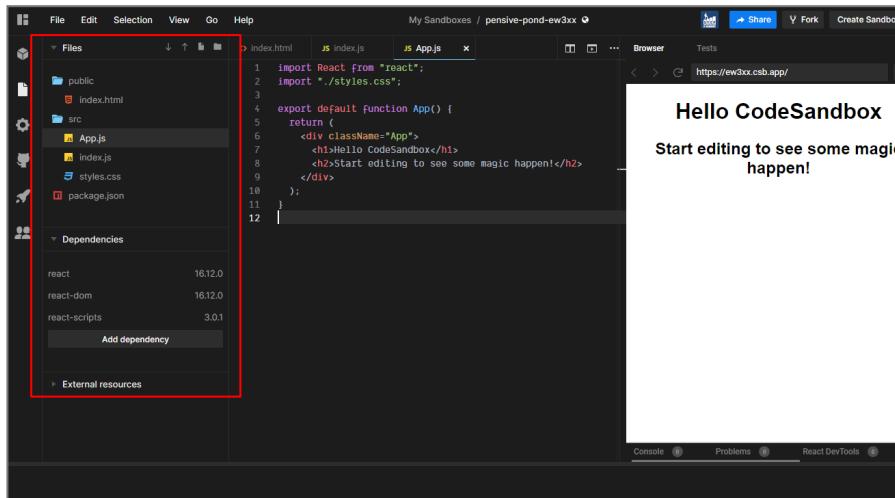


Рисунок 14

В нашем проекте есть две папки *public* (тут должны находиться файлы доступные извне) и *src* (исходный код нашего приложения). Кроме того, в корне нашего проекта находится файл *package.json* (в нём содержатся настройки проекта).

По центру окна располагается редактор исходного кода. Именно в нём мы будем редактировать текст наших файлов (рис. 15).

Сейчас открыт исходный код файла *App.js* нашего проекта. Мы видим код простейшего компонента.

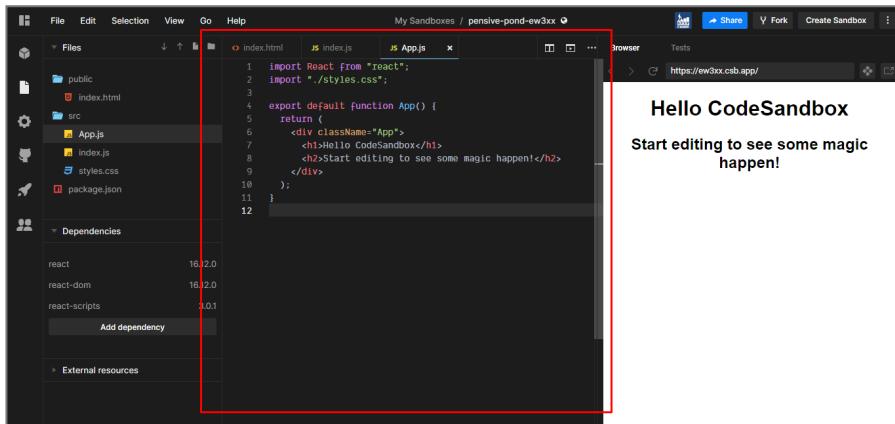


Рисунок 15

Справа располагается встроенное окно браузера. Его мы будем использовать для отображения нашего приложения. При изменении исходного кода, меняется отображение в браузере (рис. 16).

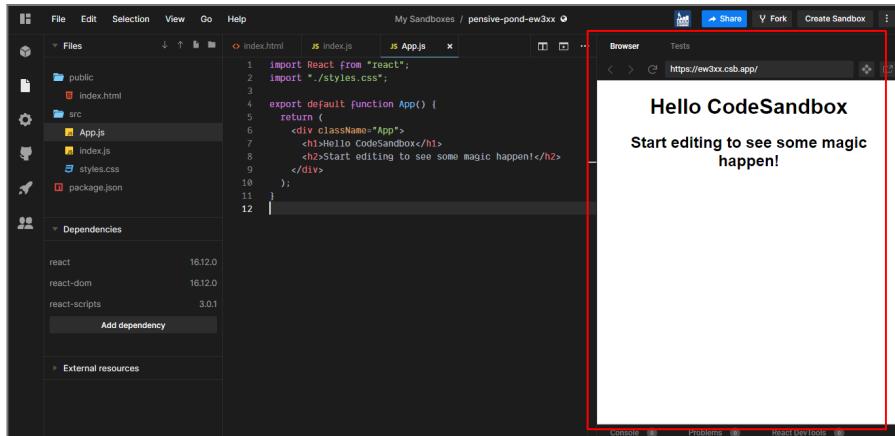


Рисунок 16

Обратите внимание на адресную строку внутри браузера. Она содержит адрес, который можно вставить в новое

окно браузера и перейти на ваше первое React-приложение (рис. 17-18).

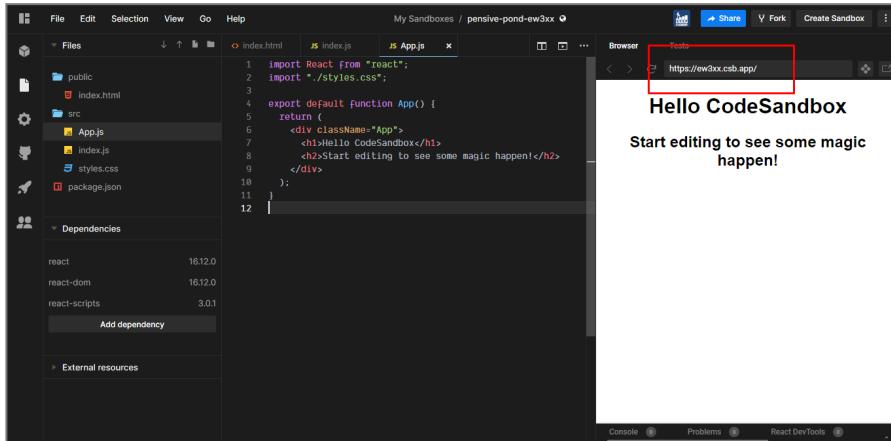


Рисунок 17

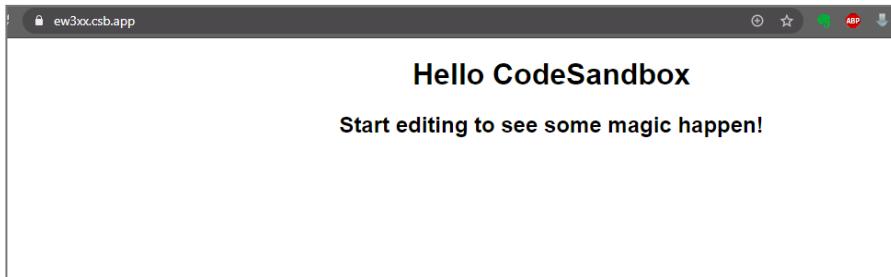


Рисунок 18

Какое название имеет наше новое приложение и как его поменять? Подведите мышку к верхней надписи [My Sandboxes/название](#) и кликните по ней (рис. 19-20).

Название моего проекта по умолчанию маловразумительное `pensive-pond-ew3xx`. Я поменяю его на `helloworld`. Рекомендуем вам, менять название проекта сразу после создания.

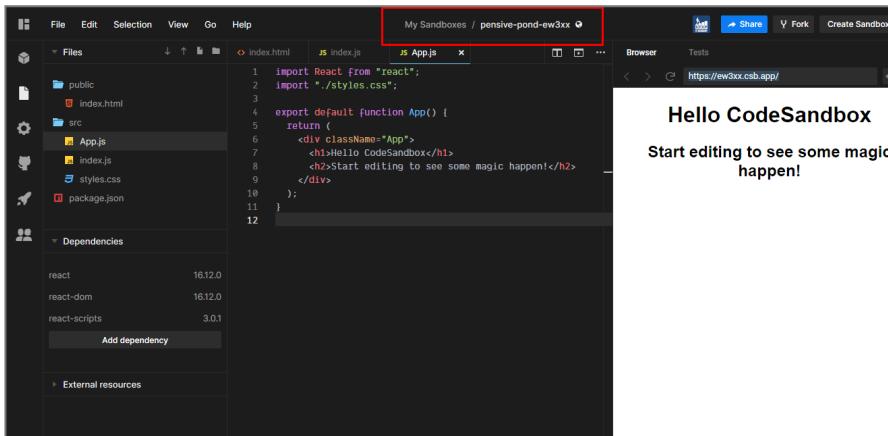


Рисунок 19

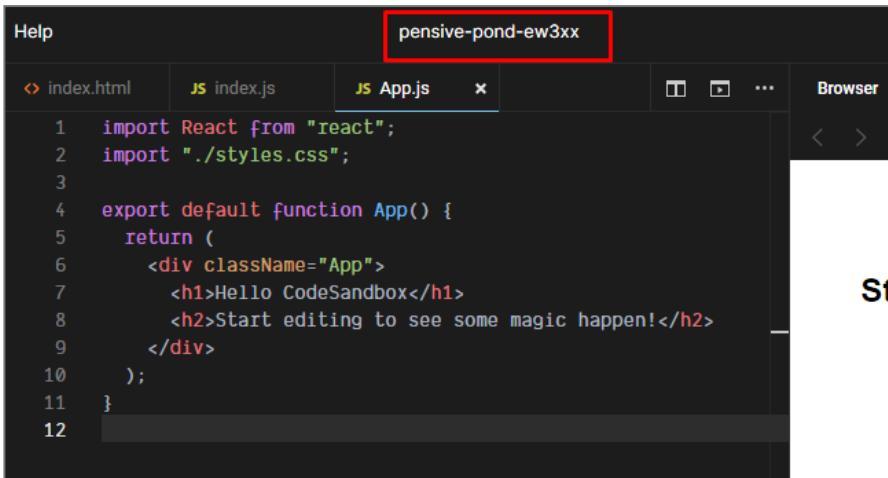


Рисунок 20

Для того, чтобы вернуться на стартовое окно кликните на квадрат в левом верхнем углу. Если вы не сохранили свои изменения, браузер предупредит вас об этом. Для сохранения изменений выберите меню **File** и нужный вам пункт из набора **Save** (рис. 21-22).

Урок № 5

The screenshot shows the 'My Sandboxes' interface with a project named 'helloworld'. The left sidebar displays a file tree with 'Files' and 'Dependencies'. Under 'Files', there are 'public' (with 'index.html'), 'src' (with 'App.js' marked as the current file, 'index.js', and 'styles.css'), and 'package.json'. Under 'Dependencies', there are 'react' (version 16.12.0), 'react-dom' (version 16.12.0), and 'react-scripts' (version 3.0.1). The right panel shows the code editor with the content of 'App.js':

```
1 import React from "react";
2 import "./styles.css";
3
4 export default function App() {
5   return (
6     <div className="App">
7       <h1>Hello world!</h1>
8       <h2>Start editing to see some magic happen!</h2>
9     </div>
10  );
11}
12
```

Рисунок 21

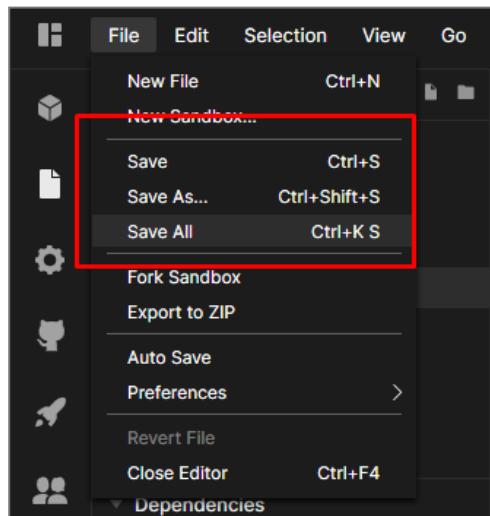


Рисунок 22

Мы вернулись на стартовое окно. Теперь в нём отображается наш первый, созданный проект (рис. 23).

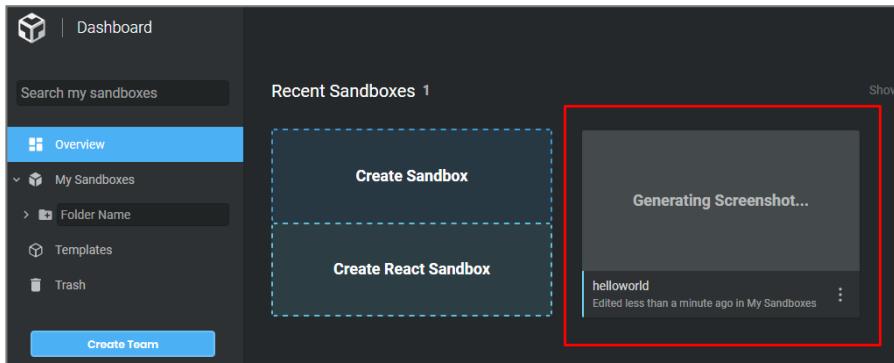


Рисунок 23

Если кликнуть на `helloworld` откроется наш проект.

Структура проекта

Приступим к анализу структуры нашего проекта. Мы уже знаем, что у нас есть несколько папок и набор файлов. Что же находится внутри каждого из них?

Начинаем с папки `public` (рис. 24).

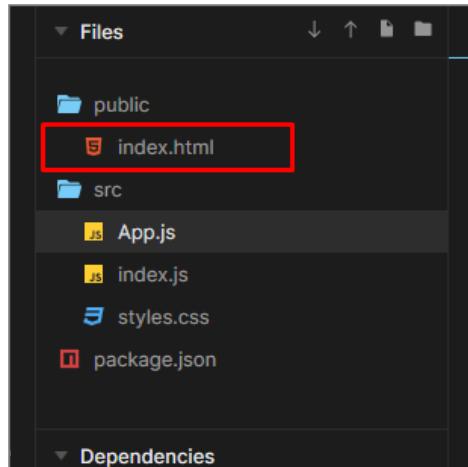


Рисунок 24

Даже из названия папки можно сделать вывод, что в этой папки должны быть файлы, которые будут общедоступны. Сейчас в нашей папке один файл *index.html*. В нём находится разметка нашей веб-страницы. К нему будут обращаться извне.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="theme-color" content="#000000">
  <!--
    manifest.json provides metadata used when your web app is added to the
    homescreen on Android. See https://developers.google.com/web/fundamentals/engage-and-retain/web-ap
    -->
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
  <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
  <!--
    Notice the use of %PUBLIC_URL% in the tags above.
    It will be replaced with the URL of the `public` folder during the build.
    Only files inside the `public` folder can be referenced from the HTML.

    Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
    work correctly both with client-side routing and a non-root public URL.
    Learn how to configure a non-root public URL by running `npm run build`.
  -->
  <title>React App</title>
</head>

<body>
  <noscript>
    You need to enable JavaScript to run this app.
  </noscript>
</body>
</html>
```

Рисунок 25

На изображении лишь часть этого документа. Нас волнует раздел **body**

```
<body>
  <noscript>
    You need to enable JavaScript to run this app.
  </noscript>
  <div id="root"></div>
</body>
</html>
```

Тут нет никакой особенной разметки, так как у нас React приложение и весь основной код будет в других папках. Обратите внимание на `div` с идентификатором `root`. Мы будем использовать его для внедрения нашего React кода.

Следующая очень важная папка `src`. Как видно из названия, это папка для хранения исходного кода. Внутри три файла.

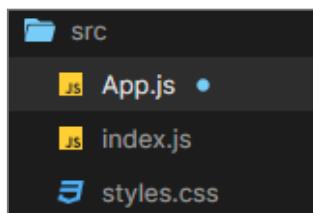


Рисунок 26

Начнем с файла `index.js`.

`Index.js` — точка входа в приложение React по умолчанию. С этого файла начинается исполнение приложения.

В коде этого файла мы получаем доступ к `div` с идентификатором `root` и подгружаем в него наш React компонент.

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

const rootElement = document.getElementById("root");
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  rootElement
);
```

Разберем код по строкам.

```
import React from "react";
import ReactDOM from "react-dom";
```

В первой строке мы импортируем пакет React в наш скрипт. В нём содержится базовая функциональность для любого React приложения. Без этого **import**, наш код не будет работать.

Вторая строка импортирует пакет ReactDOM. Он используется для работы Virtual DOM. Вы уже знаете, что такое DOM (*Document Object Model*). DOM — логическое, древовидное строение веб-страницы, которое содержит все ее элементы. Вы используете DOM из JavaScript для модификации страницы. Однако, любая модификация DOM является ресурсоёмкой, так как она заставляет браузер целиком перерисовать страницу. Virtual DOM является копией DOM. Когда вы вносите изменения в DOM из вашего React приложения, в реальности вы меняете Virtual DOM. Когда приходит пора поменять реальный DOM, React сравнивает отличия между Virtual DOM и DOM. По итогам сравнения обновляются только изменившиеся части страницы, без перерисовки всей страницы. Это даёт большой прирост производительности вашего приложения.

```
import App from "./App";
```

Мы импортируем наш пользовательский компонент под названием **App** из файла *App.js*.

Компонент — это строительный блок React приложения. Вы будете создавать большое количество компо-

нентов в рамках нашего курса. Название `App` не является ключевым словом. Его можно менять. `App` — это сокращение от `Application`

```
const rootElement = document.getElementById("root");
```

Если всё пройдет успешно `rootElement` будет ссылаться на `div` с идентификатором `root`. Вместо `const`, вы можете указывать `var` и `let`. Использование `const` это дань хорошему стилю программирования.

```
ReactDOM.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>,  
  rootElement  
) ;
```

В этой строке мы вызываем функцию `render` из пакета `ReactDOM`. Она используется для рендеринга React компонента. **Первый аргумент** функции — это React компонент, который мы хотим отобразить. `React.StrictMode` включает строгий режим проверки кода внутри данного тега. Обязательно наличие открывающего и закрывающего тега. `React.StrictMode` не имеет визуального отображения и будет работать только на этапе разработки, но не на продакшене. `App` — это наш отображаемый компонент. Форма `<App/>` это JSX о котором мы говорили. Он позволяет смешивать вместе JavaScript код и теги.

Второй аргумент — это ссылка на элемент, в котором мы хотим отобразить наш React компонент.

В файле *App.js* содержится код нашего компонента.

```
import React from "react";
import "./styles.css";

export default function App() {
  return (
    <div className="App">
      <h1>Hello CodeSandbox!</h1>
      <h2>
        Start editing to see some magic happen!
      </h2>
    </div>
  );
}
```

Разбираем его по строкам. Подключаем React и стили из файла *style.css*.

```
import React from "react";
import "./styles.css";
```

Теперь рассмотрим код нашего компонента.

```
export default function App() {
  return (
    <div className="App">
      <h1>Hello CodeSandbox!</h1>
      <h2>
        Start editing to see some magic happen!
      </h2>
    </div>
  );
}
```

Компонентом является функция под названием `App`. Код, указанный в `return` это, визуальная составляющая нашего компонента. Мы используем JSX для формирования визуальной части компонента. Наш компонент состоит из `div` и тегов `h1`, `h2` внутри него. Мы стилизуем его, используя стили из файла `style.css`. Для этого был указан атрибут `className`. В обычной HTML-разметке мы применяли атрибут `class`, в коде JSX он заменяется на `className`. Если код JSX состоит из нескольких строк, его надо взять в скобки. `Export default` указывает, что компонент `App` экспортируется наружу и его могут использовать другие модули. Без экспорта вы не сможете применять компоненты вне их файлов, так как они будут частными (закрытыми) для внешнего использования. В файле может быть только один `export default` (экспорт по умолчанию). Если вам нужно экспортировать из файла другие сущности, необходимо применять `export` без указания `default`. Мы обязательно рассмотрим этот механизм позднее в уроках.

Файл `style.css` содержит стили для нашего компонента.

```
.App {  
    font-family: sans-serif;  
    text-align: center;  
}
```

Файл `package.json` содержит настройки приложения. Например, в нём указана точка входа в наше приложение (рис. 27).

О других настройках этого файла мы поговорим более подробно позднее.

```
1
"main": "src/index.js",
"dependencies": {
  "react": "16.12.0",
  "react-dom": "16.12.0",
  "react-scripts": "3.0.1"
},
```

Рисунок 27

Вот мы и завершили разбор каркаса приложения. Существует большое количество шаблонов стартового приложения React. Если вы будете использовать другой онлайн — редактор ваш начальный код может отличаться. Тем не менее, основа будет очень похожа. Время приступить к созданию и настройке компонентов.

Компоненты

Что такое компонент?

Как вы уже знаете, приложение React состоит из набора компонентов. **Компонент** — это строительный блок для наших приложений. Компоненты дают возможность поделить наш UI (*User interface*) на независимые части, которые можно использовать отдельно друг от друга. Например, если вы для какого-то проекта создали компонент для отображения случайных чисел, вы можете его безболезненно перенести в другой проект. Если говорить с точки зрения программирования, компонент — фрагмент кода, решающий конкретную задачу и отображающийся в нашем интерфейсе. У вас уже был опыт создания первого компонента, но это была лишь часть айсберга.

Функциональные и классовые компоненты

Сейчас в React можно создавать два вида компонентов: функциональные ([function/functional components](#)) и классовые компоненты ([class components](#)). Изначально в React был только один вид компонентов — классовые компоненты. Из названия можно сделать очевидный вывод, что код компонента находится внутри класса. Большое количество кода было написано с использованием такого вида компонентов.

Функциональные компоненты (код компонента находится внутри функции) появились гораздо позднее. Однако, именно они сейчас являются рекомендуемым механизмом для создания компонентов. Это связано с удобством и простотой в написании кода. Если вы создаёте новое приложение, вам точно стоит использовать функциональные компоненты.

Значит ли это, что вам не нужно знать, как работать с классовыми компонентами? Конечно, нет. Если вам попадётся уже существующий проект с классовыми компонентами, ваши знания вам очень пригодятся.

В нашем курсе мы будем уделять больше внимания функциональным компонентам, но также не забудем и про классовый механизм. Давайте сразу договоримся об одном важном правиле: имя компонента всегда должно начинаться с большой буквы. React считает, что имена html-элементов начинаются с маленькой буквы, а имена всех компонентов с большой буквы.

Создание первого компонента

Создадим наше первое осознанное приложение React. Мы будем использовать функциональный компонент.

Наше приложение будет выводить известную строку из Шекспира «To be or not to be».

Создаём проект на *CodeSandbox*, как было описано выше. Меняем его имя на [Shakespeare](#) (рис. 28).

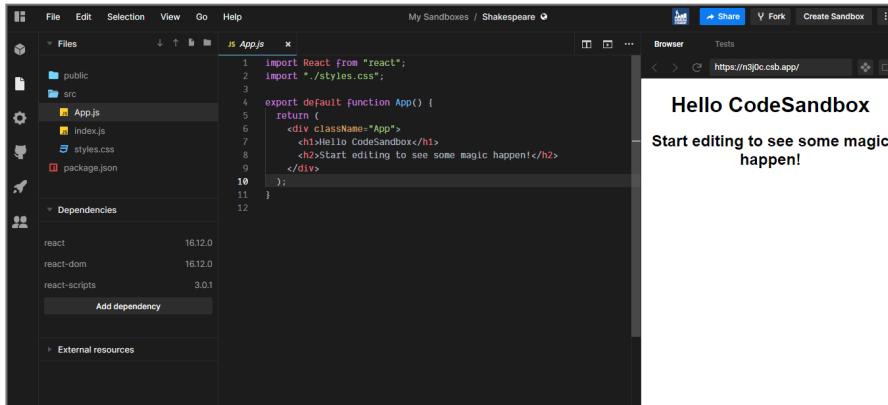


Рисунок 28

После создания проекта у нас тот же шаблон, который мы уже видели. Для начала изменим заголовок нашего приложения. Для этого зайдем в файл *index.html* и изменим содержимое тега **title**.

```
<title>Shakespeare</title>
```

Для вывода строки «To be or not to be» перейдем в файл *App.js* и изменим тот код, который возвращает [return](#).

```
import React from "react";

export default function App() {
  return (
    <div>
      <h1>To be or not to be,</h1>
    </div>
  );
}
```

```

        <h2>that is a question</h2>
      </div>
    );
}

```

Мы убрали использование стилей, поэтому у нас нет импорта стилей в начале файла. Код нашего первого функционального компонента полностью готов и работает (рис. 29).

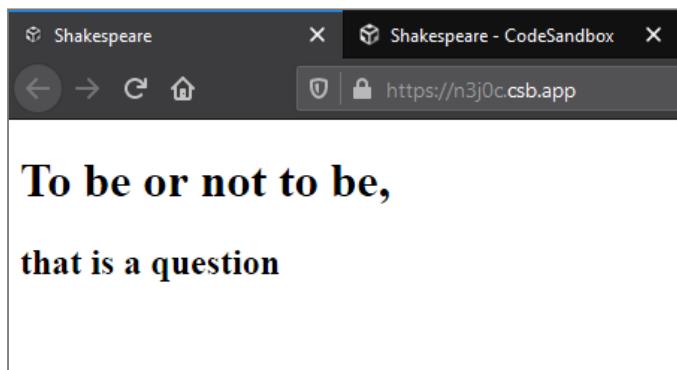


Рисунок 29

Доступ к коду этого проекта по ссылке: <https://codesandbox.io/s/shakespeare-n3j0c>.

Мы уже несколько раз в коде видели **import**. Например, с помощью него мы импортировали пакет react. Доступ к полученным возможностям мы осуществляем через объект React.

```
import React from "react";
```

Эта форма **import** называется **default import**. Она импортирует то, что было экспортовано с помощью **export**

`default`. А что если вам нужно импортировать, то что было экспортировано с помощью `export` без указания `default`? Тогда используется, так называемый именованный(named) `import`. В нашем коде мы импортировали пакет `react-dom` для доступа к `render`. Вместо этого мы могли использовать именованный `import` для `render`. Тогда вместо экспорта всех возможностей у нас был доступ только к `render`.

```
import React from "react";
import { render } from "react-dom";

import App from "./App";

const rootElement = document.getElementById("root");
render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  rootElement
);
```

В нашем коде два отличия. **Первое отличие** — это именованный импорт.

```
import { render } from "react-dom";
```

В этой форме `import` есть `{}`, которые позволяют указать, что конкретно мы будем импортировать. **Второе отличие** — мы обращаемся к `render` без указания какого-либо объекта.

```
render(
  <React.StrictMode>
    <App />
```

```

    </React.StrictMode>,
    rootElement
);

```

В своем коде, вы будете использовать обе формы import. Ссылка на код проекта: <https://codesandbox.io/s/render-bf0iy>.

В коде наших примеров мы использовали JSX, поэтому у нас была возможность встраивать теги прямо в код JavaScript. Код JSX после обработки превращается в обычный JavaScript. Если у вас есть желание вы можете сразу использовать JavaScript в своём коде вместо JSX.

Рассмотрим эту возможность. Создаём проект под названием [Shakespeare2](#). Приложение будет выводить цитату из произведения Король Лир: «Nothing will come of nothing».

Для начала используем JSX.

```

import React from "react";
import "./styles.css";

export default function App() {
  return <p>Nothing will come of nothing</p>;
}

```

Переводим этот код на чистый JavaScript.

```

import React from "react";

export default function App() {
  return React.createElement("p", null,
    "Nothing will come of nothing");
}

```

Мы используем метод `createElement` для создания элемента. Первый аргумент функции — название тега, второй — свойства (атрибуты) тега, третий — содержимое элемента.

Мы создаём параграф, не указываем никаких свойств, внутри параграфа у нас будет строка «Nothing will come from nothing».

Перейдем к файлу `index.js`. Код ниже использует JSX.

```
import React from "react";
import ReactDOM from "react-dom";

import App from "./App";

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

Переведем его на чистый JavaScript.

```
import React from "react";
import ReactDOM from "react-dom";

import App from "./App";

const rootElement = document.getElementById("root");
ReactDOM.render(React.createElement(App, null),
    rootElement);
```

Мы заменили `<App/>` на вызов `createElement` с передачей параметров. Первый параметр — название создаваемого компонента (`App`), второй аргумент — свойства (`null`, так как мы их не использовали).

Даже на таком небольшом примере видно, что использование JavaScript гораздо менее удобно, чем

JSX. В следующих примерах мы будем использовать только его.

Код проекта доступен по ссылке: <https://codesandbox.io/s/shakespeare2-922wk>.

Вернемся к синтаксису JSX и создадим пример, отображающий текущее время. При старте приложения мы хотим показывать новое время. Для получения информации о времени мы будем использовать объект `Date`. Полученное значение мы будем вставлять внутрь JSX. Каждый раз, когда вам нужно включать вычисляемые значения внутрь JSX, используйте `{}`. Создаём проект и меняем содержимое файла `App.js`.

```
export default function App() {
  /*
   * Получаем текущее время
   * Мы будем его использовать, как значение внутри h2
   * Для вставки значения, которое должно быть
   * динамически заполнено используются {}
  */
  let currentTime = new Date().toLocaleTimeString();

  return (
    <div className="App">
      <h1>Current time is</h1>
      <h2>{currentTime}</h2>
    </div>
  );
}
```

Особое внимание мы обращаем на код:

```
<h2>{currentTime}</h2>
```

`currentTime` — это наша переменная с текущим временем. Для того, чтобы вставить её значение внутрь JSX, мы используем `{}`. Это стандартный механизм React, который вы будете постоянно использовать в своих проектах.

Ссылка на проект: <https://codesandbox.io/s/currenttime-zuc9b>.

Мы уже создали несколько функциональных компонентов. Теперь попробуем создать классовый компонент. Классовые компоненты содержатся в классах. Один класс — один компонент. Класс компонента должен наследоваться от `React.Component`. В классовом компоненте вы обязательно должны реализовать метод `render`. Этот метод возвращает UI вашего компонента. Создадим проект с классовым компонентом. Компонент будет отображать известную цитату Уолта Диснея. Снова создаём проект и погружаемся в `App.js`.

```
import React from "react";
import "./styles.css";
/*
Классовые компоненты — это классы.
Они обязаны наследоваться от React.Component
Внутри класса должен быть реализован метод render
Он обязан возвращать UI нашего компонента
*/
export default class App extends React.Component {
  /*
  Обязательный метод render
  Он должен быть реализован в любом классовом
  компоненте
  */
  render() {
```

```

    return (
      <div className="App">
        <h1>
          The way to get started is to quit
          talking and begin doing
        </h1>
        <h2>Walt Disney</h2>
      </div>
    );
}
}

```

Вместо функции у нас появился класс с обязательным методом **render**. Других серьёзных изменений в нашем коде нет.

Ссылка на проект: <https://codesandbox.io/s/firstclass-component-wmwvr>.

Повторим пример с отображением текущего времени, но будем использовать классовый компонент. Снова создаём проект и меняем содержимое *App.js*.

```

import React from "react";
import "./styles.css";

export default class App extends React.Component {
  render() {
    /*
      В этот раз мы не создавали переменной.
      Мы вставили обновление времени прямо в код JSX.
      Главное не забывать использовать {}
    */
    return (
      <div className="App">
        <h1>Current time is:</h1>
      </div>
    );
  }
}

```

```

        <h2>{new Date().toLocaleTimeString()}</h2>
      </div>
    );
}
}

```

Мы обошлись без дополнительной переменной. Код для получения текущего времени был помещен непосредственно в JSX внутри {}.

Ссылка на проект: <https://codesandbox.io/s/classcurrent-time-6m0yp>.

Даже небольшое знакомство с классовыми компонентами демонстрирует нам некоторую громоздкость их конструкций.

Сейчас мы оперируем одним компонентом, но в реальной жизни количество компонентов всегда больше. Попробуем создать несколько компонентов в новом приложении. У нас будет два компонента. Один компонент будет показывать дату, второй компонент отображает текущее время. Каждый компонент располагается в своей функции. Рассмотрим файл *App.js*:

```

import React from "react";
import "./styles.css";

/*
  Компонент для отображения текущей даты
*/

function CurrentDate() {
  return <h2>{new Date().toLocaleDateString()}</h2>;
}

```

```

/*
    Компонент для отображения текущего времени
*/
function CurrentTime() {
    return <h2>{new Date().toLocaleTimeString()}</h2>;
}

/*
    компонент приложения, собирающий вместе дату
    и время
*/

export default function App() {
    return (
        <div className="App">
            <CurrentDate />
            <CurrentTime />
        </div>
    );
}

```

В нашем проекте три компонента

- **App** — наш уже знакомый компонент приложения;
- **CurrentDate** — компонент даты;
- **CurrentTime** — компонент времени.

Оба новых компонента — это функциональные компоненты. Для включения их в код компонента App мы используем уже привычную форму `<имя компонента/>`.

Ссылка на проект: <https://codesandbox.io/s/dateand-time-7t4ew>.

У вас может возникнуть вопрос: А можем ли убрать этот `div`? Можно ли написать так?

```
export default function App() {
  return (
    <CurrentDate />
    <CurrentTime />
  );
}
```

Написать, конечно, можно 😊. Тем не менее данный код не будет работать, так как мы совершаём ошибку с точки зрения React. Описание ошибки отобразится в браузере справа:

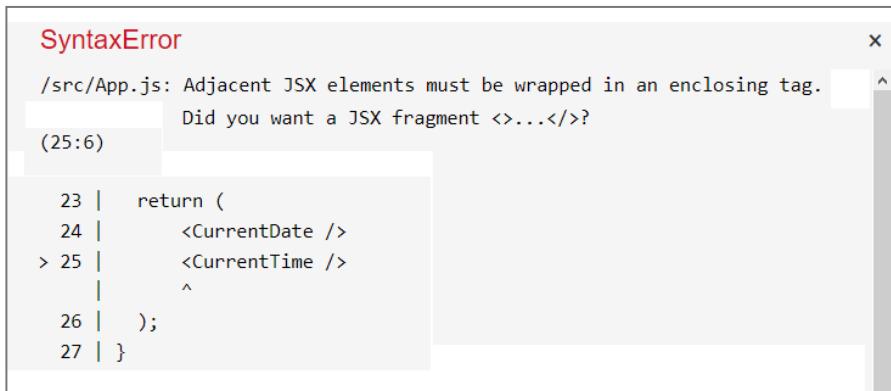


Рисунок 30

При создании кода JSX, вы всегда должны включать элементы в один главный родительский, обобщающий элемент. У нас таким элементом всегда выступал `div`.

Если вы не хотите использовать родительский элемент, порождающий новые узлы в DOM, как это делает `div`, воспользуйтесь `React.Fragment`.

`<React.Fragment>` позволяет вам группировать многочисленные элементы внутри себя. При этом он не бу-

дет создавать новые узлы в DOM. Переделаем наш код с помощью **Fragment**:

```
export default function App() {
  return (
    <React.Fragment>
      <CurrentDate />
      <CurrentTime />
    </React.Fragment>
  );
}
```

У **React.Fragment** есть сокращенная форма `<>`. Вы можете использовать её вместо классической, длинной конструкции:

```
export default function App() {
  return (
    <>
      <CurrentDate />
      <CurrentTime />
    </>
  );
}
```

Код даёт такой же результат, как и **React.Fragment**.

Решим эту же задачу с использованием классовых компонентов.

```
import React from "react";
import "./styles.css";

class CurrentDate extends React.Component {
  render() {
```

```
        return <h2>{new Date().toLocaleDateString()}</h2>;
    }
}

class CurrentTime extends React.Component {
  render() {
    return <h2>{new Date().toLocaleTimeString()}</h2>;
  }
}

export default class App extends React.Component {
  render() {
    return (
      <div className="App">
        <CurrentDate />
        <CurrentTime />
      </div>
    );
  }
}
```

В нашем коде есть три классовых компонента. В каждом из компонентов реализована функция `render`.

Ссылка на код проекта: <https://codesandbox.io/s/class-datetime-kbq8w>.

Мы пока не разносим наши компоненты по разным файлам, так как они достаточно просты и так проще начинать знакомство с React. В следующих уроках мы сделаем это.

В коде наших компонентов мы ещё не использовали обработчики событий. Попробуем создать компонент с обработчиком события. Наш компонент будет реагировать на клик по кнопке. Если событие произошло, мы будем отображать окно сообщения.

```

import React from "react";

function handleClick() {
  alert("You clicked the button");
}

export default function App() {
  return <button onClick={handleClick}>Click me!
    </button>;
}

```

`handleClick` — наша функция обработчик события. В ней мы отображаем информационное окно. Для привязки обработчика события клика к кнопке мы используем атрибут `onClick`. В обычном HTML вы использовали атрибут `onclick`. В `onClick` мы применили уже знакомые `{}`. В них мы указываем имя обработчика. В этом коде вместо обычной функции можно использовать стрелочную функцию.

```

export default function App() {
  // Стрелочная функция в качестве обработчика
  const handleClick = () =>
    alert("Hello from arrow function!");

  return <button onClick={handleClick}>Click me!
    </button>;
}

```

Использование стрелочных функций достаточно типичная практика для современного JavaScript кода.

Ссылка на проект: <https://codesandbox.io/s/button-clickhandler-fpb0n>.

Props

В наших примерах мы не передавали параметров для компонент извне. Может ли нам понадобиться такой механизм? Конечно. Параметры, переданные снаружи позволяют пользователю компонента настроить его перед использованием. Например, если вы создаёте компонент, отображающий информацию о книге, вы можете передать через его свойства название книги, данные об авторе, жанр и так далее.

Как же передать параметры компоненту? Для этого в React существует механизм под названием **Props** (сокращение от **properties** — свойства). При создании объекта компонента вы указываете в его описании набор атрибутов, которым вы задаёте значение. Например:

```
<SomeComponent text=значение color=значение ... />
```

Мы создаём объект компонента **SomeComponent**. При создании мы указываем значения для его атрибутов (свойств) **text** и **color**. Названия его свойств могут быть любыми. Главное, чтобы они имели смысловую нагрузку. Значение может быть задано фиксированное, либо же с помощью техники **{}**.

Если ваш компонент функциональный, вы должны будете указать один аргумент для него. Обычно этот аргумент называют **props** (так принято, имя при желании можно изменить). Внутри этого аргумента будут созданы свойства с именами, указанными при создании компонента.

В нашем случае это будет выглядеть так:

```
function SomeComponent(props) {
  let t = props.text;
  let c = props.color
  .....
}
```

Если у вас классовый компонент, доступ будет через `this`. Выглядеть это будет так:

```
this.props.text
this.props.color
```

Рассмотрим использование `props` на примерах. Начнем с создания функционального компонента, который будет отображать цитату и автора цитаты. Цитата и автор цитаты будут передаваться через `props`.

```
/*
 Мы указали параметр для функционального компонента,
 чтобы была возможность получить значение атрибутов.
 Обычно этот параметр называют props, хотя имя
 может быть любое.
 Параметр заполняется автоматически
 Для доступа к значению атрибута используется
 форма props.имя_атрибута
*/
function Quote(props) {
  return (
    <>
      <h2>{props.text}</h2>
      <h2>{props.author}</h2>
    </>
  );
}
```

```
export default function App() {
  /*
    ' этот символ можно получить, если нажать
    комбинацию клавиш Alt 96.
    96 на цифровой клавиатуре
  */
  let qText = 'Tell me and I forget.
              Teach me and I remember.
              Involve me and I learn.';

  let qAuthor = "Benjamin Franklin";
  /*
    Для передачи данных компоненту используем
    механизм props (свойства)
    Данные компоненту передаем через атрибуты
    Имена атрибутов мы придумали сами
  */

  return (
    <div className="App">
      <Quote text={qText} author={qAuthor} />
    </div>
  );
}
```

В нашем коде компонент называется **Quote**. Мы передаём ему цитату и информацию об авторе через **props**. Мы указали значения для свойств **text** и **author**. Для этого мы использовали {}.

```
<Quote text={qText} author={qAuthor} />
```

Для доступа к значениям внутри кода компонента мы используем **props**.

```

function Quote(props) {
  return (
    <>
      <h2>{props.text}</h2>
      <h2>{props.author}</h2>
    </>
  ) ;
}

```

Результат работы нашего проекта:

Tell me and I forget. Teach me and I remember. Involve me and I learn.

Benjamin Franklin

Рисунок 31

Ссылка на проект: <https://codesandbox.io/s/quoteprops-nf2v4>.

Для закрепления материала создадим ещё один функциональный компонент. Он будет выводить случайное число в указанном нами диапазоне. Для задания начального и конечного значения диапазона мы будем использовать `props`.

```

import React from "react";
function RandomVal(props) {
  /*
    генерируем случайное число в диапазоне от min до max
  */
  let currentVal =
    Math.floor(Math.random() *
      (props.max - props.min + 1)) + props.min;

```

```

        return <h2>{currentVal}</h2>;
    }

export default function App() {
    let start = 1;
    let end = 7;
    /*
        передаём начальные значения через props
    */
    return (
        <>
            <RandomVal min={start} max={end} />
        </>
    );
}

```

И опять же при создании компонента мы указываем значения для свойств `min` и `max`. В коде компонента мы получаем к ним доступ используя параметр `props`.

Ссылка на проект: <https://codesandbox.io/s/randomval-cvpkl>.

Рассмотрим ещё один пример по работе с `props`. На этот раз мы будем обращаться к `props` внутри классового компонента. Наш классовый компонент будет отображать цитату и автора цитаты.

```

import React from "react";
import "./styles.css";

class Quote extends React.Component {
    render() {
        /*
            Для доступа к props внутри класса мы должны
            использовать this
        */
    }
}

```

```

        return (
      <>
        <h2>{this.props.text}</h2>
        <h2>{this.props.author}</h2>
      </>
    );
  }
}

export default function App() {
  const qText = "Stay hungry, stay foolish";
  const qAuthor = "Steve Jobs";
  return (
    <div className="App">
      <Quote text={qText} author={qAuthor} />
    </div>
  );
}

```

Несмотря на то, что мы используем классовый компонент, передача свойств никак не изменилась. Она абсолютна такая же, как и в функциональном компоненте. Отличие в том, как мы получаем доступ к свойствам. Для этого мы используем конструкцию `this.props.имя_свойства`.

```

<>
  <h2>{this.props.text}</h2>
  <h2>{this.props.author}</h2>
</>

```

Ссылка на проект: <https://codesandbox.io/s/quoteclass-e53yy>

Для закрепления знаний, создайте свой собственный компонент, который будет использовать `props`.

Домашнее задание

1. Создайте и запустите приложение React, выводящее краткую информацию о вас в браузер. Например: ФИО, контактный телефон, электронный адрес.
При разработке нужно использовать функциональные компоненты и синтаксис JSX.
2. Создайте и запустите приложение React, выводящее краткую информацию о вашем городе в браузер. Например: название города, название страны, год основания, несколько фотографий достопримечательностей вашего города.
При разработке нужно использовать функциональные компоненты и синтаксис JSX.
3. Создайте и запустите приложение React, выводящее информацию о кулинарном рецепте в браузер. Например: название рецепта, составляющие рецепта (ингредиенты и их количество), последовательность приготовления, фотография готового блюда.
При разработке нужно использовать классовые компоненты и синтаксис JSX.
4. Используя рендеринг элементов создайте приложение, отображающее библиографию Шекспира. Создайте несколько компонентов для реализации разных частей приложения. Например: компонент для отображения общей информации о Шекспире, компонент для отображения информации о конкретном произведении.

5. Создайте приложение «Любимый кинофильм». Оно будет содержать информацию о вашем любимом фильме: название фильма, ФИО режиссера, год выпуска, киностудия, постер и т.д. Обязательно используйте функциональные компоненты и `props`
6. Создайте приложение «Персональная страница». Оно будет содержать информацию о вас (ФИО, телефон, email, город проживания, опыт работы, навыки, фотографию и т.д.). Обязательно используйте классовые компоненты и `props`.