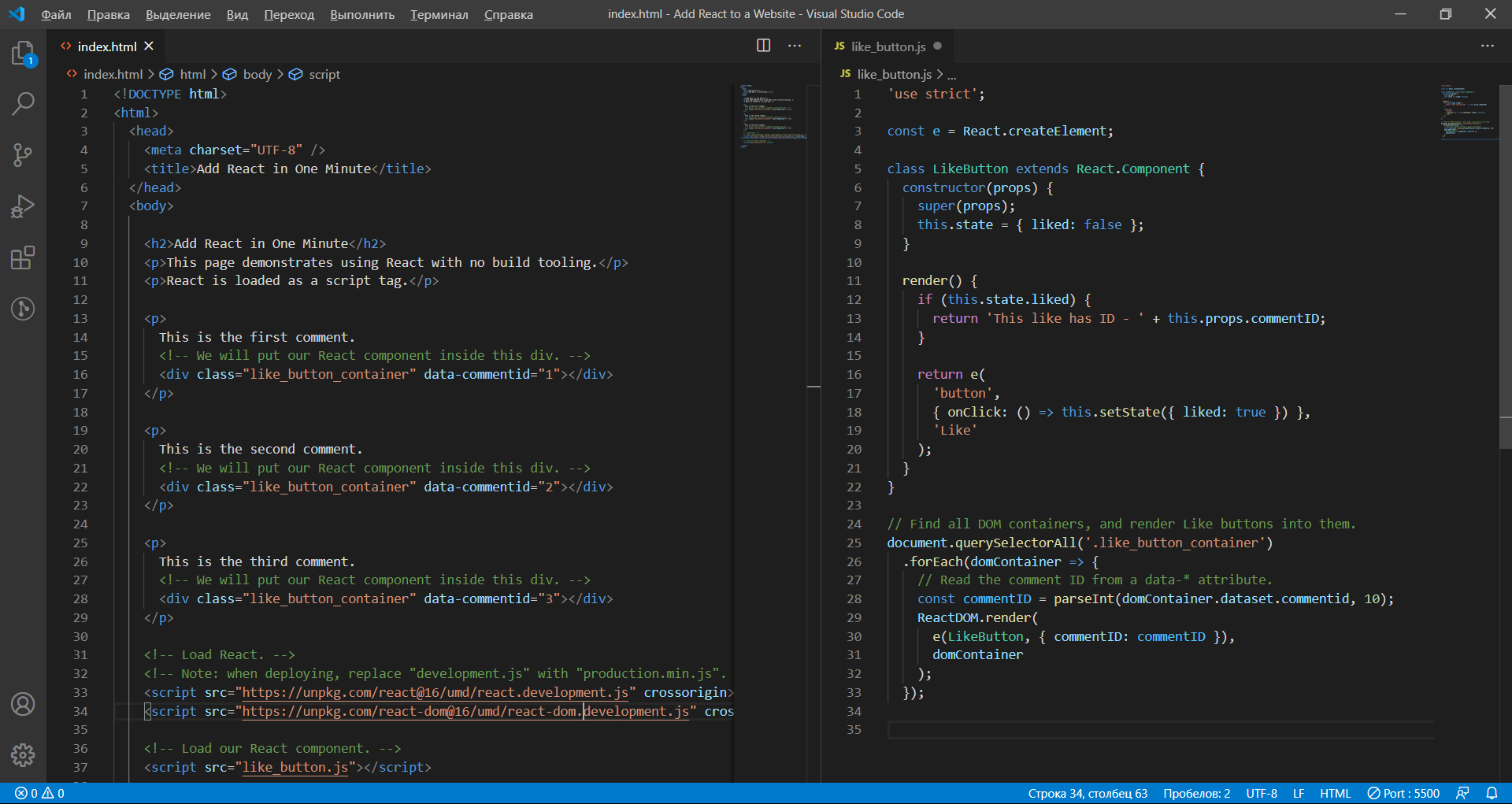
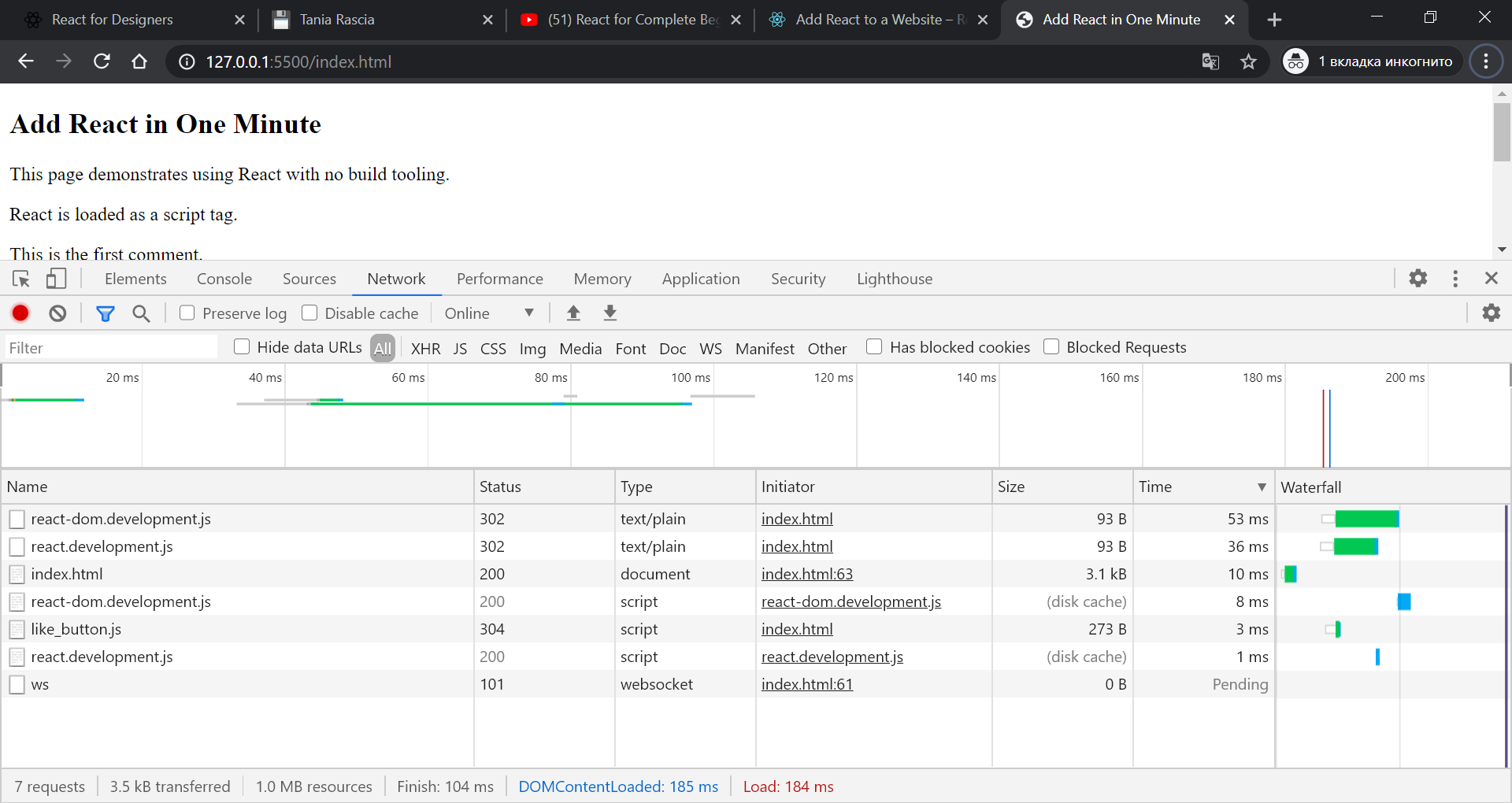
# **22.07.2020 4,5 часа Отчет: прошел главу INSTALLATION, впервые установил React через CDN, провел тестирование на загрузку файлов CDN, эксперимент описал ниже. Разобрался с механизмом CORS, crossorigin. Начал изучать главу MAIN CONCEPTS, повторил JSX, рендеринг элементов.**

Начал изучать React с помощью ресурса reactjs.org и taniarascia.com путем чтения и практики. 

Подключил файлы React по CDN. Сначала с помощью ссылок:

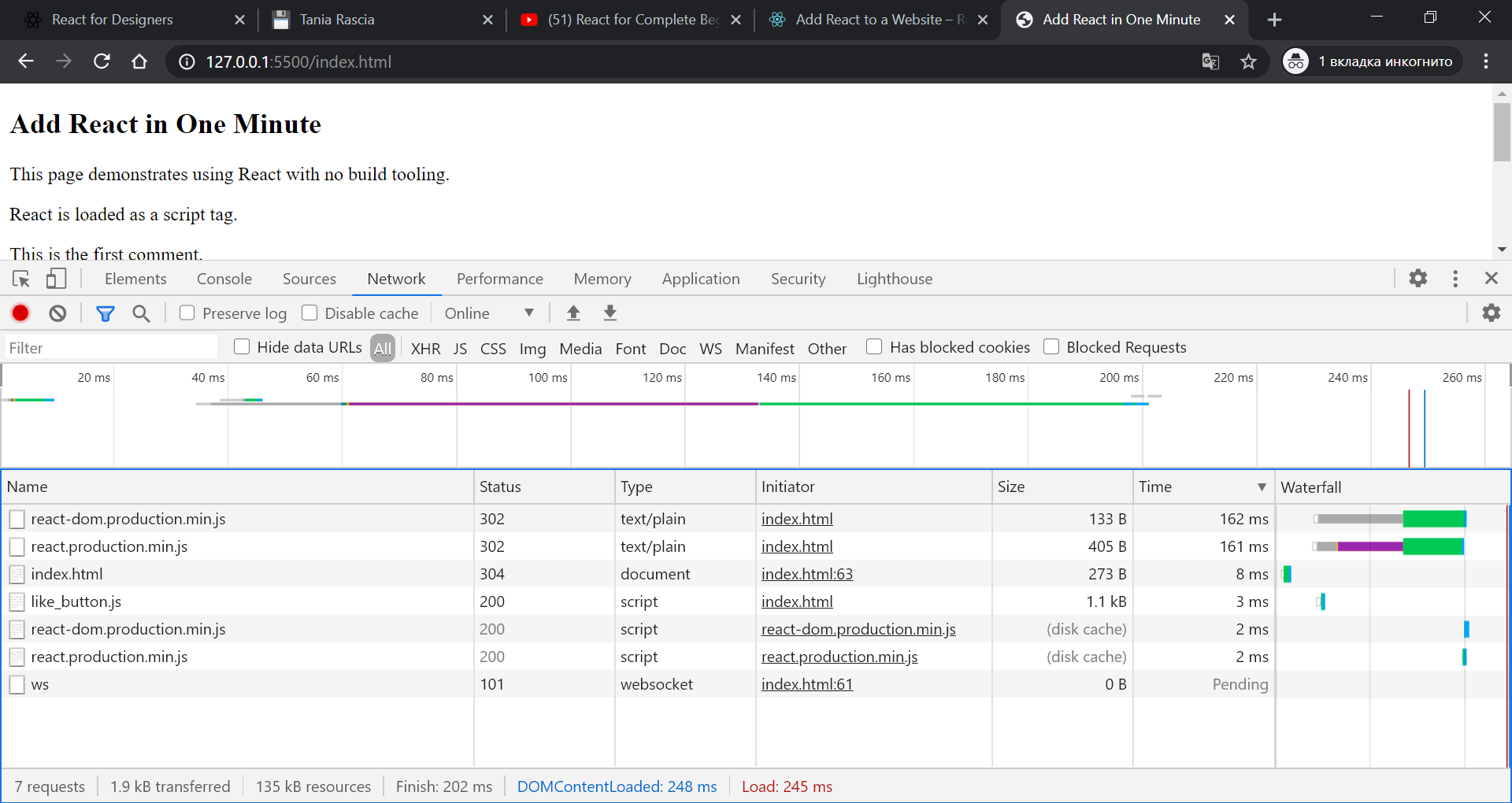
<https://unpkg.com/react@16/umd/react.development.js>  
<https://unpkg.com/react-dom@16/umd/react-dom.development.js>

Обращаем внимание на следующие данные:  
React-dom.development.js – время загрузки: 53 миллисекунды, размер: 93 байт  
React.development.js – время загрузки: 36 миллисекунд, размер 93 байт

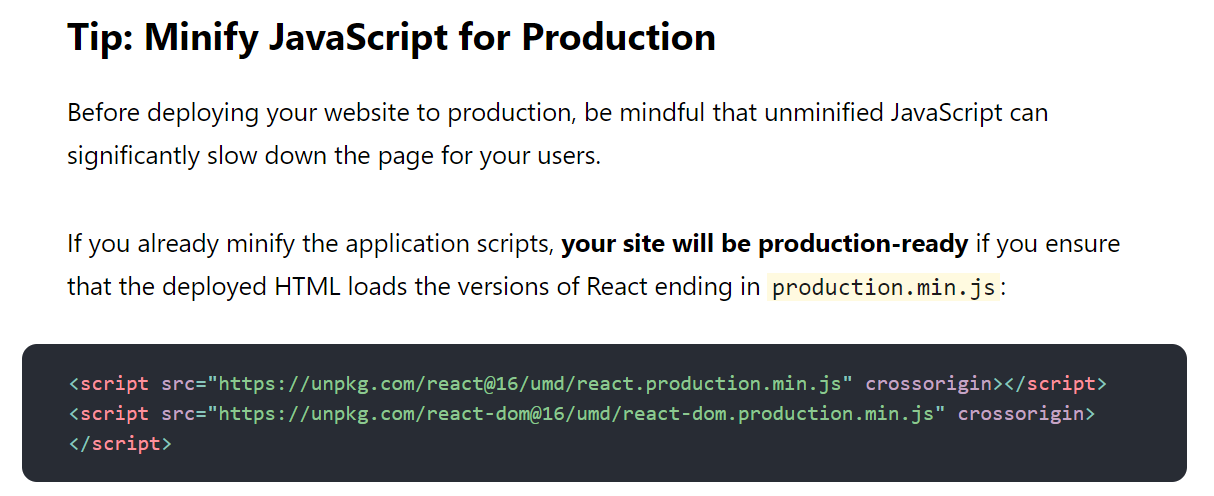


Затем с помощью ссылок:  
<https://unpkg.com/react@16/umd/react.production.min.js>  
<https://unpkg.com/react-dom@16/umd/react-dom.production.min.js>

Обращаем внимание на следующие данные:  
React-dom.production.min.js – время: 162 миллисекунды, размер: 133 байт  
React.production.min.js – время: 161 миллисекунда, размер 405 байт

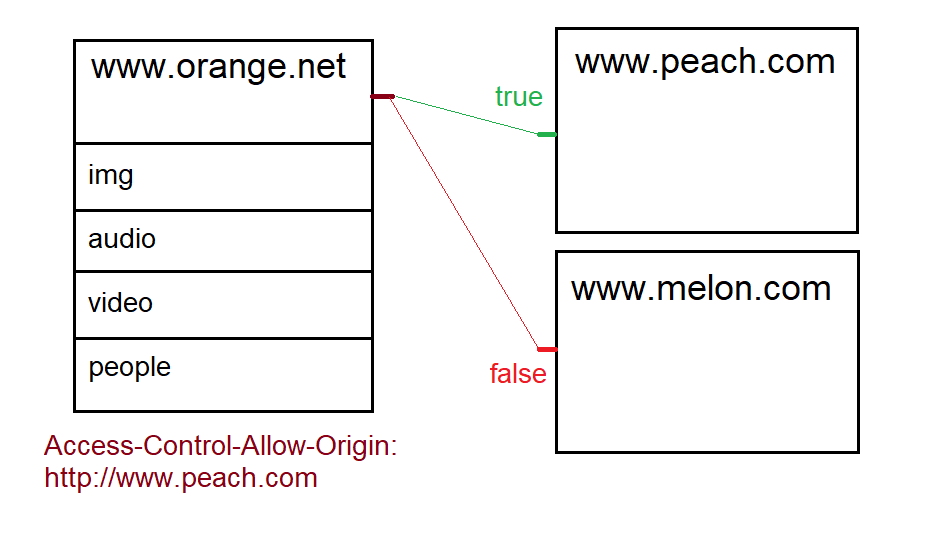
­­

Несмотря на то, что в документации для уменьшения производительности разработчики дают совет подключать файлы с расширением заканчивающимся на production.min.js, оказалось, что файлы с таким расширением весят больше и загружаются дольше.



CORS – Cross Origin Resource Sharing (Совместное использование ресурсов первоисточника).  
Это механизм, который использует дополнительные HTTP заголовки, чтобы другой домен имел доступ к ресурсам текущего домена.

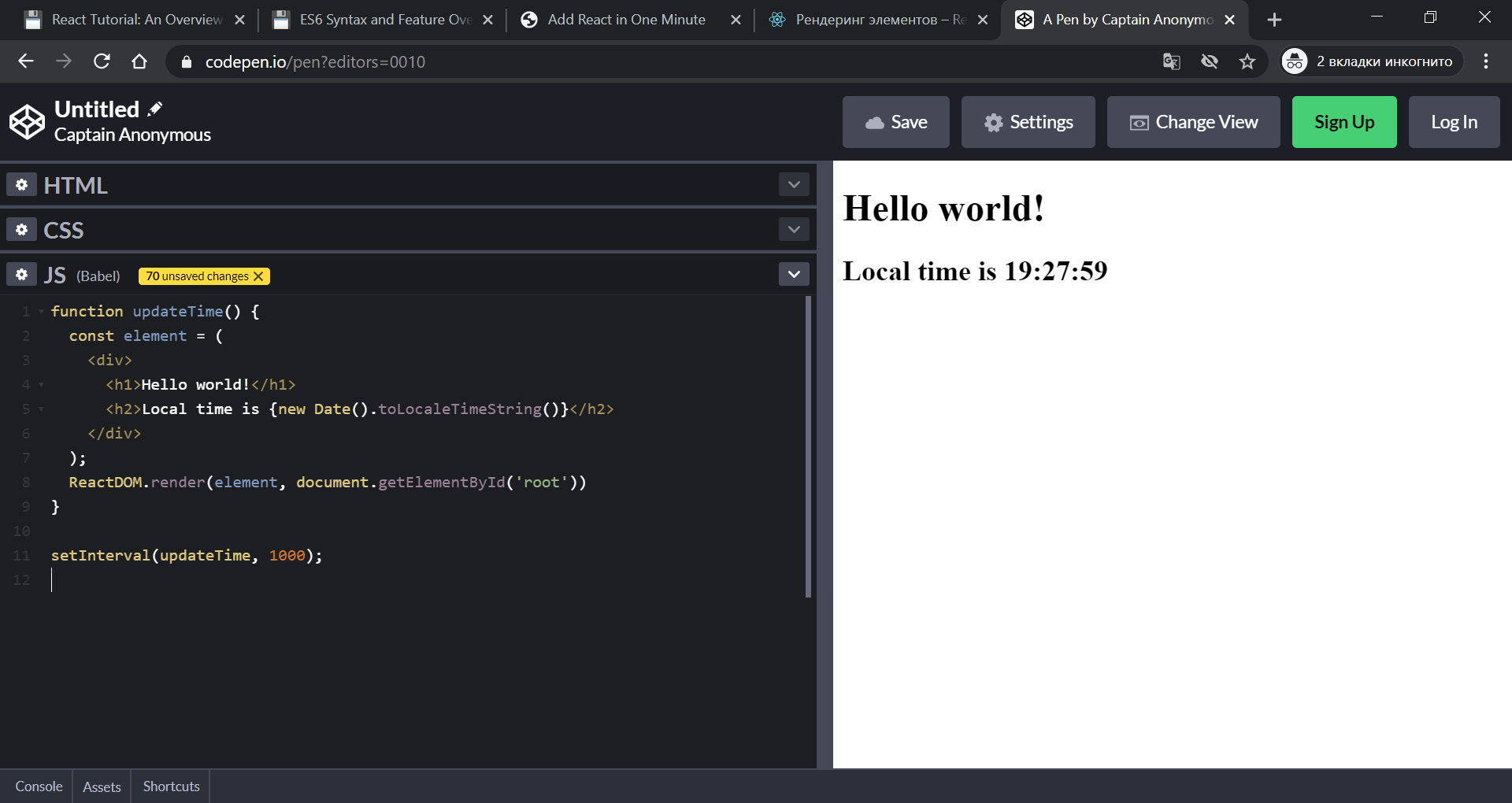
На данном примере можно понять, что peach.com имеет доступ к данным orange.net, а melon.com не имеет, потому что orange.net с помощью механизма CORS дал доступ только peach.com



На самом деле Access-Control-Allow-Origin не единственный заголовок CORS, их около 10.  
CORS встроен во многие API контейнеры, например XMLHttpRequest и Fetch.

В HTML5 некоторые теги поддерживают механизм CORS, например img или video. Соответственно имеют атрибут crossorigin, который может принимать следующие значения:  
-anonymous: CORS запросы от этого элемента не будут передавать учетные данные  
-use-credentials: CORS запросы от этого элемента будут передавать учетные данные

Если атрибут crossorigin отсутствует, то CORS не используется вообще  
Если crossorigin прописан, но не принимает никакое значение, то по умолчанию это anonymous  
  
Под учетными данными подразумевают: cookie, client-side SSL сертификаты, HTTP аутентификацию

Переписал пример в CodePen. Разобрался как работает рендеринг и Virtual DOM 

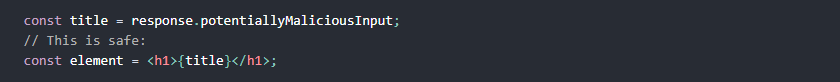
# **23.07.2020 6 часов Отчет: начал изучать компоненты путем чтения официальной документации и практики, практиковал работу с функциональными и классовыми компонентами, разобрался в чем отличие между state и props. Повторил жизненные циклы компоненты. Познакомился с setState, обработкой событий React**

Компонента – функция, которая может принимать данные, возвращать JSX разметку

JSX – это расширение языка JavaScript, с помощью которого можно создавать компоненты

Принцип React – не помещать разметку и логику в разные файлы, а создавать компоненты, которые содержать и логику и разметку

JSX защищен от атак, основанные на инъекции кода (один из самых популярных видов атак)

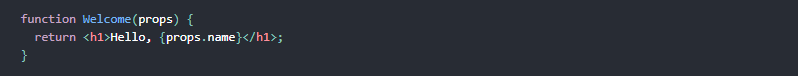
Если учитывать, что title – это текст, введенный пользователем, то данный код защищен от инъекции, так как в JSX все обрабатывается в строки  


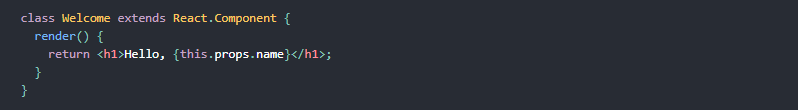
На верхней картинке мы видим современный код, созданный с помощью JSX разметки  
На средней картинке код, который получен в результате компиляции Babel  
На нижней картинке, мы видим, что на самом деле создает Babel «под капотом»



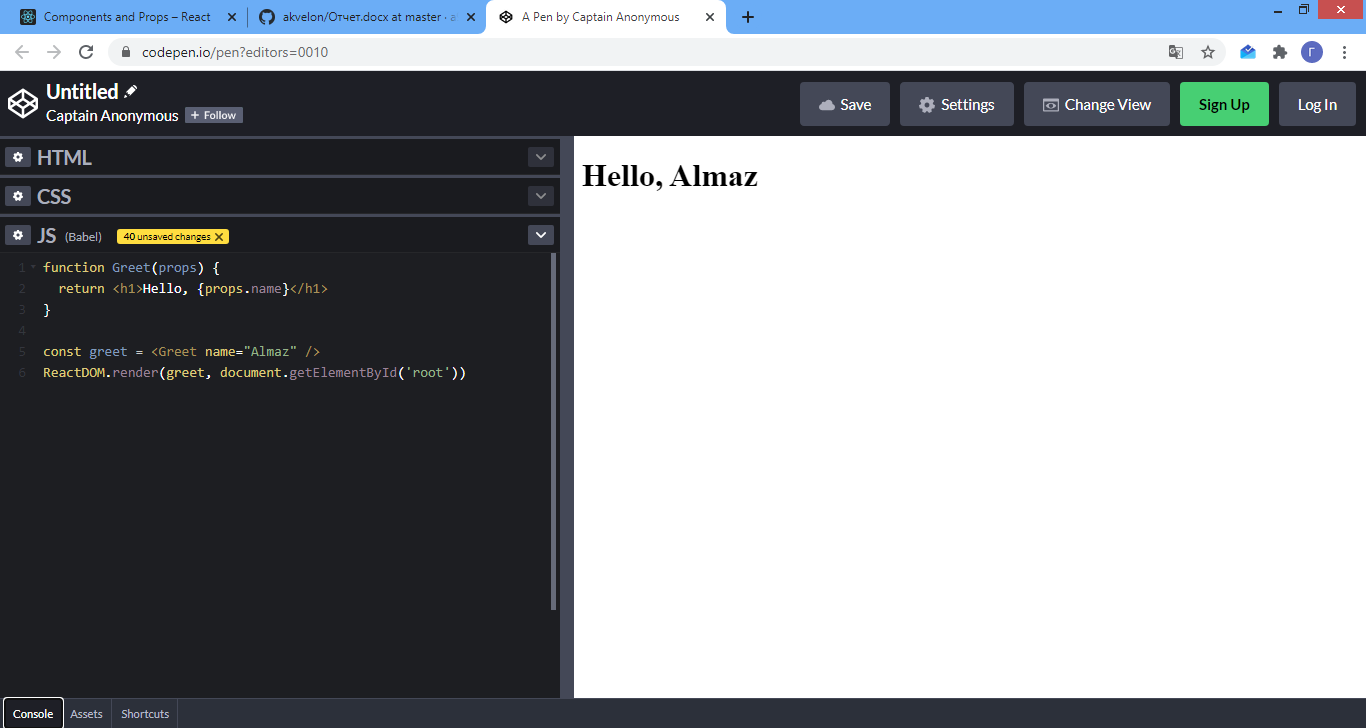
Компонента – ключевой элемент в React, с помощью которых создаются сайты и приложения  
Компоненты бывают двух видов: классовые и функциональные.

Компоненту принято писать с заглавной буквы  
Компоненту можно использовать сколько угодно, передавать в нее разные значения

Пример функциональной компоненты:  


Пример классовой компоненты:  


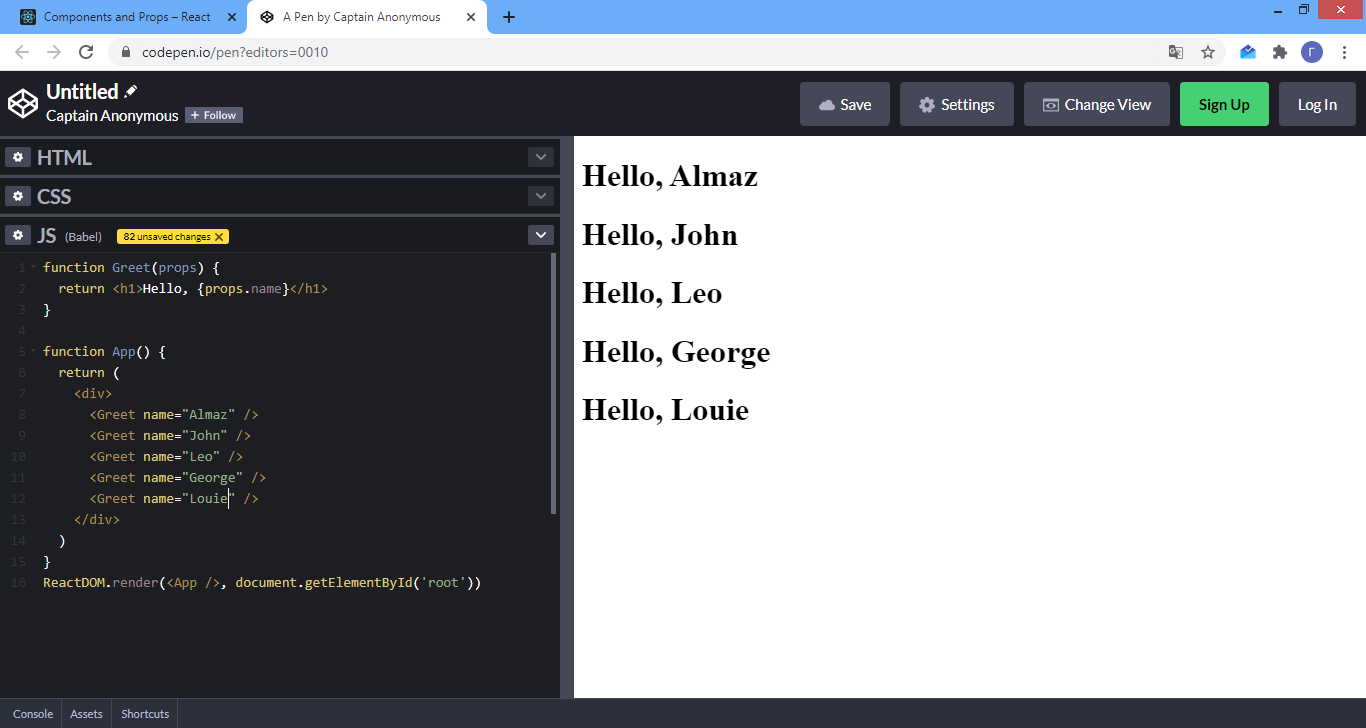
В данном примере, мы передали данные в компоненту, с помощью props



Какой алгоритм был в этом коде?

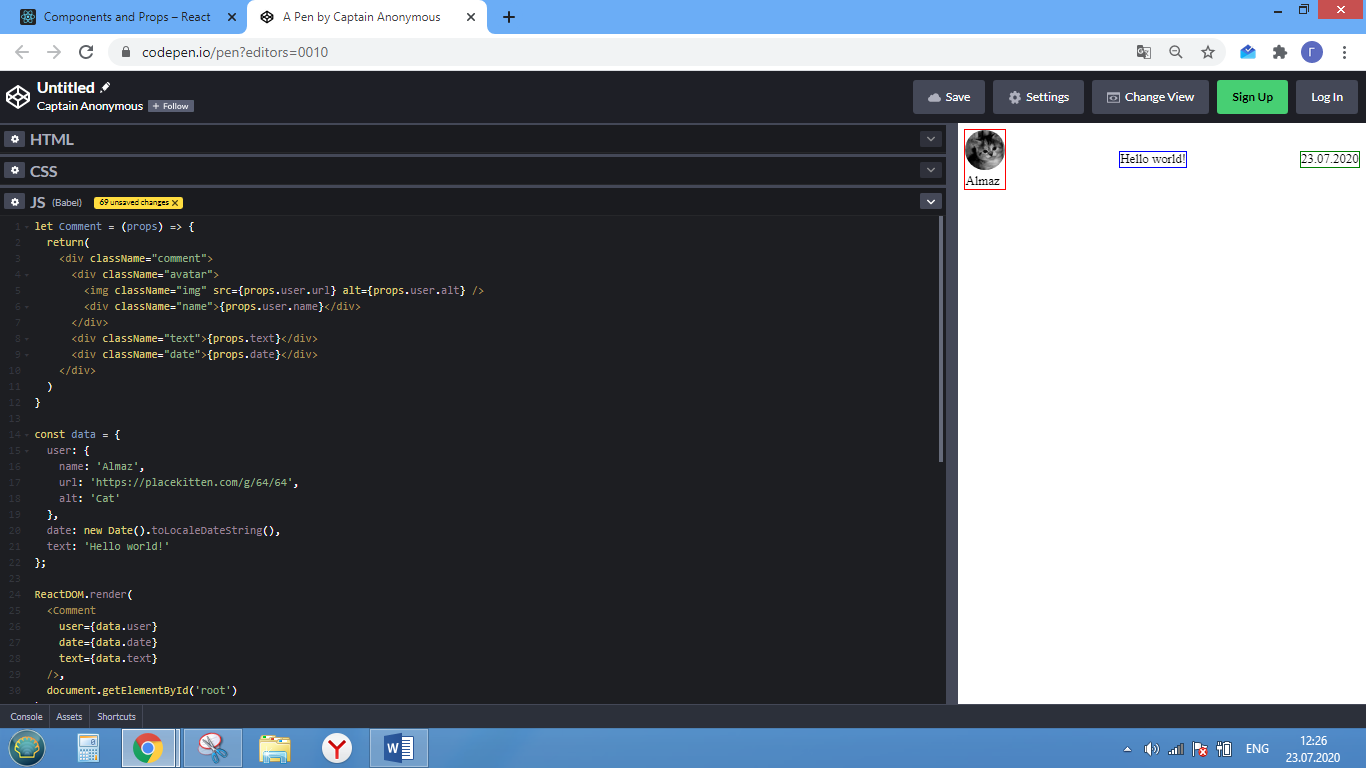
1. Вызывается функция ReactDOM.render, которая принимает два параметра:   
- переменную greet, в которой содержится компонента с параметром name  
- адрес, который имеет ID «root» в дереве документа  
Если быть точнее, то с помощью функции ReactDOM.render мы рендерим greet в root

2. React вызывает функцию Greet, передав в нее параметр name  
3. Компонента Greet возвращает элемент <h1>Hello, Almaz</h1>  
4. React DOM делает минимальные изменения, чтобы обновить DOM, передав элемент

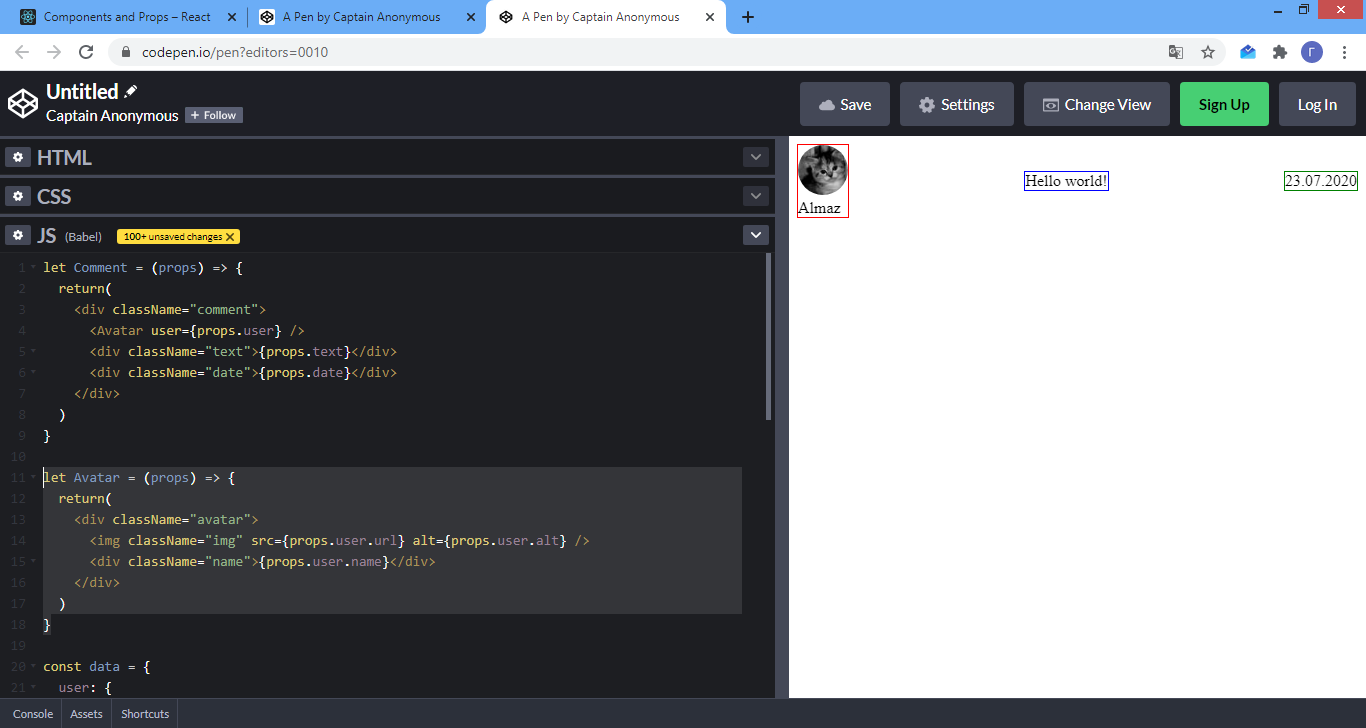
Теперь отрендерим компоненту Greet несколько раз, передав в нее другие значения:  


В данном примере мы создали компоненту App, которую отдали на рендеринг в функцию ReactDOM.render

Также на данном примере можно сделать вывод, что компоненты можно вкладывать друг в друга

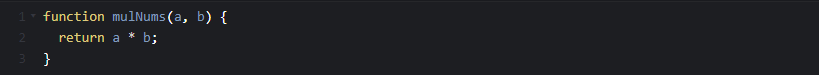


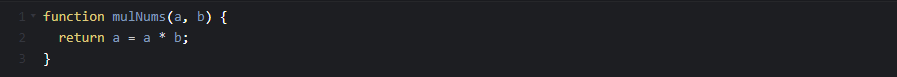
В данном примере мы разбили компоненту на 3 части (каждую обвел в рамку).  
Красная рамка отвечает за картинку, атрибут alt, имя пользователя  
Синяя рамка отвечает за текст комментария  
Зеленая рамка отвечает за дату  
  
Также, в скриншоте ниже, я разбил компоненту Comment на компоненту Avatar и две других части. Таким образом, мы можем дробить компоненты до самого конца, отделяя каждую логическую часть друг от друга.



Props – это объект, который всегда передается в компоненту как параметр. Даже если мы не прописываем ключевое слово props, то он передастся автоматически, но не будет содержать никаких данных.

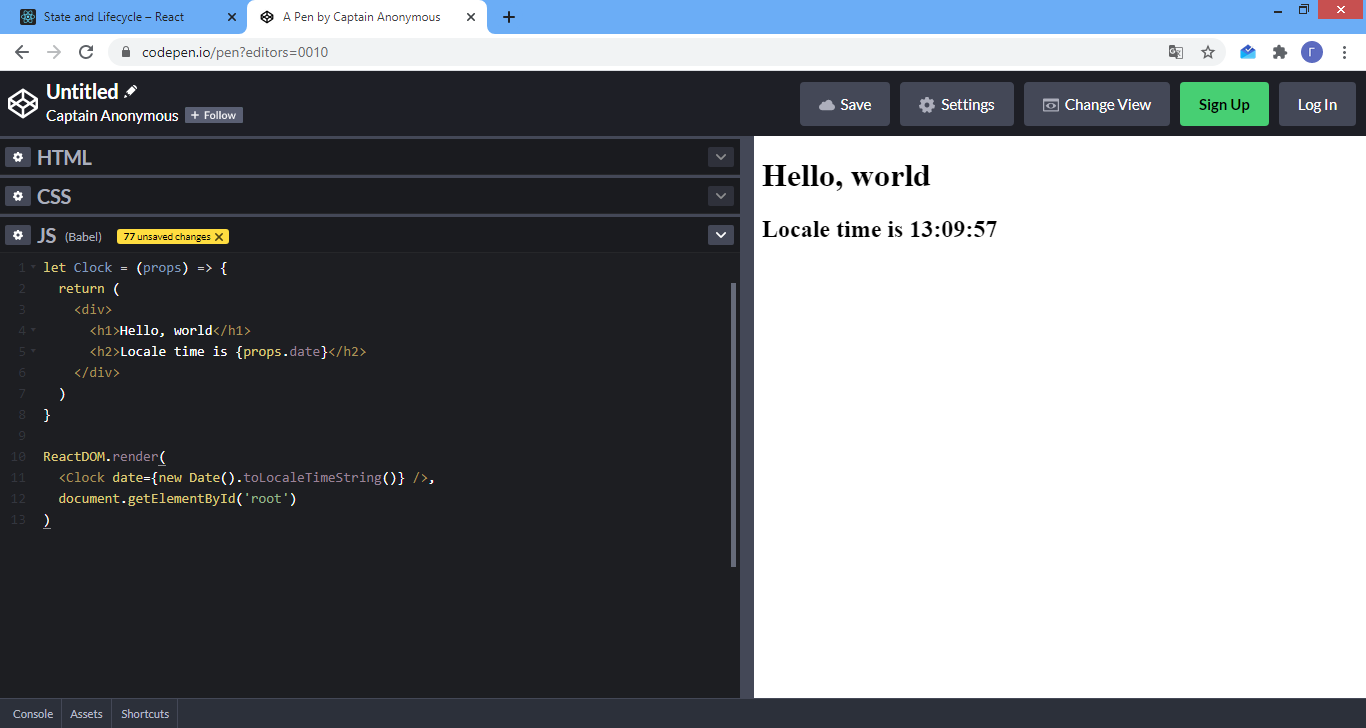
Таким образом, props нам нужен для того, чтобы мы могли передавать данные в компоненту.  
  
Компонента никогда не должна изменять значение props. Компонента должна работать, как «чистая» функция, а не наоборот.

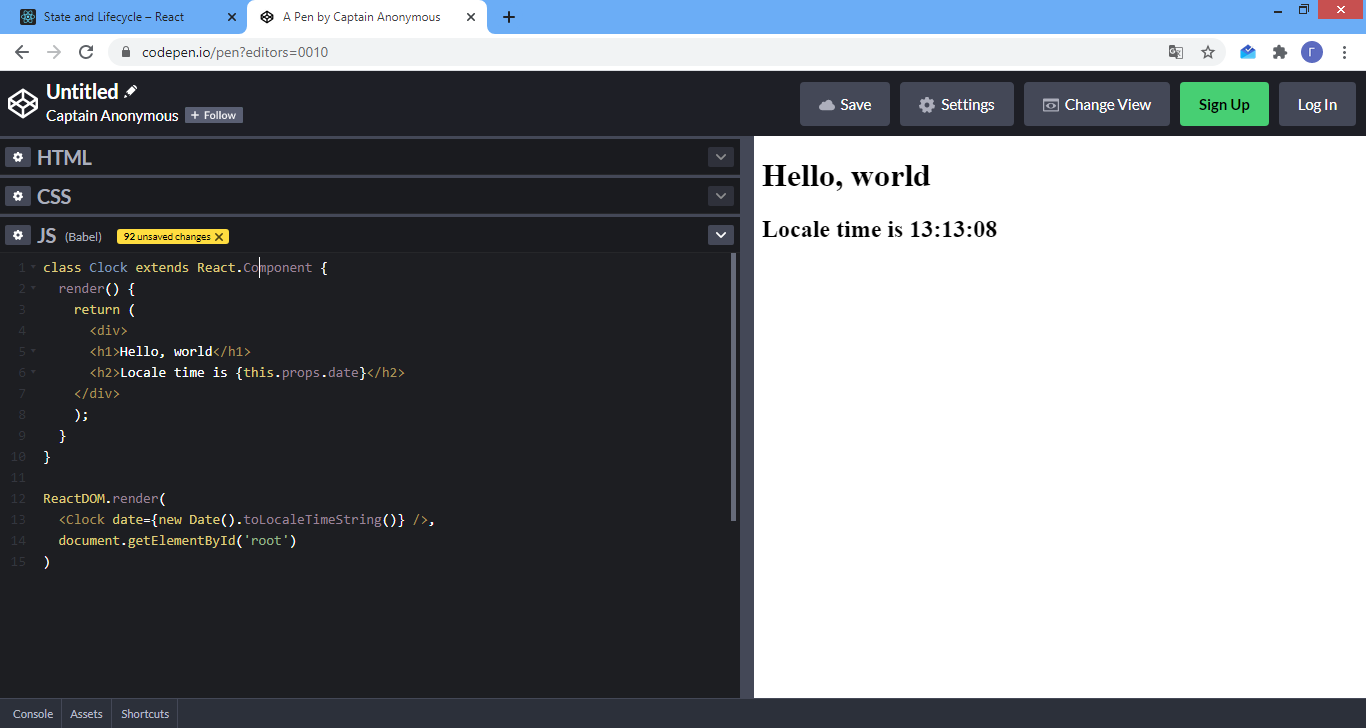
Пример «чистой» функции:  


Пример «нечистой» функции:  


Представим, что у нас есть функциональная компонента, на ее основании создадим классовую компоненту по следующему алгоритму:

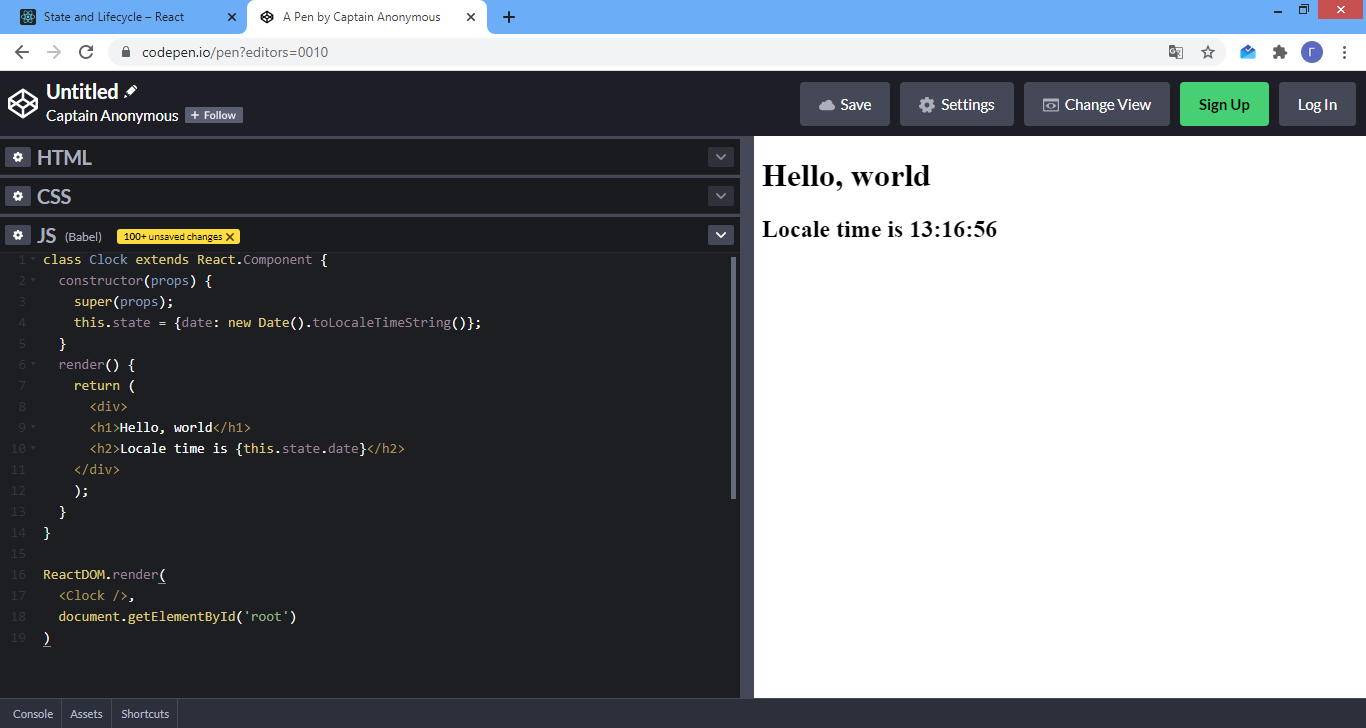
1. Создадим класс, по синтаксису как в ES6, который наследуется от React.component  
2. Добавим в самом начале одиночный метод render()  
3. Скопируем return и все, что находится внутри него  
4. Если есть props то заменим на this.props

Таким образом из функциональной компоненты:  


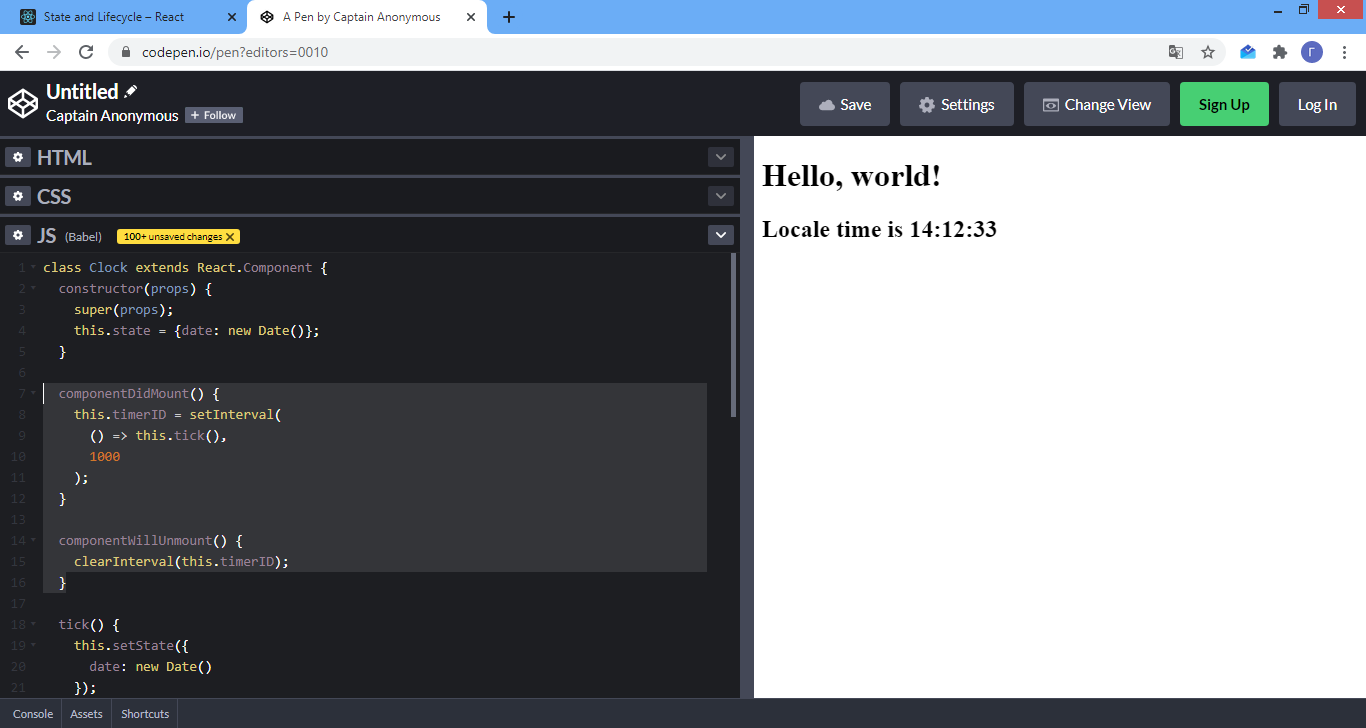
Получим классовую компоненту:  


Также в 4 шага мы можем заменить передачу данных в нашей компоненте с props на state:

1. Заменить this.props.date на this.state.date  
2. Добавить классовый конструктор передав в него значение props  
3. Добавить ключевую строку, которая генерирует текущее время:  
 this.state = {date: new Date().toLocaleTimeString()}  
4. Удалить в ReactDOM.render при рендеринге компоненты Clock данные

В итоге получится:   


В приложении, где содержится много компонент, очень важно их освобождать, когда они закончили свою работу.  
  
Когда происходит рендеринг элемента в DOM, то этот процесс называется Mounting  
Когда DOM-узел, созданный элементом, удаляется – происходит процесс Unmounting

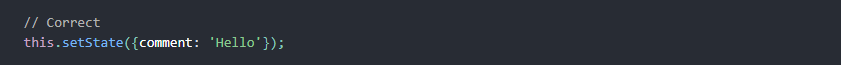
Добавим componentDidMount и componentWillUnmount  


Теперь часы обновляются каждую секунду. Часы работают по следующему алгоритму:

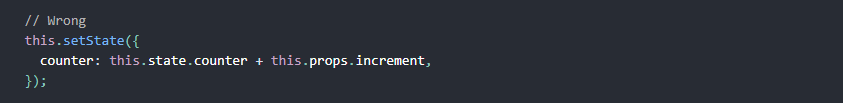
1. Как обычно все начинается с ReactDOM.render. Когда он видит, что внутри него есть компонента Clock, он вызывает конструктор компоненты  
2. React вызывает метод render() у компоненты Clock. Соответственно React узнает, что нужно отобразить и обновляет DOM с минимальными изменениями  
3. После того, как Clock вставлен в DOM, React вызывает метод жизненного цикла componentDidMount, внутри которого компонента Clock устанавливает интервал в 1 секунду для вызова функции tick()

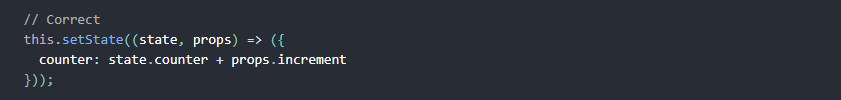
4. Таймер вызывает tick() каждую секунду, внутри функции tick() с помощью setState мы обновляем состояние компоненты. Так как функция вызывается каждую секунду, то значение this.state.date в методе render содержит новое значение, React обновляет DOM с минимальными изменениями  
5. Если компонента Clock когда либо удалится из DOM, то React вызовет метод жизненного цикла componentWillUnmount и сбросит таймер

Что нужно знать о setState?

1. Нельзя изменять состояние напрямую  
Для того, чтобы отрендерить компоненту снова:  


Единственное место, где можно писать this.state это конструктор

2. Обновления состояния могут быть асинхронными  
React может сгруппировать несколько вызовов setState() в одно обновление для улучшения производительности. Так как this.props и this.state могут обновляться асинхронно, то не стоит полагаться на их текущее состояние, чтобы обновлить следующее  


Более правильное решение в данной ситуации – передавать в setState не объект, а функцию. Причем функция получит первый параметр state отвечает за предыдущее состояние, а значение props во время обновления как второй параметр  


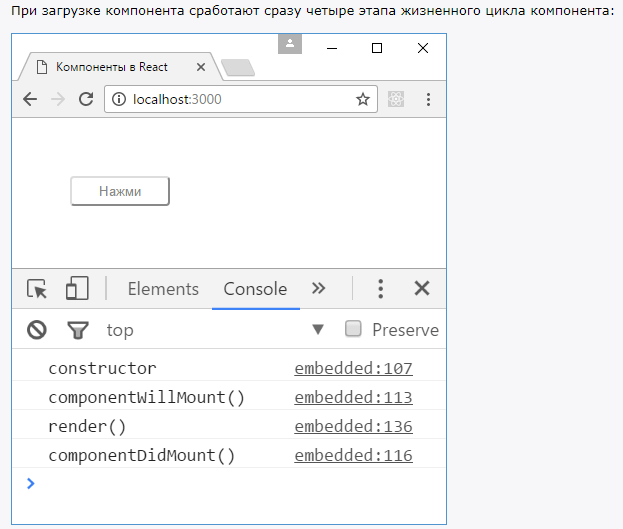
3. Обновленные состояния объединяются  
Когда мы вызываем setState и передаем новое состояние, то React объединит это новое состояние с текущим

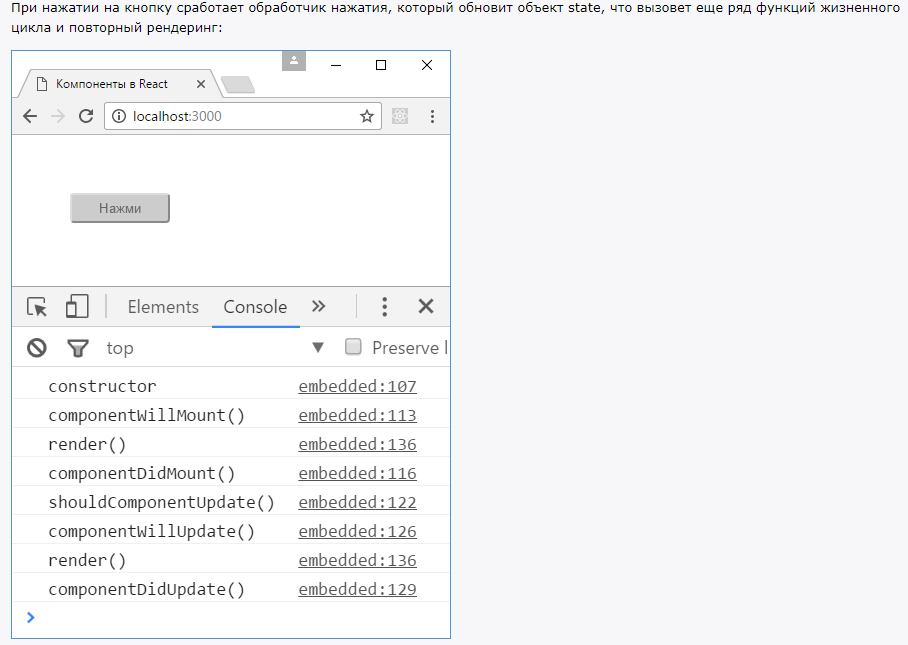
Состояние доступно только для компонента и скрыто от других компонент  
Однако данные из этого состояния можно передавать в другие компоненты с помощью props, но только в те компоненты которые находятся «ниже», чем данная.

Жизненные циклы компоненты:  
1. Constructor(props) – конструктор, в котором происходит начальная инициализация компонента  
2. componentWillMount() – вызывается перед рендерингом элемента  
3. Render() – рендеринг элемента  
4. componentDidMount() – вызывается после рендеринга элемента. Здесь можно выполнять запросы к удаленным ресурсам  
5. componentWillUnmount() – вызывается перед удалением компонента из DOM

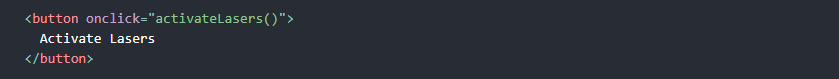
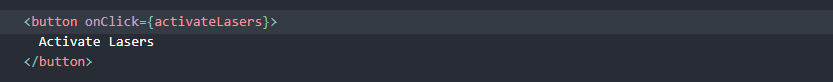
**State vs props**

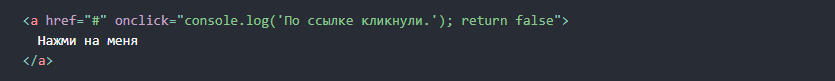
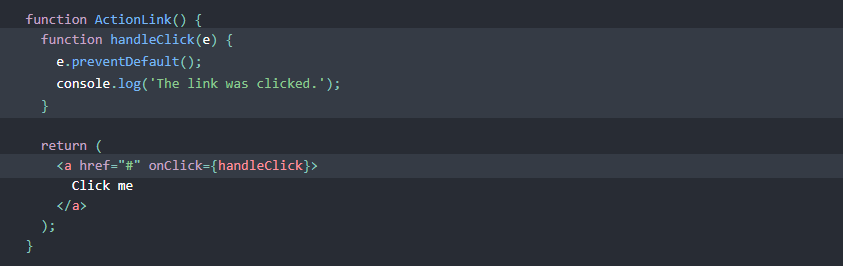
И state и props являются объектами, способами передачи данных в компоненту, но  
state – состояние, объект внутри компоненты  
props – объект переданный компоненте из родительской компоненты



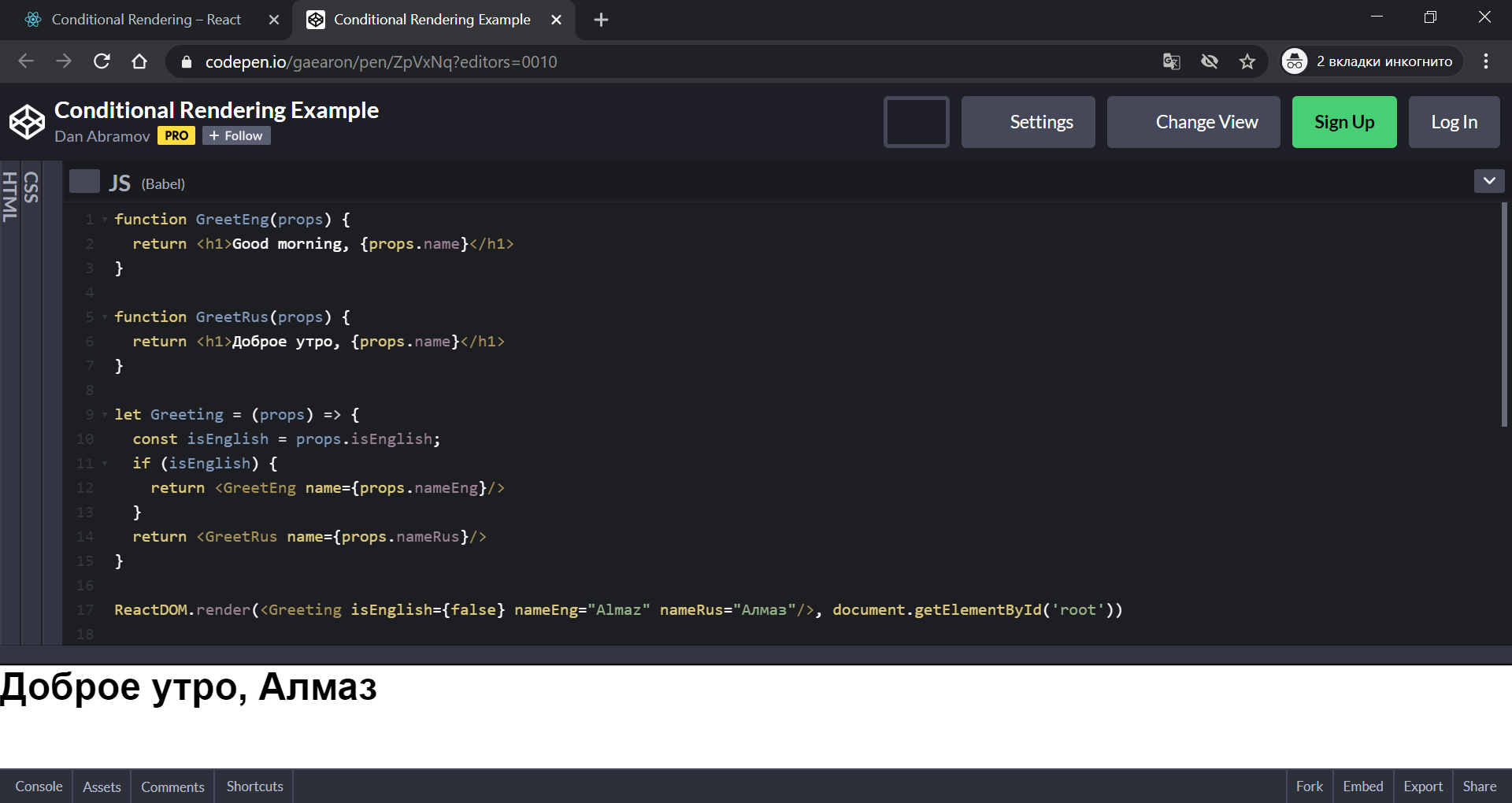


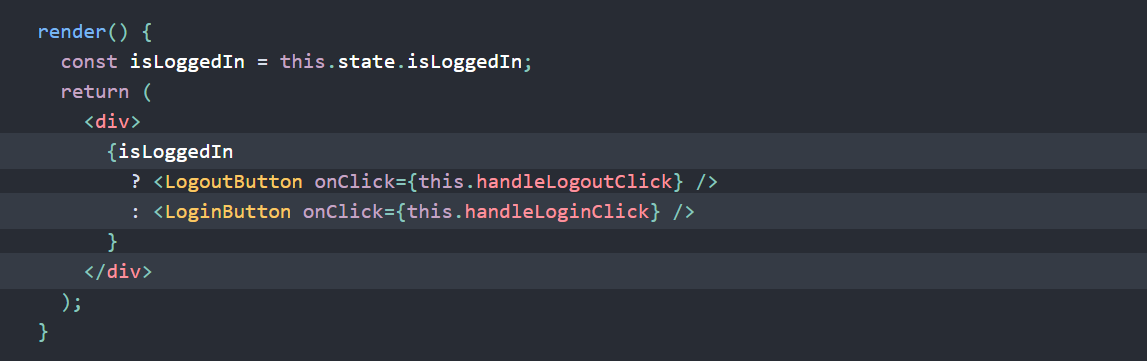
**Обработка событий**

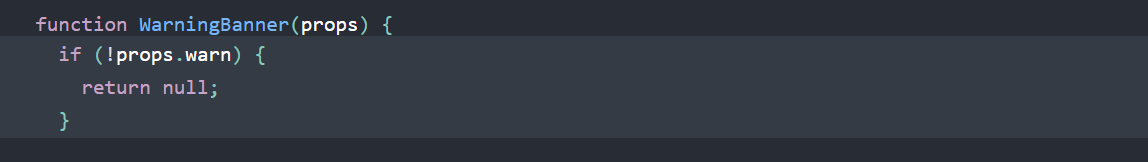
Обработка событий в React очень похожа на обработку событий в DOM. Но есть несколько отличий:  
1. События в React имеют camelCase (onClick, onMouseOver, onMouseEnter)  
2. В JSX мы передаем в функцию не строку, а обработчик событий  
  


3. В React нельзя предотвратить обработчик событий по умолчанию, вернув false  
  


# **24.07.2020 4 часа Отчет: прошел условный рендеринг, списки и ключи, формы, начал изучать подъем состояния Условный рендеринг** В React, как и в JS, существуют условные выражения. Большинство вопросов решают с помощью оператора if

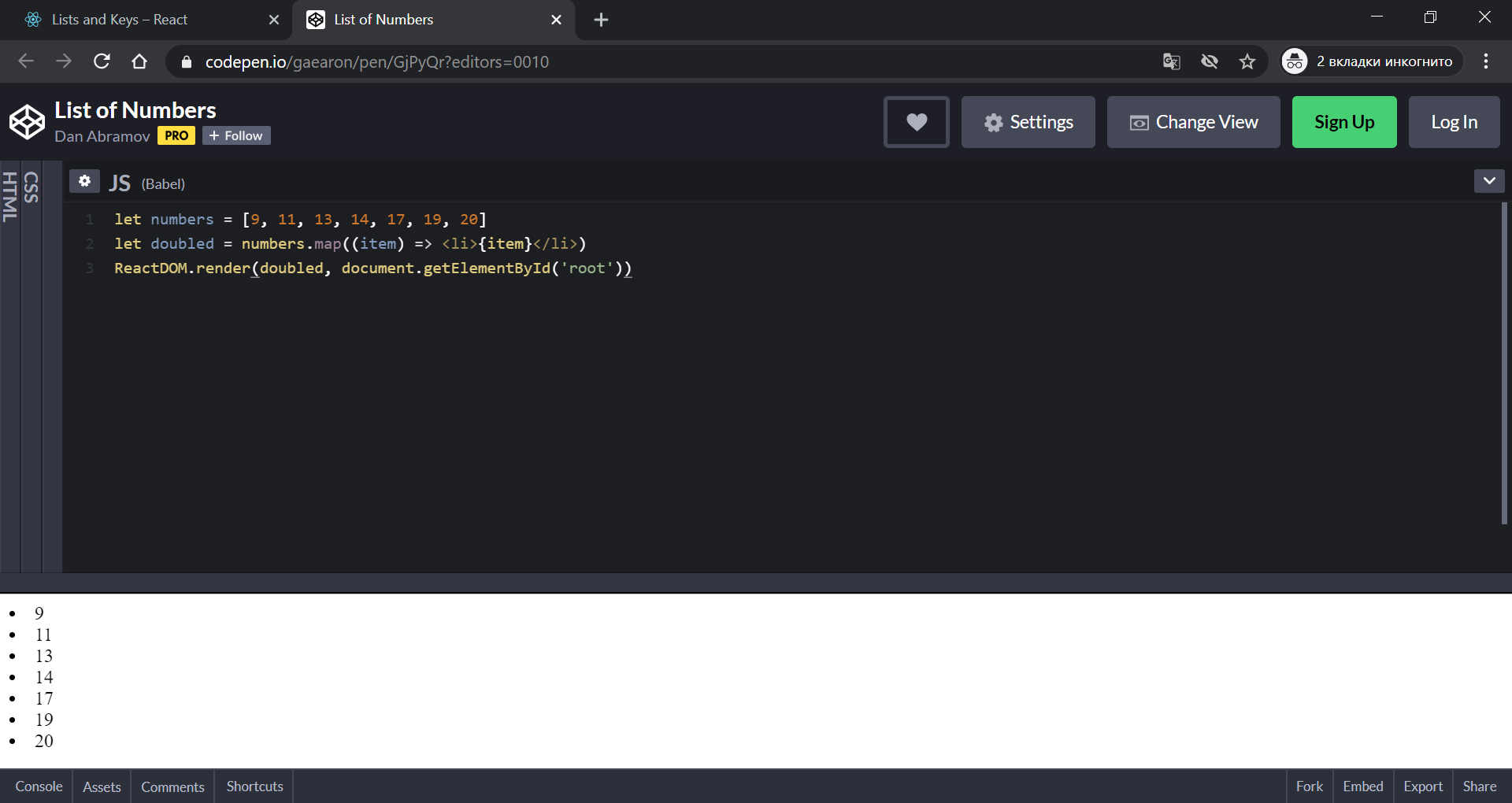


То же самое можно было бы записать строчным синтаксисом if else  


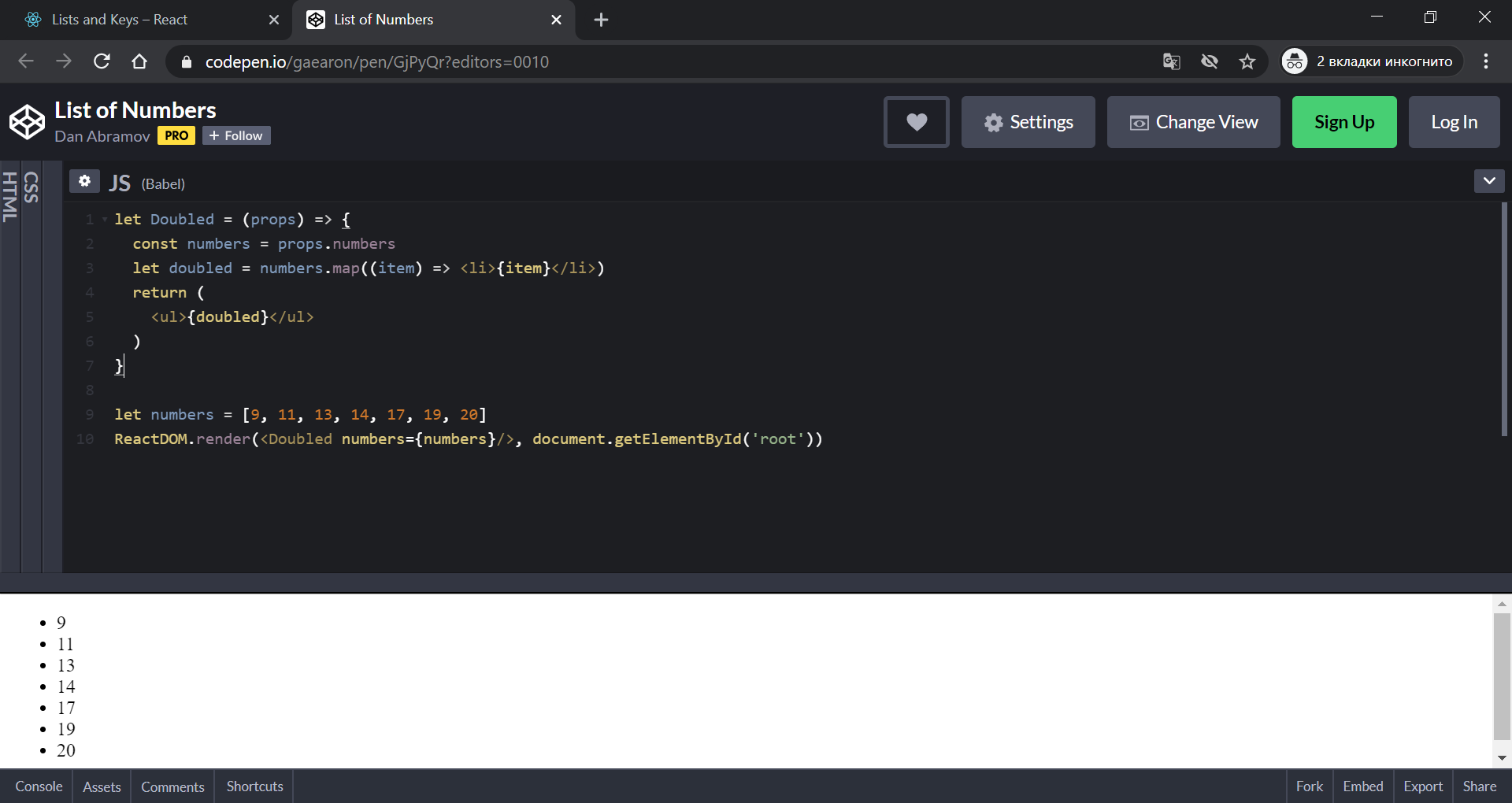
Также с помощью условного оператора if мы можем предотвращать рендеринг компоненты  


**Списки и ключи**

Отрендерим массив, измененный с помощью map



То же самое можно сделать через компоненту



Ключи в React нужны для того, чтобы отслеживать элементы массива. Их рекомендуется указывать для того, чтобы React мог сопоставлять элементы массива с течением времени.  
Лучший способ выбрать ключ – это использовать строку, которая будет отличать элементы списка.  
Когда нет ID, в крайнем случае можно использовать индексы массива для ключей, однако делать это не рекомендуется, потому что может поменяться порядок элементов. Еще это может негативно отразиться на производительности. Если в React не установить ключ для элемента, то React по умолчанию будет использовать индексы массивов для ключей.

Ключи должны быть уникальными, только среди своего блока. Им необязательно быть глобальными. То есть один и тот же ключ может правильно работать в двух разных массивах.  
Ключи служат подсказками для React, они никогда не передаются явно в компонеты. Если есть необходимость их передать, то сделать это можно через props, но не через свойство key

**Формы**

В React HTML формы ведут себя отлично от элементов DOM, так как у элементов формы есть внутреннее состояние. К примеру: в input мы можем ввести имя  
  
Управляемые компоненты  
В HTML элементы формы (input, textarea, select и т.д.) управляют своим состоянием и обновляют его когда пользователь вводит какие-то данные. В React обычно состояние является неизменным, обновляется только через setState()  
  
Мы можем скомбинировать оба подхода и сделать состояние React-компонента «единственным источником правды». Тогда React-компонент будет рендерить форму и контролировать её поведение в ответ на пользовательский ввод. Значение элемента формы input в этом случае будет контролировать React, а сам элемент будет называться «управляемый компонент»

**Подъем состояния**

Часто, несколько компонент должны обрабатывать и отражать UI на основе одних и тех же данных. Для этого рекомендуется поднимать общее состояние до ближайшего общего предка

