

---

# Содержание

Вступление	1.1
От автора	1.1.1
Подключаем react как script	1.2
Создание компонента	1.3
Использование props	1.4
If-else, тернарный оператор	1.5
Порефакторим...	1.6
Prop-types	1.7
Использование state	1.8
Подробнее о state	1.8.1
Работа с input	1.8.2
Жизненный цикл компонента	1.9
Работа с формой	1.10
Добавить новость	1.10.1
Итоги по основам	1.11
create-react-app	1.12
Приборка и импорты	1.12.1
Асинхронные запросы	1.13
Спам-фильтр	1.14
componentWillReceiveProps	1.14.1
getDerivedStateFromProps	1.14.2
Порефакторим...	1.14.3
Заключение	1.15

# React.js курс для начинающих

Это обновленная версия моего [популярного tutorials](#) (более 250 000 читателей) по React. Я очень рад, что у учебника появилась вторая жизнь. Спасибо всем, кто учится и учился по этим книгам. Спасибо за ваши письма, лайки и репосты.

В 2018м году у меня появилось онлайн-сообщество. Подробнее в разделе "[От Автора](#)"

---

В данном курсе разбираются основы React.js

Результатом курса будет небольшое приложение новостей, в котором можно добавить новость, а так же посмотреть у новости "подробнее".

После прочтения курса, вы научитесь:

1. Создавать компоненты, учитывая *propTypes*
2. Грамотно использовать *props* и *state* компонента
3. Работать с формой
4. Работать с react dev tools
5. Рефакторить и быть лучше ;)

В тексте курса часто встречаются небольшие задачки, а так же приводится их решение.

Для успешного прохождения курса, вам потребуются знания:

1. HTML/CSS
  2. Javascript (или хотя бы jQuery, если вы понимаете, что \$ всего лишь функция...)
- 

## Поддержать проект

Вы можете [поддержать проект](#), мне будет очень приятно.

Если у вас не получается поддержать проект материально, вы можете [оставить отзыв](#) в группе vk.

## Обо мне



Меня зовут Максим Пацианский, я Frontend-разработчик, стартанул в этой теме с Joomla! сайтов в 2008 году.

Занимаюсь консультированием по React более 2х лет, с момента выхода прошлых учебников.

Подробнее о моем опыте консультирования я [писал на хабре](#).

## Напутствие

Пожалуйста, выполняйте код по ходу книги. Ломайте его, "консольте", интересуйтесь.

## Полезные ссылки

Мои уроки/вебинары/соц.сети:

- [Расписание стримов и вебинаров](#) (на сайте есть текстовые версии вебинаров)
- [Youtube канал](#) с записями вебинаров и стримов
- Группа [vkontakte](#)
- Канал в [telegram](#)
- [Twitter](#)
- [Facebook](#)

[React.js](#) (EN) - офф.сайт, содержит примеры для изучения

## Консультации и платные услуги

Общение по скайпу - 45\$ / час

Создание сервисов с использованием react, поиск проблем с производительностью, помощь в собеседовании разработчиков - цена по запросу.

Пишите на **maxfarseer@gmail.com** с темой **"Консультация React v2"**

# Подключение react'a с помощью тэга script

React.js это всего лишь библиотека. Не фреймворк.

Как и любая другая библиотека, реакт добавляется на страницу с помощью тэга `script`.

Так как в современном js "рулят" модули и различного рода преобразования/сжимания и так далее - *react* отлично дружит с *webpack*, *babel* и прочими. Для простоты, мы начнем работать с реактом, как с обычной библиотекой, например, jQuery, но затем перейдем на удобный инструмент - [create-react-app](#).

---

Создайте следующие файлы в директории с вашим проектом.

```
+-- .gitignore (если будете использовать git)
+-- index.html
```

---

## index.html

В мире программирования, каждый урок начинается с hello, world.

*index.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React [RU] Tutorial</title>
    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

    <!-- Не используйте это в production -->
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">

    </script>

  </body>
</html>
```

Во-первых, мы подключили библиотеки react и react-dom как обычные скрипты

Во-вторых, мы подключили babel библиотеку, для того, чтобы наша строка

```
<h1>Привет, мир!</h1>,
```

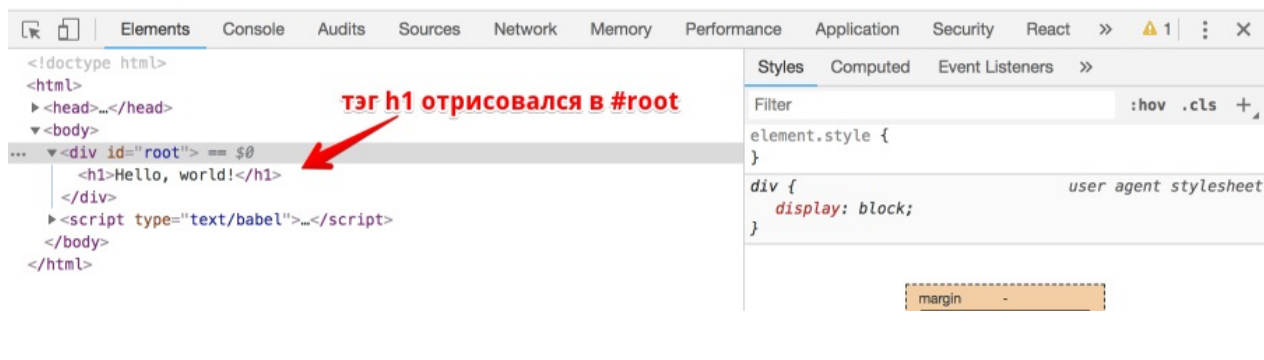
которая написана на вспомогательном языке JSX, превратилась в валидный javascript-код:

```
React.createElement("h1", null, "Hello, world!"),
```

На протяжении всего tutorials мы будем использовать JSX.



# Hello, world!



[Исходный код](#) для данного раздела.

# Создание компонента

ReactDOM.render принимает react-компонент (далее буду называть просто "компонент") и DOM-элемент, в который мы хотим "примонтировать" наше приложение.

`<h1>Hello, world!</h1>` - как ни странно, это примитивный компонент.

Пока ничего интересного, но давайте представим такой псевдо-код:

```
var photos = ['images/cat.jpg', 'images/dog.jpg', 'images/owl.jpg']

ReactDOM.render(
  <App>
    <Photos photos=photos />
    <LastNews />
    <Comments />
  </App>,
  document.getElementById('root')
);
```

Что примечательного в данном псевдо-коде? Он очень хорошо читается, ведь очевидно, что наше приложение (App) отображает: фото (кошка, собака, сова), новости и комментарии.

Хочу вас обрадовать, React.js код выглядит практически так же. Он отлично читается, так как деление на компоненты позволяет отлично структурировать код.

Давайте создадим примитивный компонент:

*index.html*



```
<!DOCTYPE html>
...
<body>
  <div id="root"></div>
  <script type="text/babel">

    const App = () => {
      return <p>Всем привет, я компонент App</p>
    }

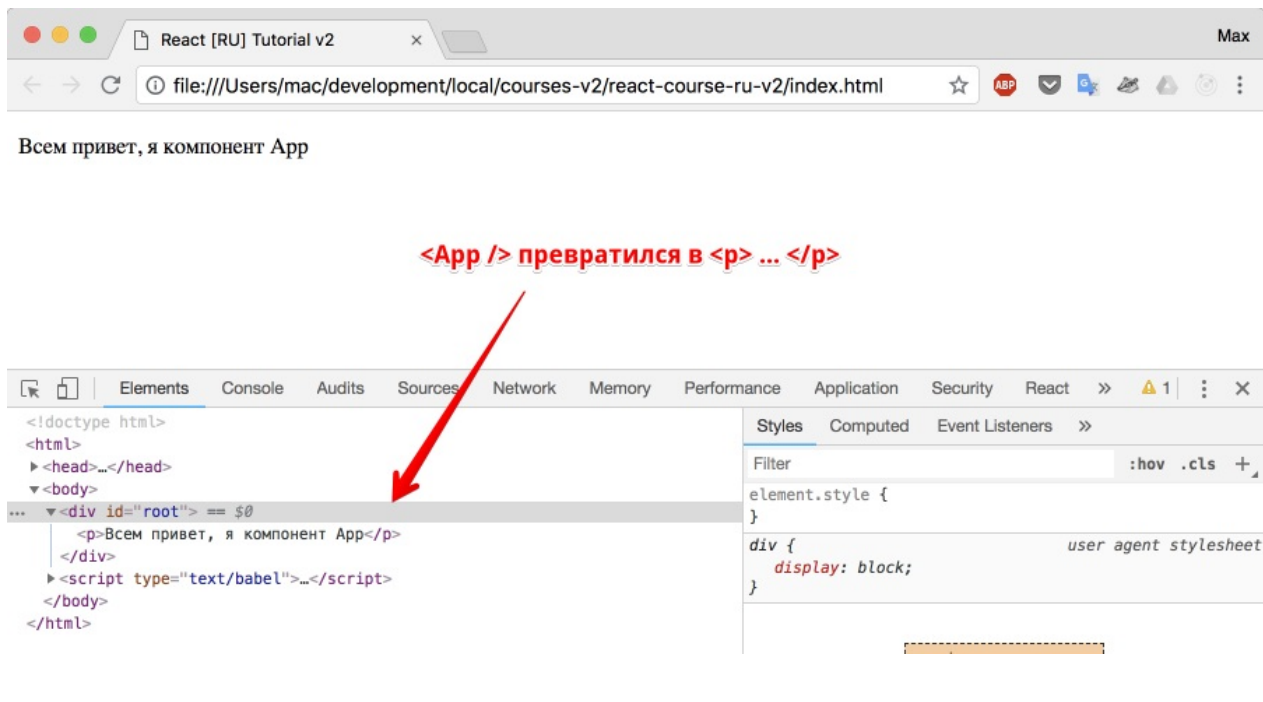
    ReactDOM.render(
      <App />,
      document.getElementById( 'root' )
    );

  </script>

</body>
</html>
```

Что примечательного? Мы скрыли в `<App />` разметку. Да, в этом примере это одна строка и чувство эйфории отсутствует, но то ли еще будет! Пока запомним, что если мы хотим отрисовать в JSX компонент, то мы обязательно должны называть и вызывать его с **Большой** буквы.

Смотрим на результирующий html-код:



Мы создали компонент с помощью функции. Но компоненты, можно создавать и с помощью `class`. Убьем сразу нескольких зайцев:

- изучим как создавать компоненты с помощью class
- как передать CSS-стиль
- как отрисовать сразу несколько компонентов

*index.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React [RU] Tutorial v2</title>
    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  >
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

    <style>
      .red {
        color: #FF0000;
      }
    </style>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">

      const App = () => {
        return <p>Всем привет, я компонент App</p>
      }

      class BigApp extends React.Component {
        render() {
          return (
            <div>
              <h1>Я компонент, BigApp</h1>
              <p className='red'>Компоненты можно вкладывать друг в друга.</p>
              <App/>
            </div>
          )
        }
      }

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

    </script>

  </body>
</html>
```

Синтаксис:

```
class (название) extends (что будем наследовать)
```

так же позволяет создать компонент. Здесь стоит отметить, что если компонент создан с помощью класса, то JSX разметка пишется внутри метода **render**. Это ключевой метод, в котором мы указываем, что будет отображаться пользователю на странице.

Компоненты созданные с помощью class, называются statefull компоненты (то есть, компоненты с *состоянием*), а компоненты созданные с помощью функции - stateless component (то есть, компоненты *без состояния*). Зачем такое деление - узнаем позже.

В примере мы добавили стиль для параграфа, через `className`, а не через `class`, как мы привыкли делать это. Почему? Потому что, мы находимся внутри JSX-синтаксиса, в котором html и js идут вперемешку, а слово class зарезервировано в javascript.

Напоследок отмечу, что мы с легкостью смогли вложить один компонент в другой.

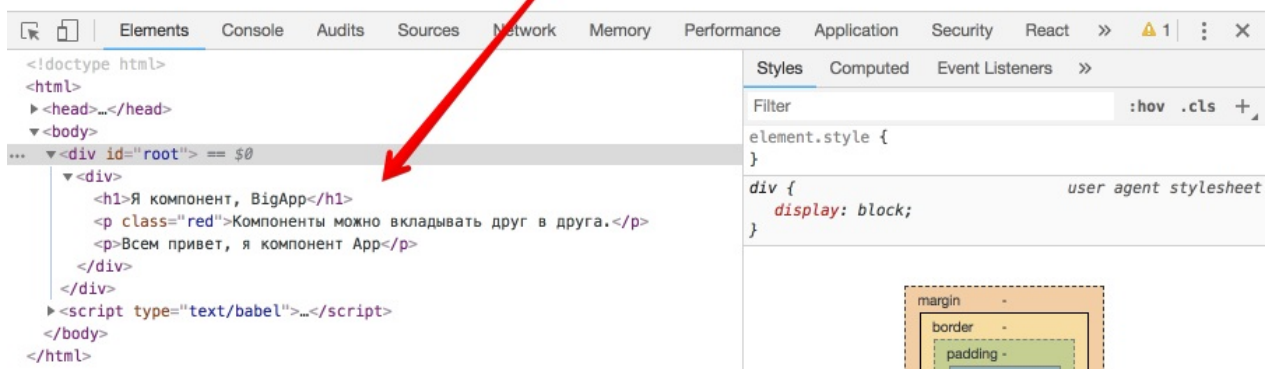


## Я компонент, BigApp

Компоненты можно вкладывать друг в друга.

Всем привет, я компонент App

разметка наших компонентов



В разметке все как мы и ожидали. Однако, я уже вижу читателей, кому не нравится лишний `div`.

## Каждый компонент должен возвращать один узел

Рассмотрим проблему с `div`-ом. Как написано в заголовке, мы должны возвращать всегда один `dom`-узел. Попробуем удалить `div`:

*index.html*

```
<script type="text/babel">

  const App = () => {
    return <p>Всем привет, я компонент App</p>
  }

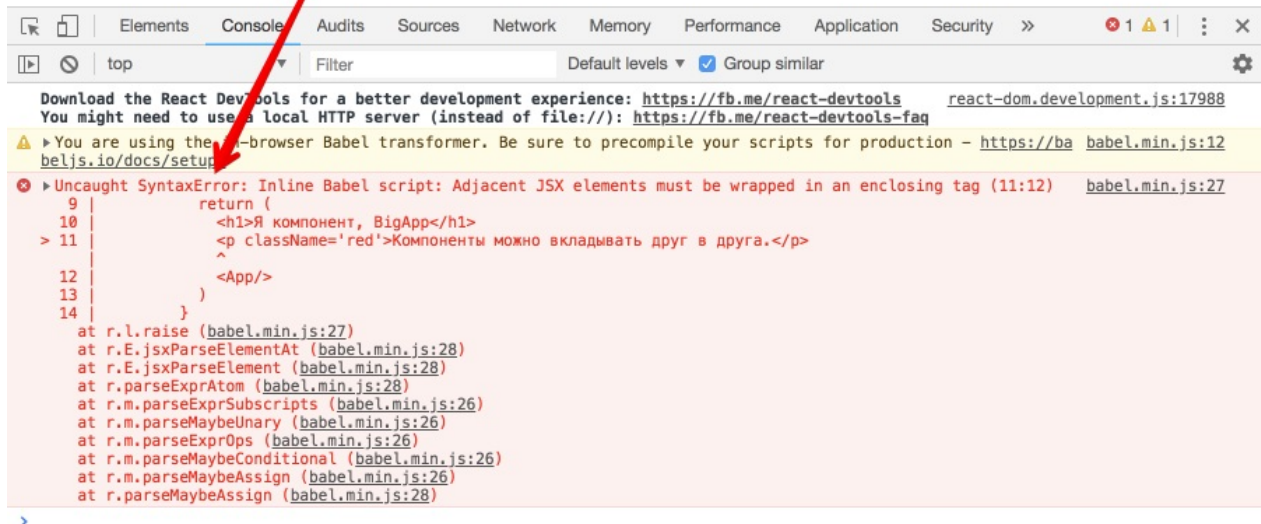
  class BigApp extends React.Component {
    render() {
      // убрали div
      return (
        <h1>Я компонент, BigApp</h1>
        <p className='red'>Компоненты можно вкладывать друг в друга.</p>
        <App/>
      )
    }
  }

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

</script>
```



### синтаксическая ошибка



Ошибка: jsx-элементы должны быть обернуты в один тэг. Что делать, если не хочется городить еще один div? Ответ: `React.Fragment`

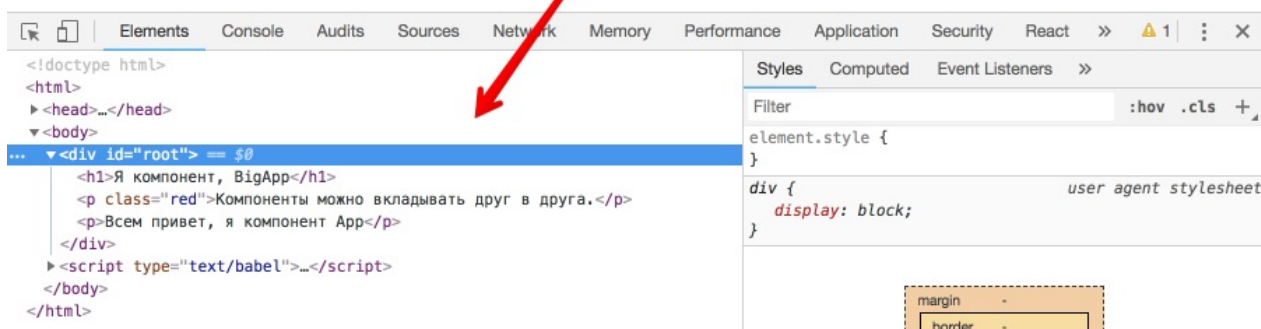


## Я компонент, BigApp

Компоненты можно вкладывать друг в друга.

Всем привет, я компонент App

### лишнего div'a нет



Все довольны. Разницы особо нет, как вам больше нравится, так и пишете, но помните: все что вы возвращаете в `render` методе или в `return` у `stateless`-компонента должно быть обернуто в один тэг / `React.Fragment`.

Давайте разовьем идею: научим BigApp отображать новости. Для этого, нам потребуется создать компонент `<News />` и вложить его в BigApp.

*index.html*

```
const App = () => {
  return <p>Всем привет, я компонент App</p>
}

const News = () => {
  return <p>К сожалению, новостей нет</p>
}

class BigApp extends React.Component {
  render() {
    return (
      <React.Fragment>
        <h1>Я компонент, BigApp</h1>
        <p className='red'>Компоненты можно вкладывать друг в друга.</p>
        <App />
        <News />
      </React.Fragment>
    )
  }
}

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

Давайте, вновь взглянем на код и поищем примечательные места.

**Во-первых** - мы никак не изменили код внутри ReactDOM.render. Мы просто вложили в BigApp еще один компонент.

**Во-вторых**, как уже было сказано - компонент `<BigApp />` содержит в себе компонент `<News />`, словно это просто дочерний `<div></div>` элемент.

**В-третьих**, наш компонент `<News />` такой же примитивный, как и App, поэтому мы создали его через функцию (а не через class).

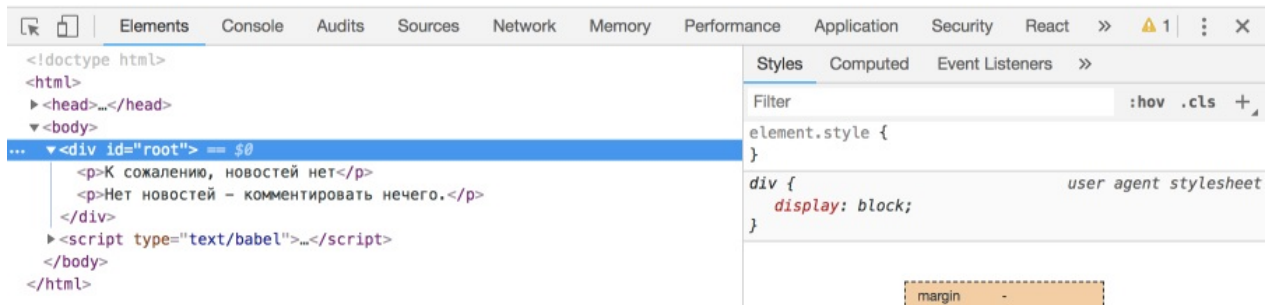
---

Задача на понимание происходящего: Удалите компонент `<BigApp/>`, оставьте `<App />` (не переписывая его на statefull-компонент). В `<App />` отображайте `<News />`. Так же создайте компонент `<Comments />` и сделайте, чтобы он отображался после новостей. Текст компонента: "Нет новостей - комментировать нечего."



К сожалению, новостей нет

Нет новостей - комментировать нечего.



*Решение для задачи всегда публикуется ниже по тексту, и обычно содержит сначала подсказки, а потом код всего решения. Здесь подсказок нет.*

**Решение:**

*index.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React [RU] Tutorial v2</title>
    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  >
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

    <style>
      .red {
        color: #FF0000;
      }
    </style>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">

      const News = () => {
        return <p>К сожалению, новостей нет</p>
      }

      const Comments = () => {
        return <p>Нет новостей - комментировать нечего.</p>
      }

      const App = () => {
        return (
          <React.Fragment>
            <News />
            <Comments />
          </React.Fragment>
        )
      }

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

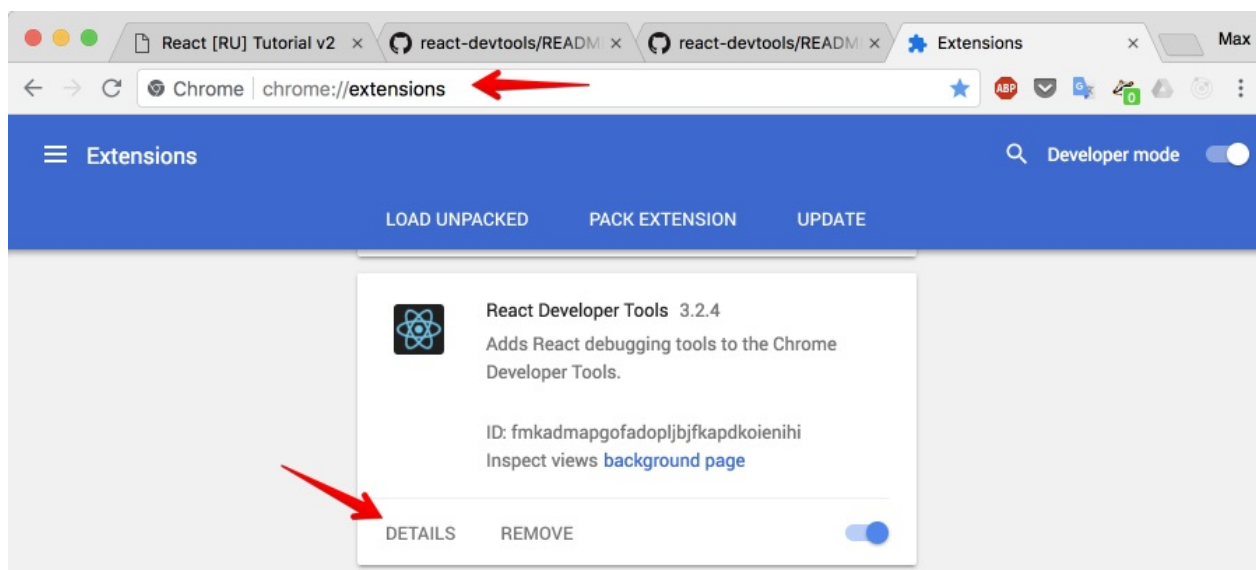
    </script>

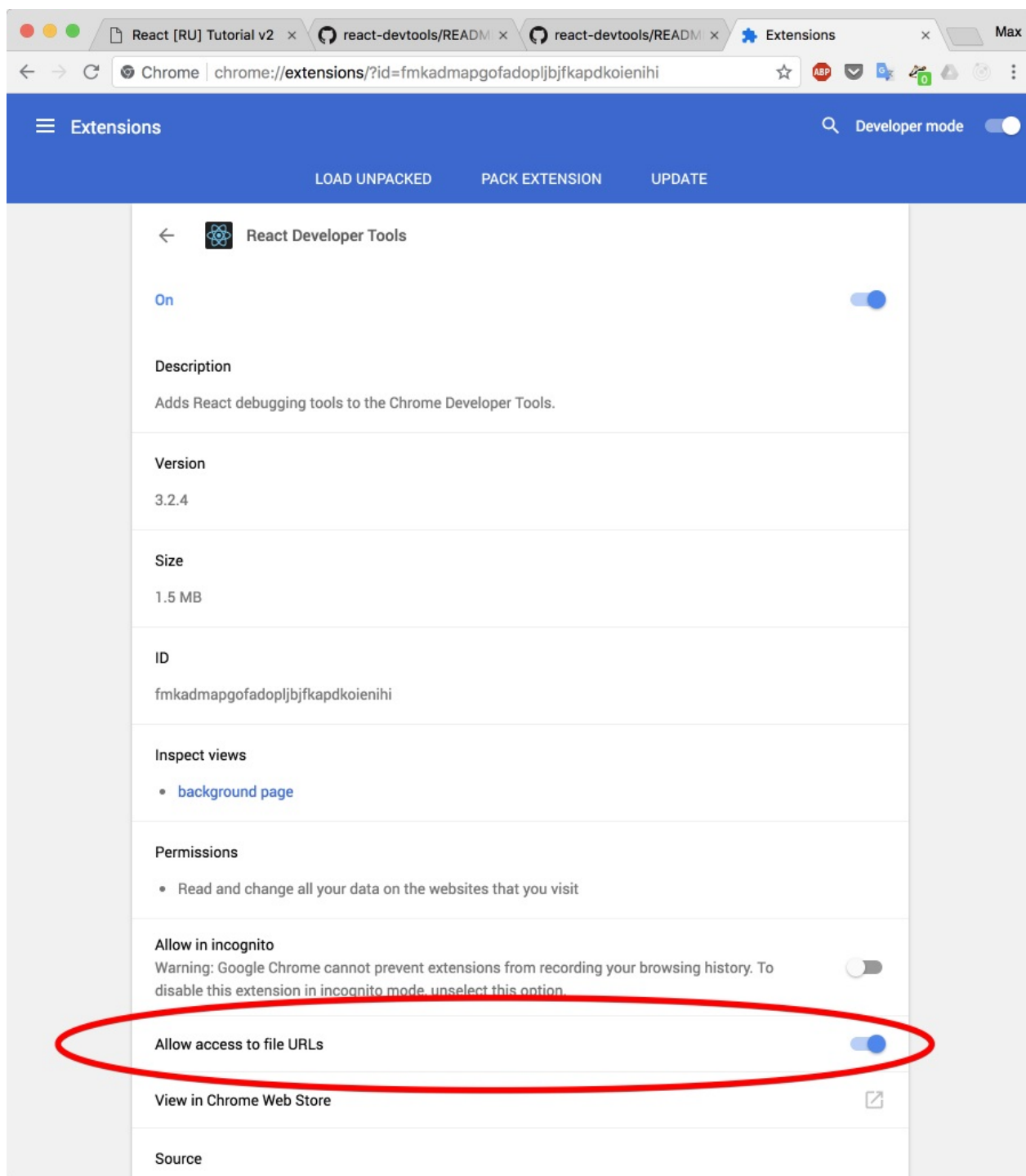
  </body>
</html>
```

Прежде чем переходить к следующему уроку, предлагаю вам установить react devtools (плагин для [хрома](#), плагин для [мозилы](#)).

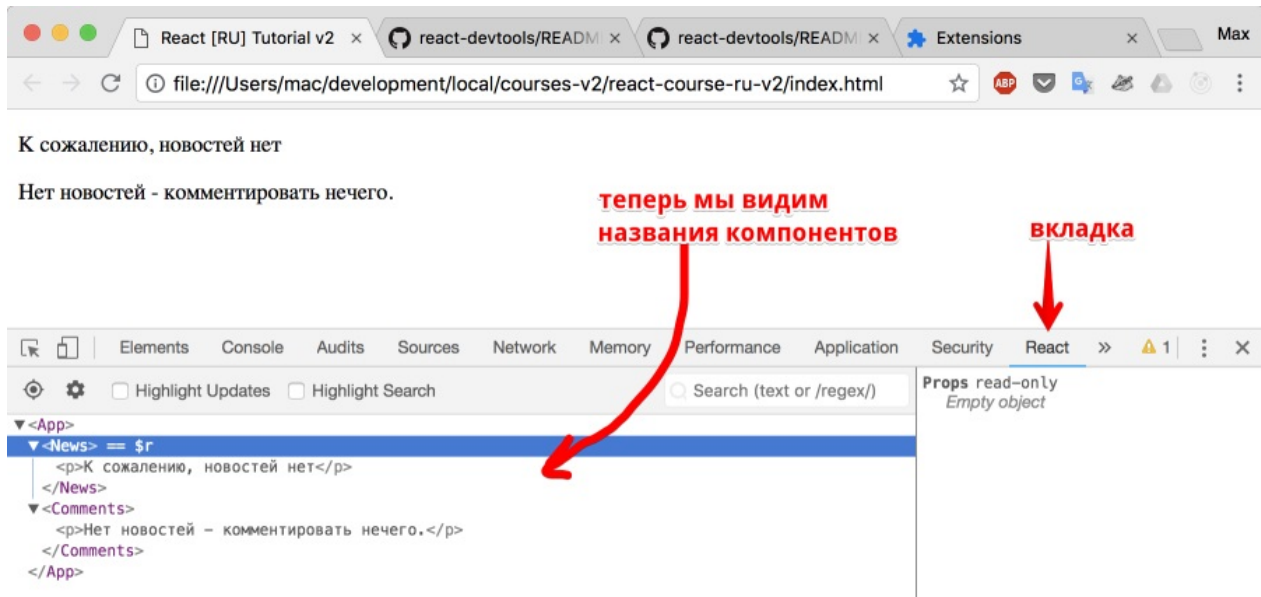


Так как мы разрабатываем просто в файлике `index.html`, нужно активировать опцию в плагине (в хrome выглядит так):





После установки и настройки откройте вкладку React в консоли разработчика.



Пытливый читатель уже заметил окошечко "Props". Ок, об этом и поговорим в следующей главе.

[Исходный код](#) на данный момент.

# Использование props

У каждого компонента могут быть свойства. Они хранятся в `this.props`, и передаются компоненту как атрибуты.

## Общий вид:

```
const wizard = {name: Garry, surname: Potter};  
<MyComponent data={wizard} eshe_odno_svoistvo={[1,2,3,4,5]} />
```

В свойство можно передать любой javascript примитив, объект, переменную и даже выражение. Значение свойства должно быть взято в фигурные скобки.

Значения доступны через `this.props.ИМЯ_СВОЙСТВА` (в statefull-компонентах) или в первом аргументе функции (в stateless).

В нашем случае, если говорить про statefull, мы получим:

- `this.props.data` - объект `{name: Garry, surname: Potter}`
- `this.props.eshe_odno_svoistvo` - массив `[1, 2, 3, 4, 5]`

`this.props` используются **только** для чтения!

---

Давайте создадим несколько новостей для нашего приложения.

Добавьте в начало script-тэга массив с данными

```
const myNews = [
  {
    author: 'Саша Печкин',
    text: 'В четверг, четвертого числа...'
  },
  {
    author: 'Просто Вася',
    text: 'Считаю, что $ должен стоить 35 рублей!'
  },
  {
    author: 'Max Frontend',
    text: 'Прошло 2 года с прошлых учебников, а $ так и не стоит 35'
  },
  {
    author: 'Гость',
    text: 'Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru'
  }
];
```

И измените строку с подключением компонента News следующим образом:

*index.html*

```
...
const myNews = [
  {
    author: 'Саша Печкин',
    text: 'В четверг, четвертого числа...'
  },
  {
    author: 'Просто Вася',
    text: 'Считаю, что $ должен стоить 35 рублей!'
  },
  {
    author: 'Max Frontend',
    text: 'Прошло 2 года с прошлых учебников, а $ так и не стоит 35'
  },
  {
    author: 'Гость',
    text: 'Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru'
  }
];

const News = () => {
  return <p>К сожалению, новостей нет</p>
}

const Comments = () => {
  return <p>Нет новостей - комментировать нечего.</p>
}

const App = () => {
  return (
    <React.Fragment>
      <News data={myNews}/> {/* добавили свойство data */}
      <Comments />
    </React.Fragment>
  )
}

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

Обратите внимание, комментарии внутри JSX пишутся в фигурных скобках: `{/* текст комментария */}`

Так же прошу заметить, что JSX это не весь js-код, который содержится в тэге script, грубо говоря JSX - это HTML-разметка + переменные/выражения. Поэтому в остальных местах можно писать комментарии привычным для вас способом ( `//...` или `/*...*/` )

Откройте в консоли вкладку react (конечно, предварительно обновив страницу).

The screenshot shows a web browser window with the URL `file:///Users/mac/development/local/courses-v2/react-course-ru-v2/index.html`. The page content displays the text "К сожалению, новостей нет" and "Нет новостей - комментировать нечего." Below the browser window, the React DevTools component inspector is open. The left pane shows the component tree with `<News data=[...]` selected. A red arrow points from the text "props (свойства) компонента News" to the `data` prop in the component tree. The right pane shows the props for the selected component, with `data` expanded to show an array of four news items.

К сожалению, новостей нет

Нет новостей - комментировать нечего.

**props (свойства) компонента News**

Props read-only

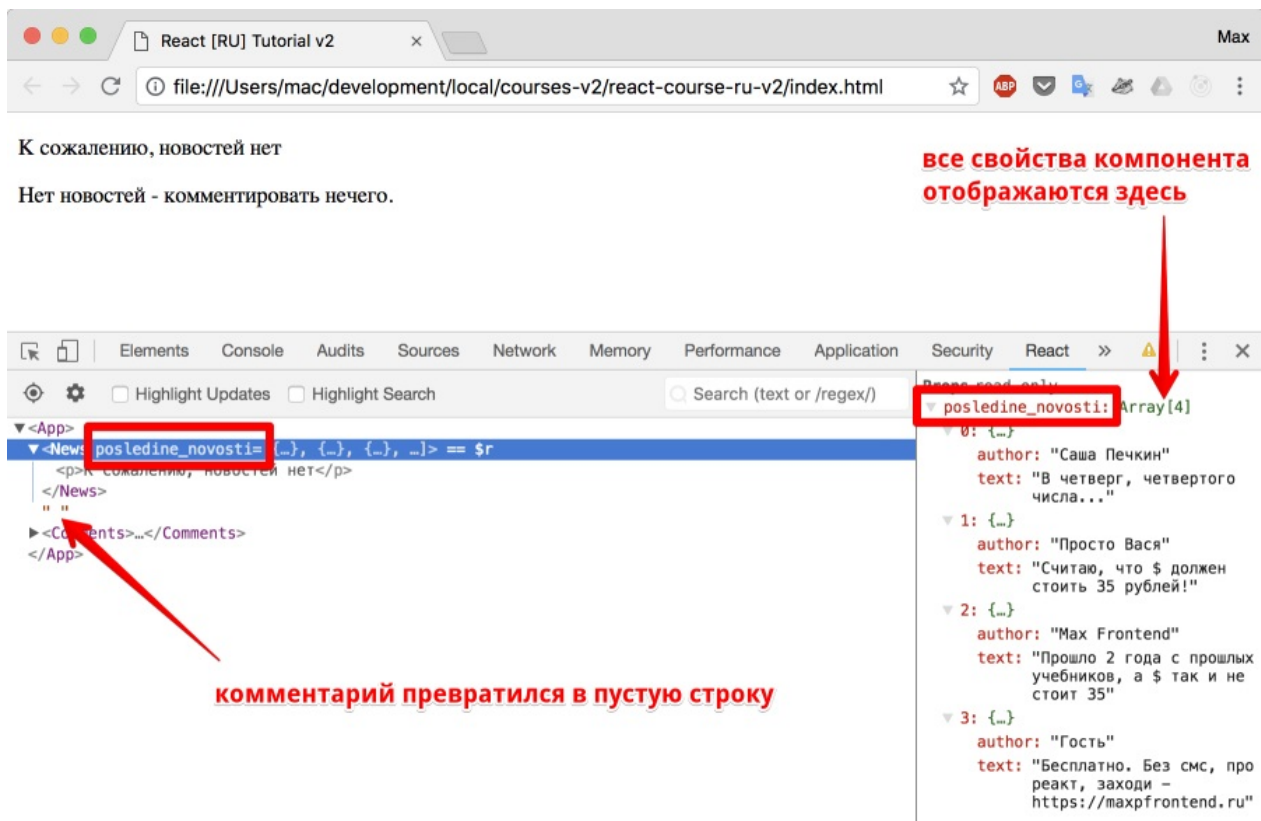
`data: Array[4]`

- `0: {}`
  - `author: "Саша Печкин"`
  - `text: "В четверг, четвертого числа..."`
- `1: {}`
  - `author: "Просто Вася"`
  - `text: "Считаю, что $ должен стоить 35 рублей!"`
- `2: {}`
  - `author: "Max Frontend"`
  - `text: "Прошло 2 года с прошлых учебников, а $ так и не стоит 35"`
- `3: {}`
  - `author: "Гость"`
  - `text: "Бесплатно. Без смс, про реакт, заходи - https://maxpfrendend.ru"`

**Напомню:** в данный момент, мы добавили свойство `data` в наш компонент `<News />`. Необязательно было называть свойство так, можно было написать, например:

```
<News posledine_novosti={my_news} />
```

Тогда в консоли разработчика, это выглядело бы так:



Ок, у нашего компонента есть свойство, в котором лежат наши новости, но компонент не умеет их отображать. Это легко исправить.

Представим HTML разметку для наших новостей:

```
<div class="news">
  <p class="news__author">Саша Печкин:</p>
  <p class="news__text">В четверг, четвертого числа...</p>
</div>
```

**Вопрос:** У нас есть разметка для одного элемента данных, есть данные целиком (массив `myNews`). Как отобразить эти данные?

**Ответ:** Нужно создать шаблон, пройти по всем переменным в массиве с новостями и подставить значения.

Когда вам нужно отобразить переменную в шаблоне разметки JSX - она так же оборачивается в фигурные скобки. На практике это выглядит проще, чем в теории, поэтому давайте представим, как может выглядеть наша JSX разметка:



```
this.props.data.map(function(item, index) {  
  return (  
    <div key={index}>  
      <p className="news__author">{item.author}</p>  
      <p className="news__text">{item.text}</p>  
    </div>  
  )  
})
```

1. Мы использовали метод массивов - *Map*. Если вы незнакомы с ним, [прочитайте документацию](#).
2. Мы обернули разметку внутри *return* в корневой элемент `<div>`. Мы могли бы обернуть ее в любой другой элемент, главное, **напоминаю** - внутри *return* всегда должен возвращаться DOM-узел (то есть, что угодно, обернутое в родительский тэг или в `React.Fragment`).
3. Мы использовали у родительского элемента атрибут **key** ( `<div key={index}>` ). Если объяснить предельно просто: реакту нужна уникальность, чтобы все его механизмы работали корректно. По "ключу" он будет понимать с каким именно дочерним узлом вы работаете и какому родителю он принадлежит. Индекс - не идеальный вариант для ключа, мы это исправим далее.
4. Мы использовали в шаблоне, значения переменных + текст, например `<p className="news__author">{item.author}</p>` которое могло бы быть представлено в нативном js-коде как `<p className="news__author">' '+item.author+' '</p>` (пустая строка + значение переменной + двоеточие)

В результате работы функции `map`, мы получили новый массив, состоящий из необходимых нам реакт-элементов. Это и есть решение нашей задачи, нам осталось лишь сохранить этот массив в переменной, например `newsTemplate`, и в *render* функции компонента `<News />` вернуть разметку + "переменную-шаблон".

News по-быстрому переделаем в statefull (через `class ... extends`):

*index.html*

```
...
const myNews = [
  {
    author: 'Саша Печкин',
    text: 'В четверг, четвертого числа...'
  },
  {
    author: 'Просто Вася',
    text: 'Считаю, что $ должен стоить 35 рублей!'
  },
  {
    author: 'Max Frontend',
    text: 'Прошло 2 года с прошлых учебников, а $ так и не стоит 35'
  },
  {
    author: 'Гость',
    text: 'Бесплатно. Без смс, про реакт, заходи - https://maxpffrontend.ru'
  }
];

class News extends React.Component {
  render() {
    const newsTemplate = this.props.data.map(function(item, index) {
      return (
        <div key={index}>
          <p className="news__author">{item.author}</p>
          <p className="news__text">{item.text}</p>
        </div>
      )
    })

    console.log(newsTemplate)

    return (
      <div className="news">
        {newsTemplate}
      </div>
    )
  }
}
...
```

Посмотрим что получилось:



Саша Печкин:

В четверг, четвертого числа...

Просто Вася:

Считаю, что \$ должен стоить 35 рублей!

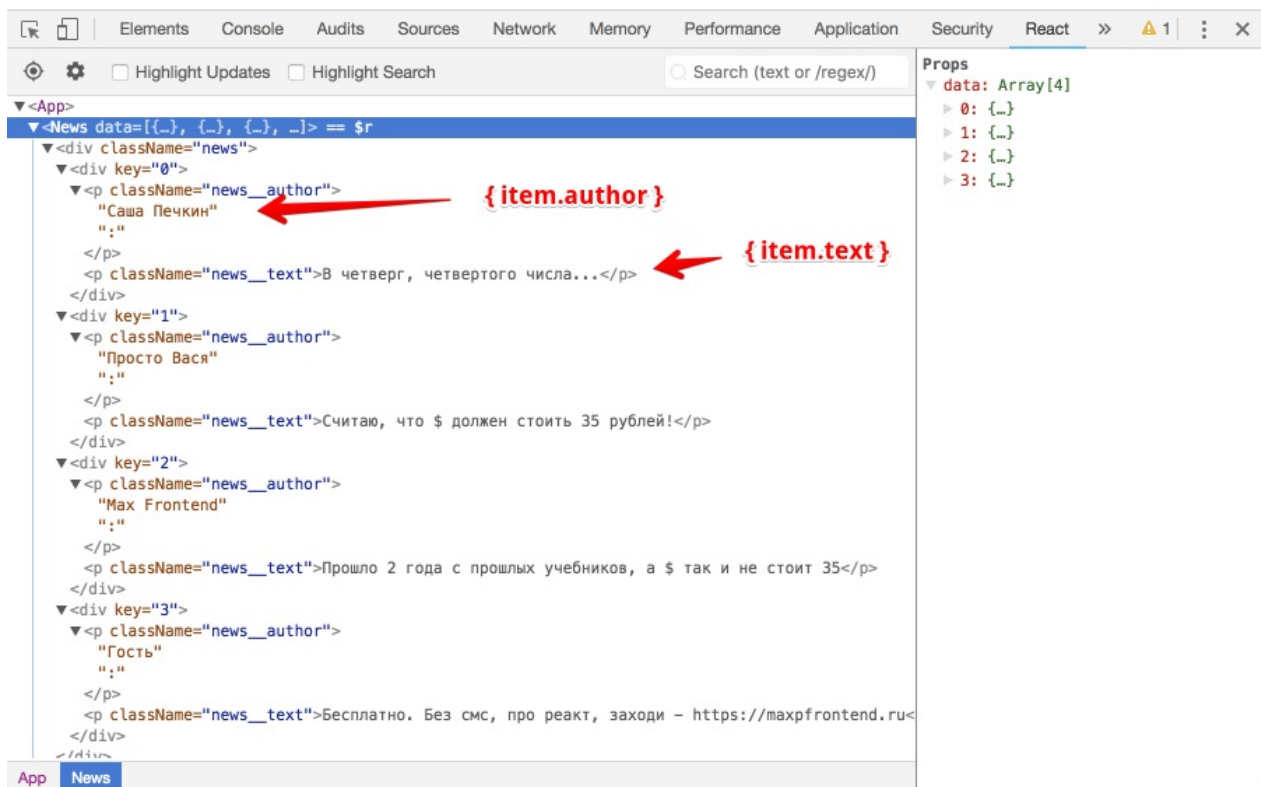
Max Frontend:

Прошло 2 года с прошлых учебников, а \$ так и не стоит 35

Гость:

Бесплатно. Без смс, про реакт, заходи - <https://maxpffrontend.ru>

Нет новостей - комментировать нечего.



Вы еще не в восторге? Количество кода - кот наплакал.

Напоследок, мне не хочется, но придется вновь разрушить магию. Давайте посмотрим, что возвращает `console.log(newsTemplate)` (откройте вкладку Console).

Взглянем в консоль:



Саша Печкин:

В четверг, четвертого числа...

Просто Вася:

Считаю, что \$ должен стоять 35 рублей!

**Новости!**

Max Frontend:

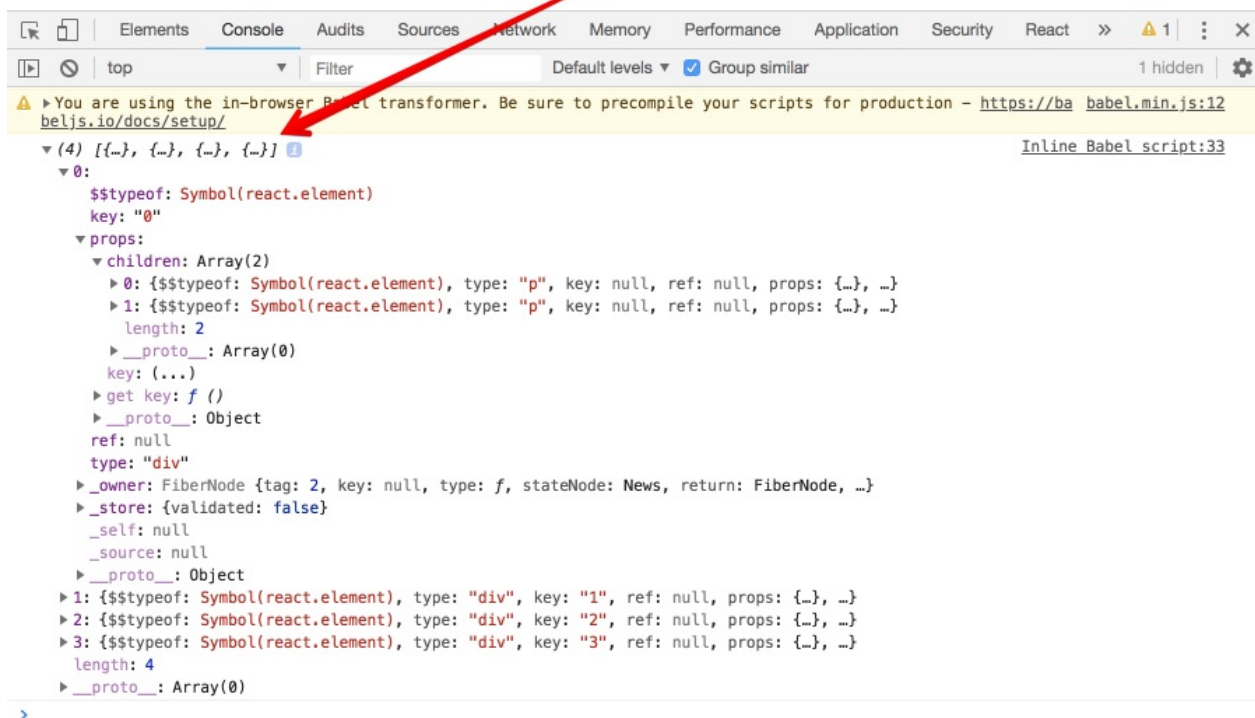
Прошло 2 года с прошлых учебников, а \$ так и не стоит 35

Гость:

Бесплатно. Без смс, про реакт, заходи - <https://maxpfrendend.ru>

Нет новостей - комментировать нечего.

**newsTemplate массив из React элементов**



Объект, у объекта свойства... все как обычно в мире javascript.

## Почему не стоит указывать index в качестве ключа

Если кому интересны причины, то приведу вам комментарий из старого учебника:

P.S. Здесь и в продолжении всего курса в коде для отображения массива новостей используется `key = {index}` . Обратите внимание на следующую ветку [комментариев](#) (спасибо *DeLaVega* и *geakstr*).

Суть в том, что `index` не лучший вариант для ключа, когда ваши "айтемы" могут менять порядок. У нас новости не меняются, но тем не менее, мы можем быстро решить нашу проблему добавив "действительно" уникальное значение, которое не будет изменяться, если у элементов поменяется `index`.

Конечно же, это свойство называется `id` ;) Добавим его в массив и будем использовать в качестве ключа.

*index.html*

```
...
const myNews = [
  {
    id: 1, // добавили id
    author: 'Саша Печкин',
    text: 'В четверг, четвертого числа...'
  },
  {
    id: 2,
    author: 'Просто Вася',
    text: 'Считаю, что $ должен стоить 35 рублей!'
  },
  {
    id: 3,
    author: 'Max Frontend',
    text: 'Прошло 2 года с прошлых учебников, а $ так и не стоит 35'
  },
  {
    id: 4,
    author: 'Гость',
    text: 'Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru'
  }
];

class News extends React.Component {
  render() {
    const newsTemplate = this.props.data.map(function(item) {
      return (
        <div key={item.id}> {/* используем id в качестве ключа */}
          <p className="news__author">{item.author}</p>
          <p className="news__text">{item.text}</p>
        </div>
      )
    })

    return (
      <div className="news">
        {newsTemplate}
      </div>
    )
  }
}
...
```

**Итого:** мы научились отображать свойства компонента.

[Исходный код](#) на текущий момент. Не забудьте удалить console.log.



## If-else, тернарный оператор

Помните, у нас была фраза "новостей нет"? Хорошо бы ее отображать, если новостей действительно нет.

Для начала, научимся отображать общее количество новостей, допустим внизу, после списка новостей.

Как бы сказал участник игры "Угадай JS" - я напишу это за одну строку. Что скажете вы? Подсказка:

```
class News extends React.Component {
  render() {
    const newsTemplate = this.props.data.map(function(item) {
      return (
        <div key={item.id}> {/* используем id в качестве ключа */}
          <p className="news__author">{item.author}</p>
          <p className="news__text">{item.text}</p>
        </div>
      )
    })

    return (
      <div className="news">
        {newsTemplate}
        {/* эта строка здесь */}
      </div>
    )
  }
}
```

**Ответ:**

```
<strong>Всего новостей: {data.length}</strong>
```

Поиграйтесь с переменной *myNews*. Сделайте ее пустым массивом, добавьте/удалите элементы. Пообновляйте страницу. Количество новостей должно работать корректно.

---

Вернемся к нашей задаче. Алгоритм прост:



Создаем переменную *newsTemplate*, если новости есть - в переменную по-прежнему будем передавать результат работы функции *map*, иначе - будем передавать сразу разметку что "новостей нет".

Компонент News:

```
class News extends React.Component {
  render() {
    const { data } = this.props // аналогично записи const data = this.props.data
    let newsTemplate

    if (data.length) { // если новости есть, пробегись map'ом
      newsTemplate = data.map(function(item) {
        return (
          <div key={item.id}>
            <p className="news__author">{item.author}</p>
            <p className="news__text">{item.text}</p>
          </div>
        )
      })
    } else { // если новостей нет - сохрани в переменную параграф
      newsTemplate = <p>К сожалению новостей нет</p>
    }

    return (
      <div className="news">
        {newsTemplate}
        <strong>Всего новостей: {data.length}</strong>
      </div>
    );
  }
}
```



Саша Печкин:

В четверг, четвертого числа...

Просто Вася:

Считаю, что \$ должен стоить 35 рублей!

Max Frontend:

Прошло 2 года с прошлых учебников, а \$ так и не стоит 35

Гость:

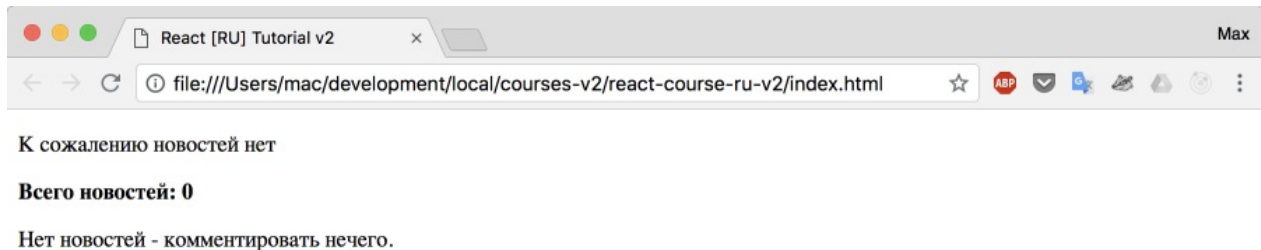
Бесплатно. Без смс, про реакт, заходи - <https://maxfrontend.ru>

**Всего новостей: 4**

Нет новостей - комментировать нечего.

Для тех, кто совсем мало знаком с js, напомним, что:

```
if (data.length)
// можно представить как
if (data.length > 0)
```



Если нет новостей - зачем нам показывать, что всего новостей 0? Давайте решим это с помощью css класса `.none`, который будем добавлять если новостей нет.

Добавьте новый стиль:

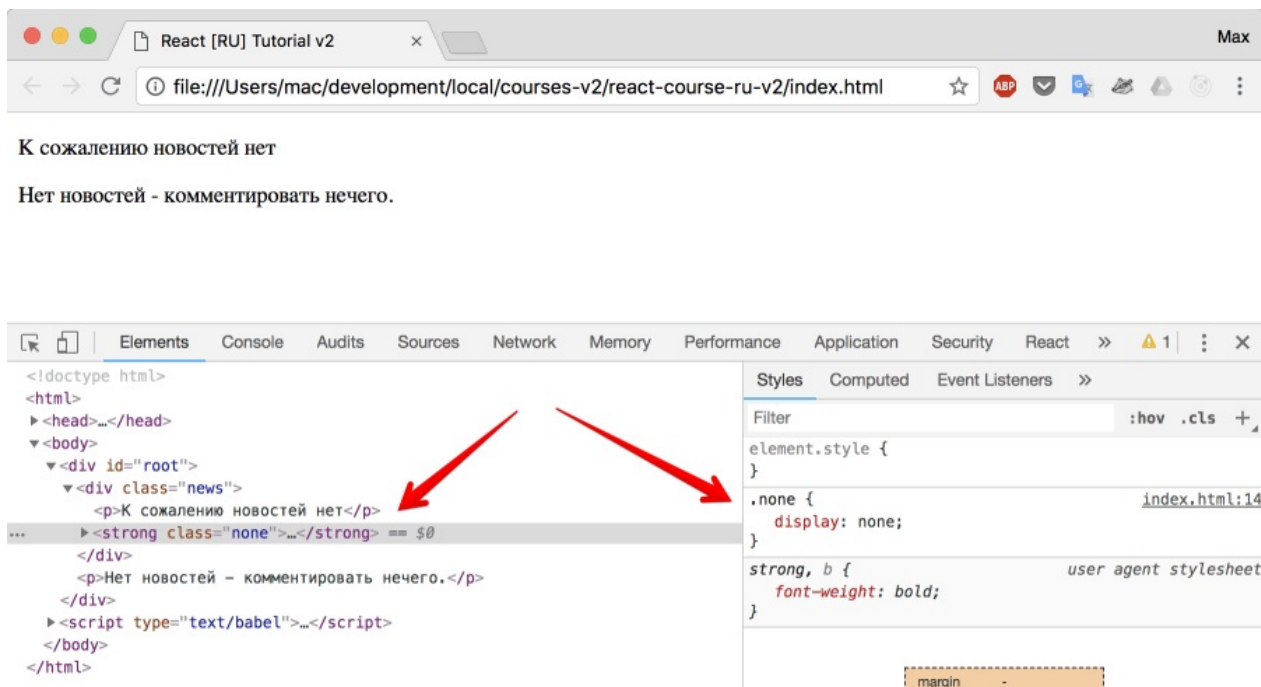
```
.none {
  display: none;
}
```

С классом `.none` все вновь решается в одну строку.

Измените строку про количество новостей следующим образом:

```
<strong className={data.length > 0 ? '' : 'none'}>Всего новостей: {data.length}</strong>
```

Проще простого: *есть новости ? 'пустой класс' : 'класс .none'*



Для работы с классами, когда их становится больше и условия становятся сложнее, можно использовать [classNames \(NPM пакет\)](#). Но сейчас в этом нет необходимости.

Вообще, тема работы со стилями в react-приложениях очень обширна. Я больше склоняюсь к SCSS + обычные классы или к styled-components.

Мы скрыли наш элемент strong с помощью класса, но при таком подходе элемент остался в DOM-дереве. Можем исправить это, не отображая элемент вовсе.

```
...
return (
  <div className="news">
    {newsTemplate}
    {
      data.length ? <strong>Всего новостей: {data.length}</strong> : null
    }
  </div>
);
...
```

**Итого:** если вам нужно отобразить что-то в зависимости от условий, делайте это так же, как если бы react не был подключен, но не забывайте, что "условия" внутри JSX пишутся в фигурных скобках. Для удобства, мы использовали *переменную-шаблон*, которую объявили **заранее**, а затем в зависимости от условия сохраняли в нее необходимую разметку.

[Исходный код](#) на данный момент.

P.S. [официальная документация](#) про If-else внутри JSX

# Порефакторим...

Для начала, удалите вовсе компонент `<Comments />` (и `const Comments...` соответственно).

Далее, давайте представим: у наших новостей появляются какие-то дополнительные поля, пользователь начинает взаимодействовать с ними, например "пометить как прочитанное" и так далее. Нам было бы удобно, чтобы каждая новость была представлена отдельным компонентом.

**Задача:** `<News />` должен рендерить список компонентов `<Article />`. Каждый компонент `<Article />` должен получать соответствующие данные, например: первый экземпляр получит первый элемент массива, второй - второй и так далее.

То есть, раньше в тар мы возвращали JSX-разметку. Но мы так же можем возвращать и компонент.

Попробуйте сами, а потом смотрите решение ниже.

Подсказка #1: if-else нашего компонента `<News />`

```
if (data.length) {
  newsTemplate = data.map(function(item) {
    return <Article key={item.id} data={item}/>
  })
} else {
  newsTemplate = <p>К сожалению новостей нет</p>
}
```

Подсказка #2 (по сути решение задачи): компонент `<Article />`

```
class Article extends React.Component {
  render() {
    const { author, text } = this.props.data
    return (
      <div className="article">
        <p className="news__author">{author}</p>
        <p className="news__text">{text}</p>
      </div>
    )
  }
}
```

Что любопытно, больше не изменилось ни-че-го.

Саша Печкин:  
В четверг, четвертого числа...

Просто Вася:  
Считаю, что \$ должен стоить 35 рублей!

Мах Frontend:  
Прошло 2 года с прошлых учебников, а \$ так и не стоит 35

Гость:  
Бесплатно. Без смс, про реакт, заходи - <https://maxpffrontend.ru>

**Всего новостей: 4**

первый компонент Article его верстка и свойства

```

<App>
  <News data={...}>
    <div className="news">
      <Article key="1" data={id: 1, author: "Саша Печкин", text: "В четверг, четвертого числа..."} />
      <Article key="2" data={id: 2, author: "Просто Вася", text: "Считаю, что $ должен стоить 35 р"} />
      <Article key="3" data={id: 3, author: "Мах Frontend", text: "Прошло 2 года с прошлых учебник"} />
      <Article key="4" data={id: 4, author: "Гость", text: "Бесплатно. Без смс, про реакт, заходи"} />
    </div>
    <strong>
      "Всего новостей: "
      "4"
    </strong>
  </div>
</News>
</App>
  
```

```

Key "1"
Props
  data: {...}
    author: "Саша Печкин"
    id: 1
    text: "В четверг, четвертого числа..."
  
```

Добавьте заголовок "Новости" в `<App />` перед компонентом `<News />`

```

const App = () => {
  return (
    <React.Fragment>
      <h3>Новости</h3>
      <News data={myNews}/>
    </React.Fragment>
  )
}
  
```

Добавьте красоты (CSS) по вкусу, либо возьмите мой вариант:

```
.none {
  display: none;
}

body {
  background: rgba(0, 102, 255, 0.38);
  font-family: sans-serif;
}

p {
  margin: 0 0 5px;
}

.article {
  background: #FFF;
  border: 1px solid rgba(0, 89, 181, 0.82);
  width: 600px;
  margin: 0 0 5px;
  box-shadow: 2px 2px 5px -1px rgb(0, 81, 202);
  padding: 3px 5px;
}

.news__author {
  text-decoration: underline;
  color: #007DDC;
}

.news__count {
  margin: 10px 0 0 0;
  display: block;
}
```

С новыми стилями, код сценария выглядит так:

*index.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React [RU] Tutorial v2</title>
    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  >
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

    <style>
      .none {
        display: none;
      }

      body {
```

```
    background: rgba(0, 102, 255, 0.38);
    font-family: sans-serif;
  }

  p {
    margin: 0 0 5px;
  }

  .article {
    background: #FFF;
    border: 1px solid rgba(0, 89, 181, 0.82);
    width: 600px;
    margin: 0 0 5px;
    box-shadow: 2px 2px 5px -1px rgb(0, 81, 202);
    padding: 3px 5px;
  }

  .news__author {
    text-decoration: underline;
    color: #007DDC;
  }
  .news__count {
    margin: 10px 0 0 0;
    display: block;
  }
</style>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">

    const myNews = [
      {
        id: 1, // добавили id
        author: 'Саша Печкин',
        text: 'В четверг, четвертого числа...'
      },
      {
        id: 2,
        author: 'Просто Вася',
        text: 'Считаю, что $ должен стоить 35 рублей!'
      },
      {
        id: 3,
        author: 'Max Frontend',
        text: 'Прошло 2 года с прошлых учебников, а $ так и не стоит 35'
      },
      {
        id: 4,
        author: 'Гость',
        text: 'Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru'
      }
    ];
```



```

class Article extends React.Component {
  render() {
    const { author, text } = this.props.data
    return (
      <div className="article">
        <p className="news__author">{author}</p>
        <p className="news__text">{text}</p>
      </div>
    )
  }
}

class News extends React.Component {
  render() {
    const { data } = this.props
    let newsTemplate

    if (data.length) {
      newsTemplate = data.map(function(item) {
        return <Article key={item.id} data={item}/>
      })
    } else {
      newsTemplate = <p>К сожалению новостей нет</p>
    }

    return (
      <div className="news">
        {newsTemplate}
        {
          data.length ? <strong className={'news__count'}>Всего новостей: {data.
length}</strong> : null
        }
      </div>
    );
  }
}

const App = () => {
  return (
    <React.Fragment>
      <h3>Новости</h3>
      <News data={myNews}/>
    </React.Fragment>
  )
}

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

</script>

```

```

    </body>
  </html>

```

В целом меня устраивает почти все. Осталось немного отполировать метод `render` компонента `<News />`. Правило следующее: стараемся в `render` держать как можно меньше кода, чтобы его было легко читать вашим коллегам. Для этого, мы `newsTemplate` будем заполнять внутри нового метода, который будем вызывать в `render`.

```

class News extends React.Component {
  renderNews = () => {
    const { data } = this.props
    let newsTemplate = null

    if (data.length) {
      newsTemplate = data.map(function(item) {
        return <Article key={item.id} data={item}/>
      })
    } else {
      newsTemplate = <p>К сожалению новостей нет</p>
    }

    return newsTemplate
  }
  render() {
    const { data } = this.props

    return (
      <div className="news">
        {this.renderNews()}
        {
          data.length ? <strong className={'news__count'}>Всего новостей: {data.length}
        </strong> : null
        }
      </div>
    );
  }
}

```

Что примечательного в этом коде?

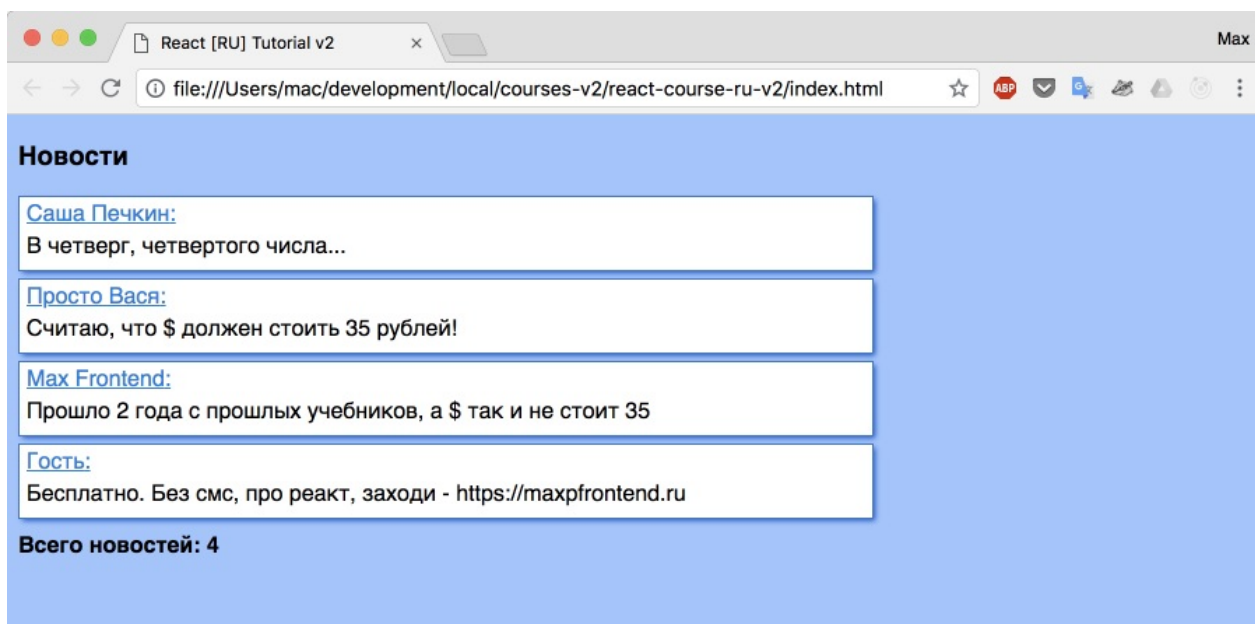
Мы создали метод `renderNews` (внутри `class`) с помощью так называемой "жирной стрелочной функции" (запись вида `methodName = () => ...`). При такой записи, внутри функции мы не теряем контекст `this`. То есть, можем обращаться к `this.props`, например.

Почему мы метод `render` не описываем через жирную стрелочную функцию? Потому что, это метод жизненного цикла react-компонента, и туда `this` "прокидывает" уже сам `react`.

Далее, так как мы `renderNews` создали в качестве метода, значит внутри компонента, мы должны обращаться к нему как `this.xxx` (где `xxx` - название метода, в нашем случае `renderNews` ).

Что же изменилось, спросите вы? Было много кода в `render`, стало чуть выше. Дело в том, что когда ваши компоненты разрастутся, очень удобно иметь "рендер" хорошо читаемым, то есть таким, в котором все лишнее спрятано.

Посмотрим, что вышло в итоге:



[Исходный код](#) на данный момент.

# Prop-types

(скучный, но небольшой теоретический перекур)

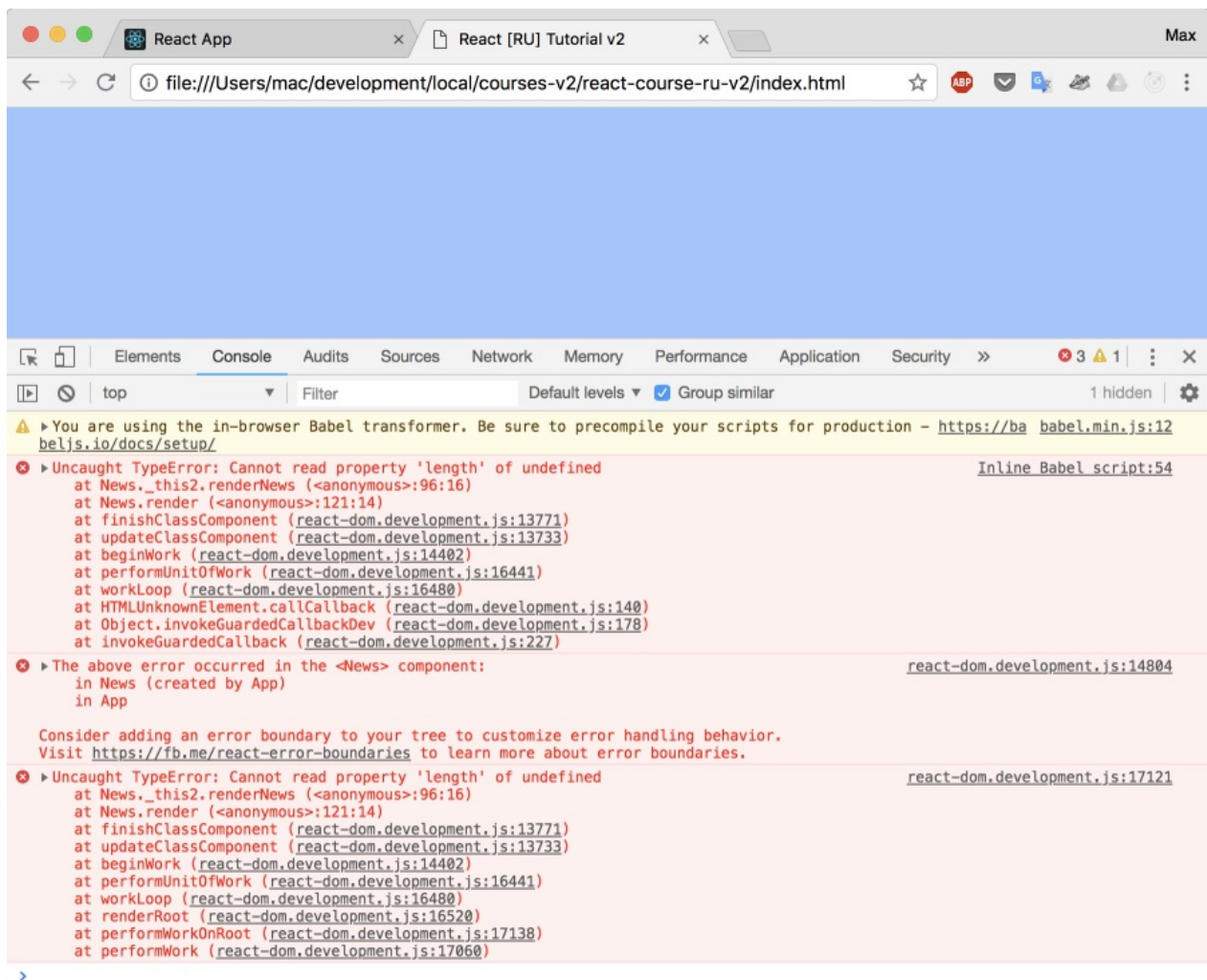
Перед выполнением данного урока, не забывайте, что *PropTypes* не работает с *production* версией реакта. Эта фишка только для разработки, так как валидация - дорогая операция.

---

Давайте сломаем наш код:

```
const App = () => {  
  return (  
    <React.Fragment>  
      <h3>Новости</h3>  
      <News/> {/* удалили передачу data */}  
    </React.Fragment>  
  )  
}
```

Обновим страницу - видим сообщение об ошибке.



В принципе, все понятно - мы пытаемся вызвать метод `map` у `undefined`. У примитива `undefined`, как известно никаких методов нет - ошибка, получите распишитесь. Хорошо, что кода мало и мы быстро выяснили в чем проблема. Еще лучше, что есть возможность улучшить наше положение, добавив *propTypes* - специальное свойство, которое будет "валидировать" наш компонент.

Добавьте загрузку еще одного скрипта в документ:

```
<script src="https://unpkg.com/prop-types@15.6/prop-types.js"></script>
```

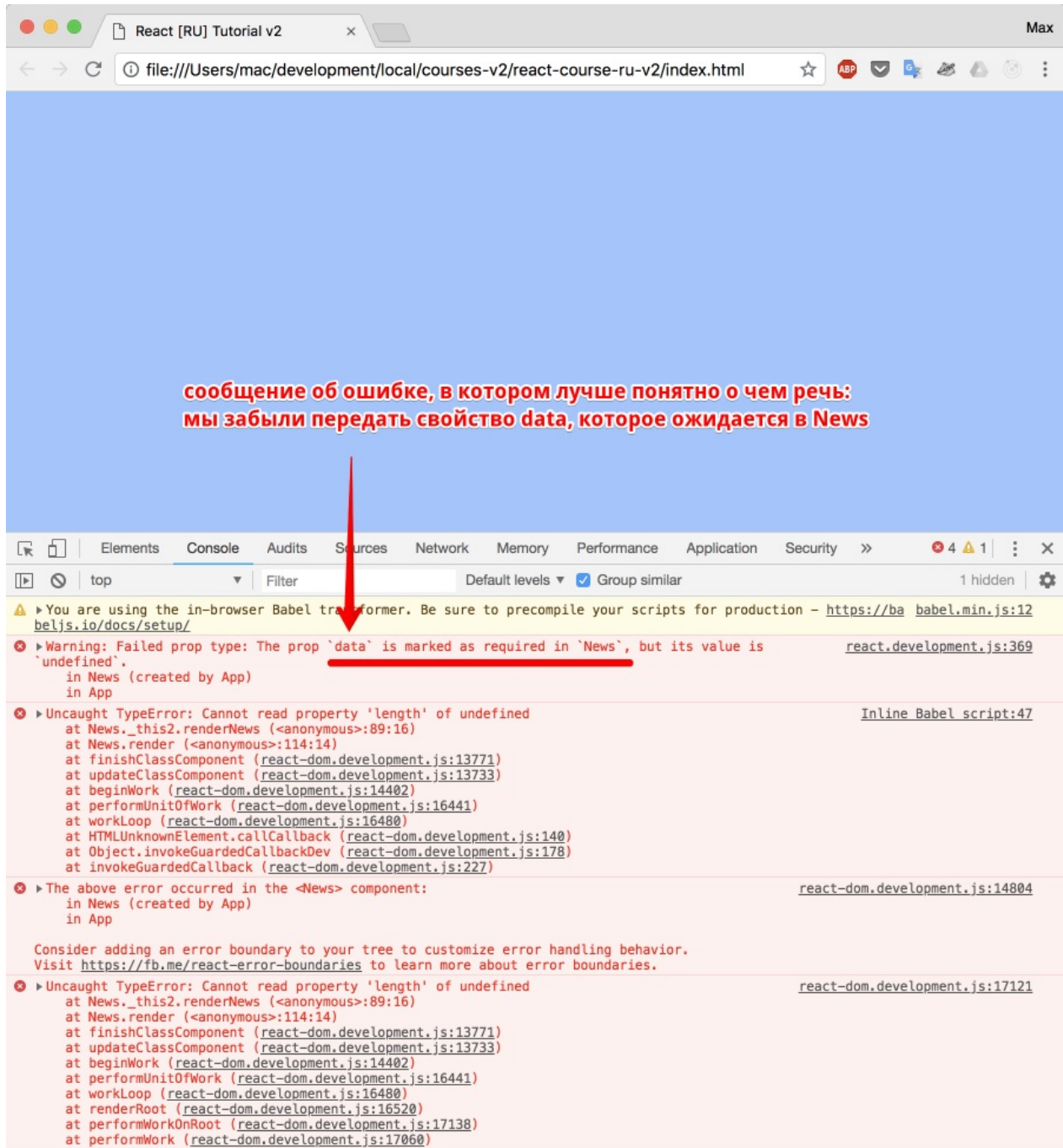
Пусть вас не смущает версия 15.6 у пакета `prop-types`. Он с некоторых пор живет своей жизнью, отдельно от `react`.

Внесите изменения в компонент `<News />`

```
...
class News extends React.Component {
  renderNews = () => {
    ...
  }
  render() {
    ...
  }
}

// добавили propTypes.
// propTypes (с маленькой буквы) = свойство News
News.propTypes = {
  data: PropTypes.array.isRequired // PropTypes (с большой буквы) = библиотека prop-ty
  pes
}
...
```

Обновите страницу:



Гораздо лучше! Исходя из текста ошибки нам сразу понятно куда копать: в render методе App, не указано свойство data, которое ожидается в News. Справедливости ради, ниже есть подобное сообщение, но ошибка там имеет универсальный текст.

Подробные "простыни" текста об ошибке - заслуга новых версий реакта. Это удобно.

Восстановим свойство data.

```
const App = () => {  
  return (  
    <React.Fragment>  
      <h3>Новости</h3>  
      <News data={myNews}/>  
    </React.Fragment>  
  )  
}
```

Вновь все работает, и наша консоль чиста.

---

## Подробнее о propTypes

Приведу выдержку из [офф.документации](#):



```
MyComponent.propTypes = {
  propTypes: {
    // Вы можете указать, каким примитивом должно быть свойство
    optionalArray: PropTypes.array,
    optionalBool: PropTypes.bool,
    optionalFunc: PropTypes.func,
    optionalNumber: PropTypes.number,
    optionalObject: PropTypes.object,
    optionalString: PropTypes.string,
    optionalSymbol: PropTypes.symbol,

    // ...

    // Вы можете указать, что свойство может быть одним из ...
    optionalUnion: PropTypes.oneOfType([
      PropTypes.string,
      PropTypes.number,
      PropTypes.instanceOf(Message)
    ])

    // ...

    // Вы можете указать, конкретную структуру объекта свойства
    optionalObjectWithShape: PropTypes.shape({
      color: PropTypes.string,
      fontSize: PropTypes.number
    }),

    // Вы можете указать, что свойство ОБЯЗАТЕЛЬНО
    requiredFunc: React.PropTypes.func.isRequired,

    // Если нужно указать, что свойство просто обязательно, и может быть любым примитивом
    requiredAny: React.PropTypes.any.isRequired,

    // ... (в документации еще есть варианты)
  }
};
```

Согласно этому листингу, мы можем перевести правило, указанное в компоненте

```
<News /> :
```

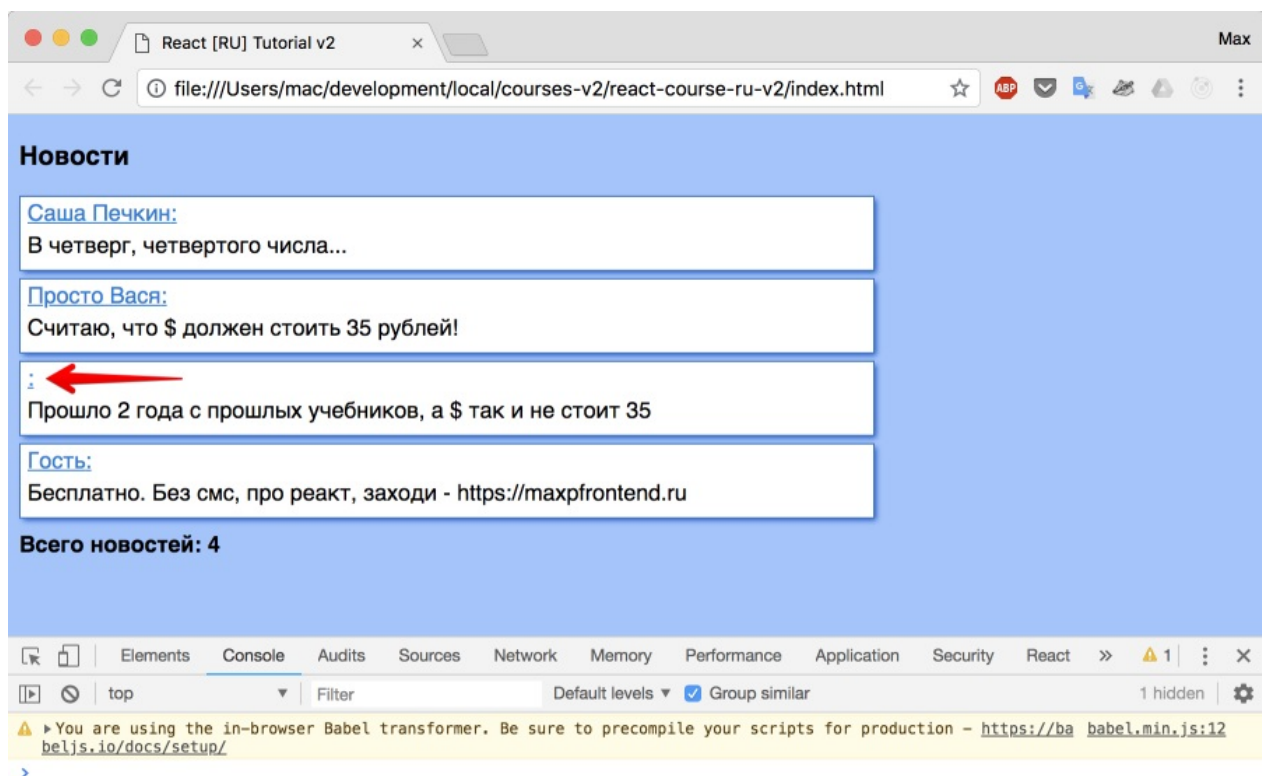
`PropTypes.array.isRequired` - свойство должно быть массивом и оно обязательно должно быть!

Я вижу по глазам некоторых (да-да, вижу), что все это какая-то бесполезная лабуда. И так понятно - есть ошибка, есть возможность тыкнуть на нее в дебаггере и посмотреть. Специально для вас, следующая ситуация: удалите из массива `myNews`

автора, например в третьем элементе:

```
const myNews = [
  {
    id: 1,
    author: 'Саша Печкин',
    text: 'В четверг, четвертого числа...',
    bigText: 'в четыре с четвертью часа четыре чёрненьких чумазеньких чертёнка чертили чёрными чернилами чертёж.'
  },
  {
    id: 2,
    author: 'Просто Вася',
    text: 'Считаю, что $ должен стоить 35 рублей!',
    bigText: 'А евро 42!'
  },
  {
    id: 3,
    // удалили автора
    text: 'Прошло 2 года с прошлых учебников, а $ так и не стоит 35',
    bigText: 'А евро опять выше 70.'
  },
  {
    id: 4,
    author: 'Гость',
    text: 'Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru',
    bigText: 'Еще есть группа VK, telegram и канал на youtube! Вся инфа на сайте, не р
еклама!'
  }
];
```

Посмотрим результат:



Никаких ошибок. Но наше приложение **не работает** так как надо. Кто виноват? Реакт? Backend-программист который прислал нам такие данные?

Программист, может и виноват. Но реакт точно нет. У нас получилось, что в `this.props.data.author` содержится `undefined` (переменная не определена). Поэтому `react` так и поступил, и показал нам "ничего" (на скриншоте это только "двоеточие").

The screenshot shows a web browser displaying a news application. The page has a blue header with the word "Новости" (News). Below it, there are four news items, each with an author and a text. The third item, "Просто Вася", has a text that says "Прошло 2 года с прошлых учебников, а \$ так и не стоит 35". A red arrow points from the text "значение this.props.data.author" to the author field of the third item. Another red arrow points from the text "шаблон отрисовал НИЧЕГО плюс двоеточие" to the text of the third item. Below the news items, it says "Всего новостей: 4".

The React DevTools component inspector is open, showing the component tree on the left and the props for the selected component on the right. The component tree shows the following structure:

```
<App>
  <h3>Новости</h3>
  <News data={...}>
    <div className="news">
      <Article key="1" data={id: 1, author: "Саша Печкин", text: "В четверг, четвертого числа...">
        ...
      </Article>
      <Article key="2" data={id: 2, author: "Просто Вася", text: "Считаю, что $ должен стоить 35 р">
        ...
      </Article>
      <Article key="3" data={id: 3, text: "Прошло 2 года с прошлых учебников, а $ так и не ст...">
        <div className="article">
          <p className="news__author">
            ":"
          </p>
          <p className="news__text">Прошло 2 года с прошлых учебников, а $ так и не стоит 35</p>
        </div>
      </Article>
      <Article key="4" data={id: 4, author: "Гость", text: "Бесплатно. Без смс, про реакт, заходи">
        ...
      </Article>
      <strong className="news__count">
        "Всего новостей: "
        "4"
      </strong>
    </div>
  </News>
</App>
```

The props for the selected component (Article) are shown on the right:

```
Key "3"
Props
  data: {...}
    bigText: "А евро опять выше 70."
    id: 3
    text: "Прошло 2 года с прошлых учебников, а $ так и не стоит 35"
```

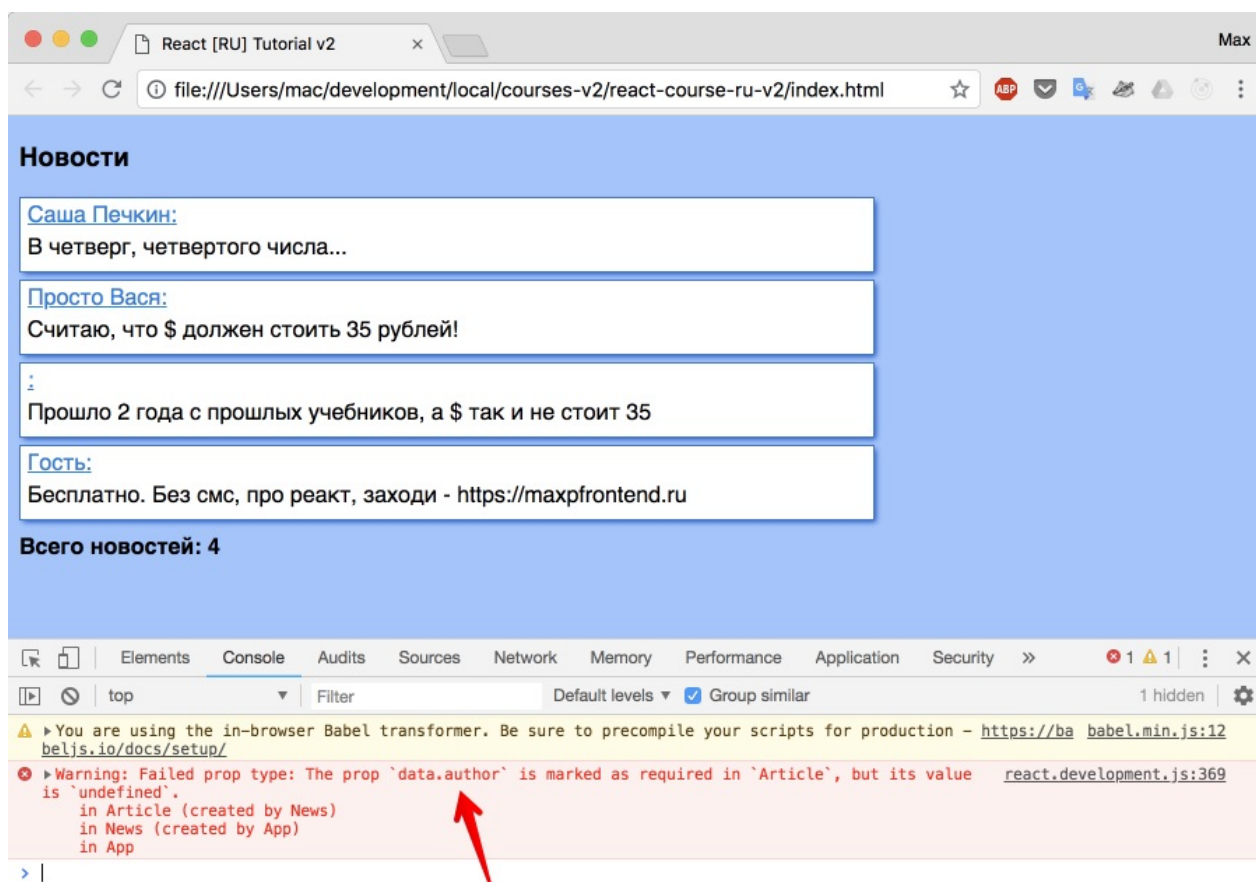
A red arrow points from the text "свойство author отсутствует (то есть равно undefined)" to the props object, indicating that the author property is missing.

Такую ошибку сложно отловить.

Добавьте `propTypes` в компонент `<Article />`

```
class Article extends React.Component {  
  render() {  
    ...  
  }  
}  
  
Article.propTypes = {  
  data: PropTypes.shape({  
    author: PropTypes.string.isRequired,  
    text: PropTypes.string.isRequired  
  })  
}
```

В таком случае вы получите сообщение об ошибке:



**свойство data.author помечено как "Обязательное", но его значение undefined**

Разве это не прекрасно?

[Исходный код](#) на данный момент.

P.S. Не забудьте вернуть автора ;)



# Использование state

Вернемся от теории к практике: давайте покликаем по ссылкам-кнопочкам, поизменяем свойства компонентов...

Упс, не выйдет! Как вы помните, свойства (*this.props*) следует использовать только для чтения, а для динамических свойств нужно использовать так называемое "состояние" (*state*).

Итак, встречайте - **this.state** ;)

Так как мне нужно в этом разделе сохранить минимум теории и больше практики, сразу перейдем к делу. Предлагаю вместе решить следующую задачу: *у новости есть ссылка "подробнее", по клику на которую - бинго, текст новости целиком.*

---

Начнем с изменения данных:

```
const myNews = [
  {
    id: 1,
    author: 'Саша Печкин',
    text: 'В четверг, четвертого числа...',
    bigText: 'в четыре с четвертью часа четыре чёрненьких чумазеньких чертёнка чертили чёрными чернилами чертёж.'
  },
  {
    id: 2,
    author: 'Просто Вася',
    text: 'Считаю, что $ должен стоить 35 рублей!',
    bigText: 'А евро 42!'
  },
  {
    id: 3,
    author: 'Max Frontend',
    text: 'Прошло 2 года с прошлых учебников, а $ так и не стоит 35',
    bigText: 'А евро опять выше 70.'
  },
  {
    id: 4,
    author: 'Гость',
    text: 'Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru',
    bigText: 'Еще есть группа VK, telegram и канал на youtube! Вся инфа на сайте, не р еклама!'
  }
];
```

Затем, научимся отображать полный текст новости сразу после вводного текста:

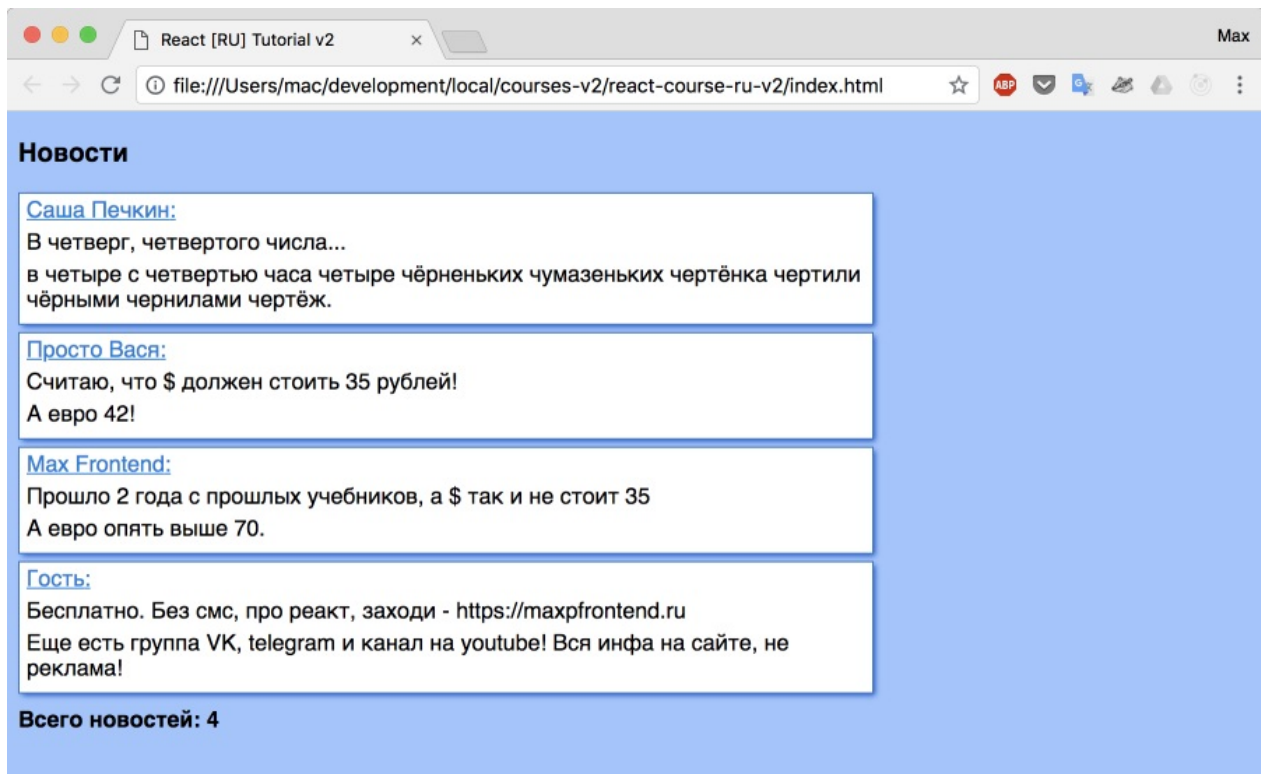


```
class Article extends React.Component {
  render() {
    const { author, text, bigText } = this.props.data // вытащили bigText из даты
    return (
      <div className='article'>
        <p className='news__author'>{author}</p>
        <p className='news__text'>{text}</p>
        <p className='news__big-text'>{bigText}</p>
      </div>
    )
  }
}

Article.propTypes = {
  data: PropTypes.shape({
    author: PropTypes.string.isRequired,
    text: PropTypes.string.isRequired,
    bigText: PropTypes.string.isRequired // добавили propTypes для bigText
  })
}
```

Опять же, больше ничего изменять не нужно. Данные корректно отобразятся.

Проверим...



Отлично, можно продолжать работу: добавим ссылку - "подробнее". Приведу фрагмент кода:

```
...
return (
  <div className='article'>
    <p className='news__author'>{author}</p>
    <p className='news__text'>{text}</p>
    <a href="#" className='news__readmore'>Подробнее</a>
    <p className='news__big-text'>{bigText}</p>
  </div>
)
...
```

Проверьте и если все ок - мы готовы к работе с состоянием компонента.

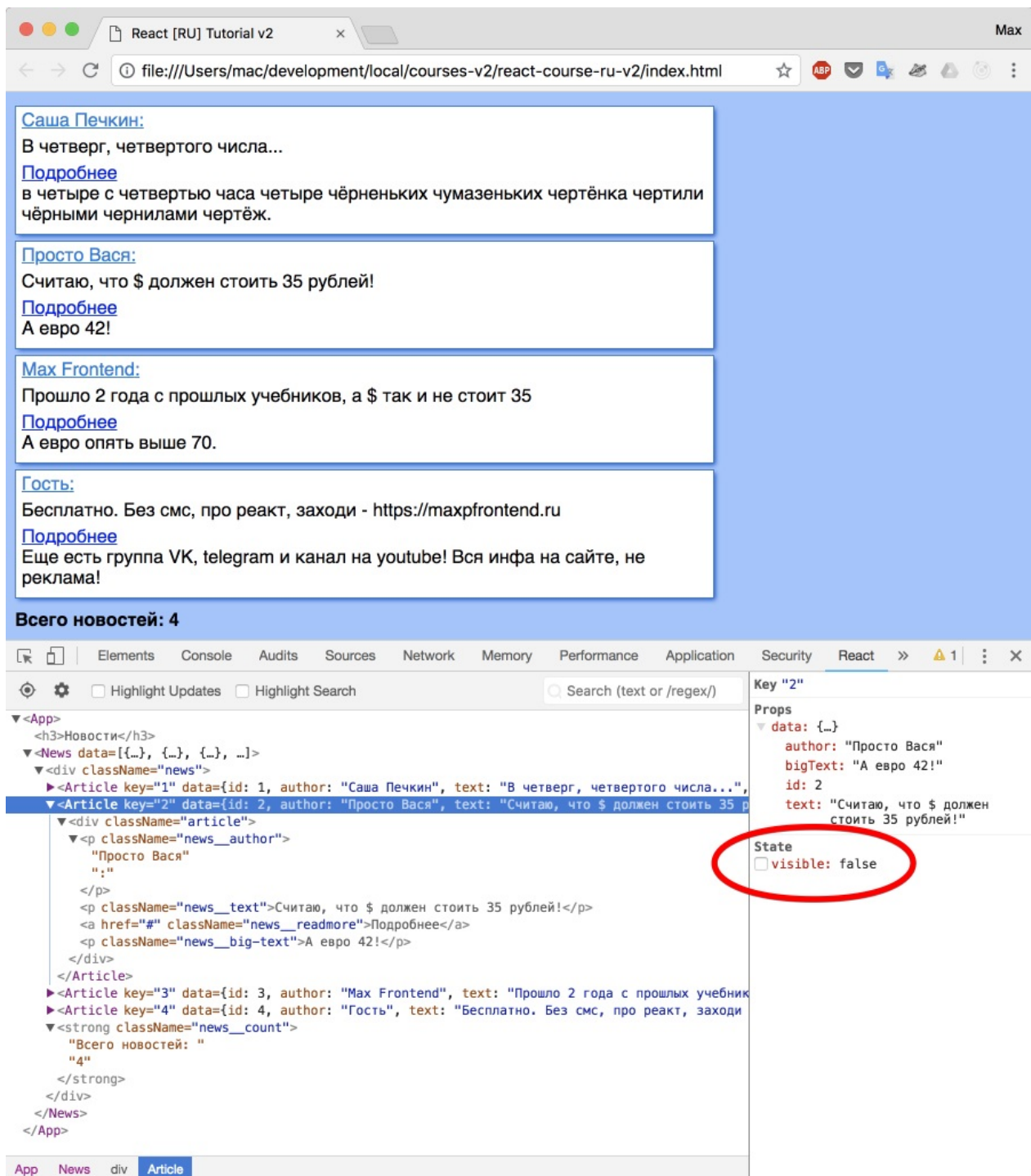
## Начальное состояние

*С состоянием (state) можно работать только в statefull компонентах (class)*

Если вы определяете какое-то изменяемое свойство в компоненте, необходимо указать начальное состояние (в терминологии react.js - *initial state*). Для этого, у компонента нужно просто определить свойство state:

```
class Article extends React.Component {
  state = {
    visible: false, // определили начальное состояние
  }
  render() {
    const { author, text, bigText } = this.props.data
    return (
      <div className='article'>
        <p className='news__author'>{author}</p>
        <p className='news__text'>{text}</p>
        <a href="#" className='news__readmore'>Подробнее</a>
        <p className='news__big-text'>{bigText}</p>
      </div>
    )
  }
}
```

Посмотрим в консоли на вкладке React:



В состоянии (в *state*) появилось свойство. Будем использовать его в момент `render'a`.

Формализуем задачу:

- если значение `this.state.visible === false` -> рисуй "подробнее", не рисуй "большой текст";
- если же `this.state.visible === true` -> не рисуй "подробнее", рисуй большой текст;

Будем использовать логическое выражение внутри JSX, значит - фигурные скобки, и внутри js-выражение.

```
class Article extends React.Component {
  state = {
    visible: false,
  }
  render() {
    const { author, text, bigText } = this.props.data
    const { visible } = this.state // вытащили visible из this.state
    return (
      <div className='article'>
        <p className='news__author'>{author}</p>
        <p className='news__text'>{text}</p>
        { /* если не visible, то показывай */
          !visible && <a href="#" className='news__readmore'>Подробнее</a>
        }
        { /* если visible, то показывай */
          visible && <p className='news__big-text'>{bigText}</p>
        }
      </div>
    )
  }
}
```

Мы использовали одну и ту же переменную состояния в **двух** местах когда описывали обычное javascript выражение. Если вы не знакомы с И / ИЛИ, то рекомендую почитать у Кантора ([Логические операторы](#)).

Обратите внимание, для написания комментариев не понадобились дополнительные фигурные скобки, так как мы уже находились внутри "выражения" внутри JSX.

Можете проверить в браузере, и покликать на чекбокс внутри state зоны. Шаблон уже будет реагировать. А мы продолжим делать все это по-человечески, чтобы можно было кликать на "подробнее".

---

## Обработка кликов - onClick

Чтобы обработать клик, нам необходимо указать атрибут *onClick* у элемента.

В качестве обработчика у нас будет функция, которая изменяет состояние. Для изменения состояния, нужно **обязательно** использовать метод **setState**, а не перезаписывать значение переменной в `this.state` напрямую.

```
class Article extends React.Component {
  state = {
    visible: false,
  }
  handleReadMoreClick = (e) => { // добавили метод
    e.preventDefault()
    this.setState({ visible: true })
  }
  render() {
    const { author, text, bigText } = this.props.data
    const { visible } = this.state
    return (
      <div className='article'>
        <p className='news__author'>{author}</p>
        <p className='news__text'>{text}</p>
        { /* добавили onClick */
          !visible && <a onClick={this.handleReadMoreClick} href="#" className='news__r
eadmore'>Подробнее</a>
        }
        {
          visible && <p className='news__big-text'>{bigText}</p>
        }
      </div>
    )
  }
}
```

Проверьте в браузере, кликните на ссылку "подробнее".

Каждый компонент `<Article />` имеет свое состояние! Поэтому, при клике на подробнее в одном из компонентов, только его состояние изменяется, и только у этой новости отображается полный текст.

The screenshot shows a web browser window with the URL `file:///Users/mac/development/local/courses-v2/react-course-ru-v2/index.html`. The page displays a list of four news items under the heading "Новости". Each item has an author, a text snippet, and a "Подробнее" link. The fourth item, by "Гость", is highlighted with a red arrow. Below the list, it says "Всего новостей: 4".

Below the browser window, the React DevTools component inspector is open. The left pane shows the component tree with the following structure:

```
<App>  
  <h3>Новости</h3>  
  <News data={[...], {...}, {...}, ...]>  
    <div className="news">  
      <Article key="1" data={id: 1, author: "Саша Печкин", text: "В четверг, четвертого числа...", ...}>  
      <Article key="2" data={id: 2, author: "Просто Вася", text: "Считаю, что $ должен стоить 35 рублей!", ...}>  
      <Article key="3" data={id: 3, author: "Max Frontend", text: "Прошло 2 года с прошлых учебников, а $ так и не стоит 35", ...}>  
      <Article key="4" data={id: 4, author: "Гость", text: "Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru", ...}>  
        <div className="article">  
          <p className="news__author">Гость</p>  
          <p className="news__text">Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru</p>  
          <p className="news__big-text">Еще есть группа VK, telegram и канал на youtube! Вся инфа на сайте, не реклама!</p>  
        </div>  
      <strong className="news__count">Всего новостей: 4</strong>  
    </div>  
  </News>  
</App>
```

The right pane shows the props and state for the selected `Article` component (key="4"). The state is:

```
{  
  visible: true  
}
```

Red arrows point from the news item in the browser to the corresponding entry in the component tree and its state in the DevTools.

## Итого:

Для сохранения динамических свойств, используется состояние (*state*) компонента.

Для обработки клика, используется свойство `onClick` + функция-обработчик.

Существуют и другие стандартные события, которые работают по такому же принципу.

Полный список [здесь](#). Мы будем знакомиться с ними по мере необходимости.

Для изменения состояния, используется метод **setState**, который принимает объект с аргументами, которые нужно изменить. Например, у нас есть состояние:

```
...
state = {
  visible: false,
  rating: 0,
  eshe_odno_svoistvo: 'qweqwe'
}
...
```

Чтобы изменить рейтинг, нужно написать следующий setState:

```
this.setState({rating: 100500})
```

Чтобы изменить все три свойства:

```
this.setState({
  rating: 100500,
  visible: true,
  eshe_odno_svoistvo: 'привет'
})
```

Так же у *setState* есть возможность указать *callback* функцию, которая будет вызвана после того, как новое состояние "установится".

```
...
readmoreClick: function(e) {
  e.preventDefault();
  this.setState({ visible: true }, () => {
    alert('Состояние изменилось');
  });
},
...
```

---

Еще *setState* есть возможность... (на самом деле есть, но с нас пока хватит).

Посмотреть полный список возможностей setState можно в [документации](#).

[Исходный код](#) на данный момент.

## Подробнее о state

В этом разделе мы посмотрим как изменение *state* влияет на компонент и немного "зацепим" *stateless* архитектуру.

---

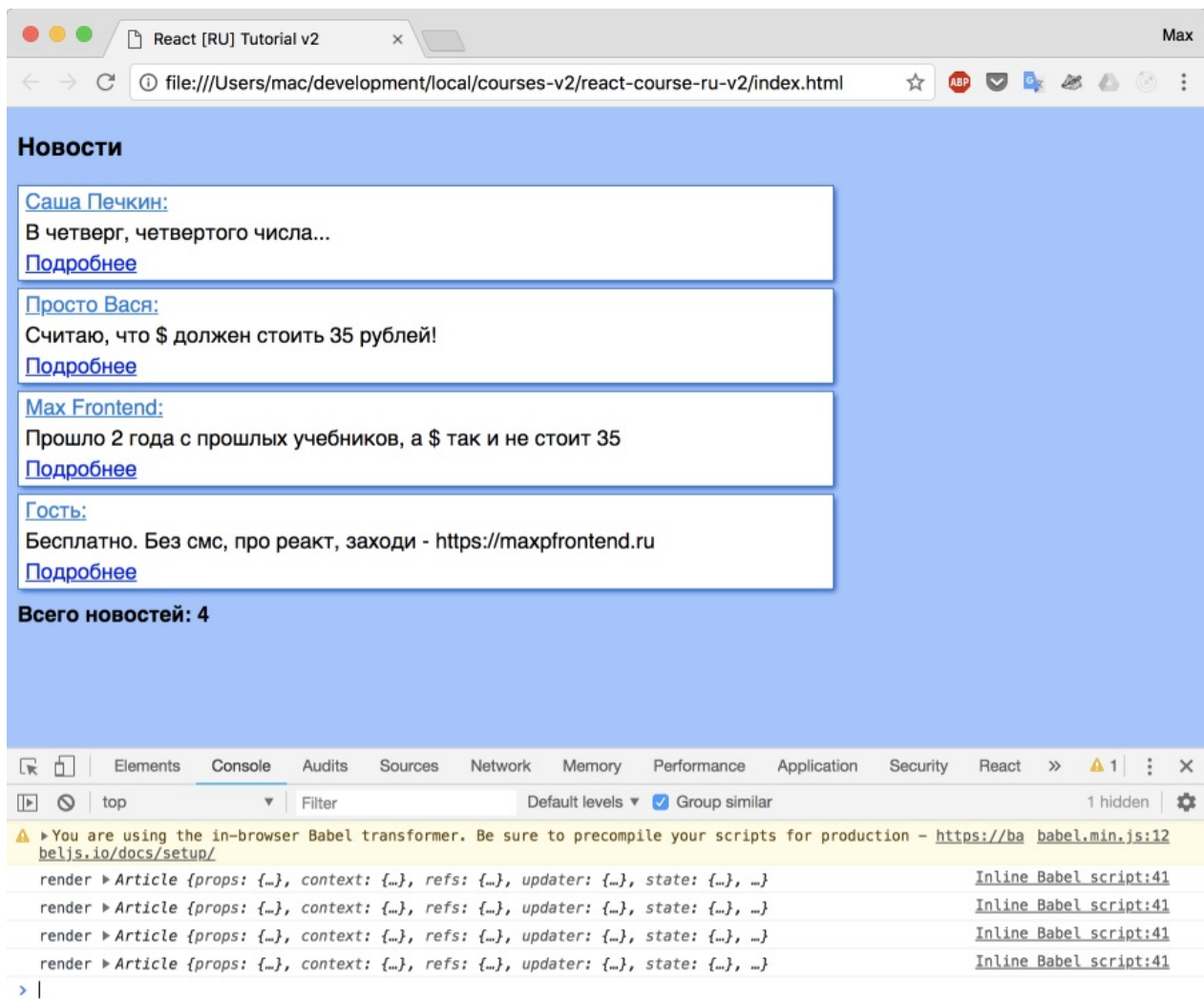
### Изменение state вызывает render компонента

Все указано в подзаголовке, предлагаю нам в этом убедиться:

Фрагмент компонента `<Article />` :

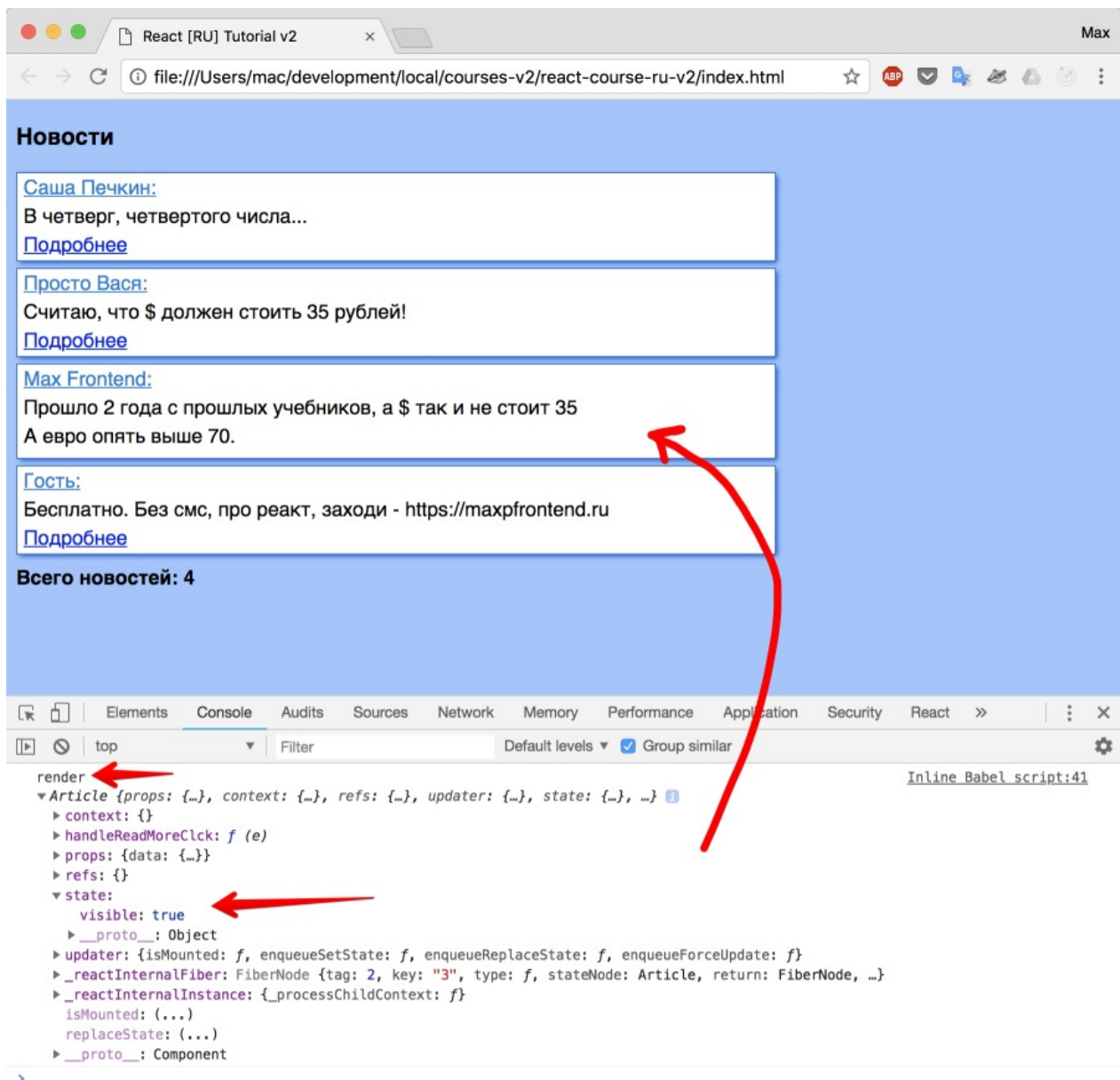
```
...
render() {
  const { author, text, bigText } = this.props.data
  const { visible } = this.state
  console.log('render', this); // добавили console.log
  return (
    <div className='article'>
      <p className='news__author'>{author}</p>
      <p className='news__text'>{text}</p>
      {
        !visible && <a onClick={this.handleReadMoreClick} href="#" className='news__readmore'>Подробнее</a>
      }
      {
        visible && <p className='news__big-text'>{bigText}</p>
      }
    </div>
  )
}
...
```





**4 компонента Article - 4 рендера**

Очистите консоль, и нажмите подробнее на любой из новостей:



Убедились? Несколько правил:

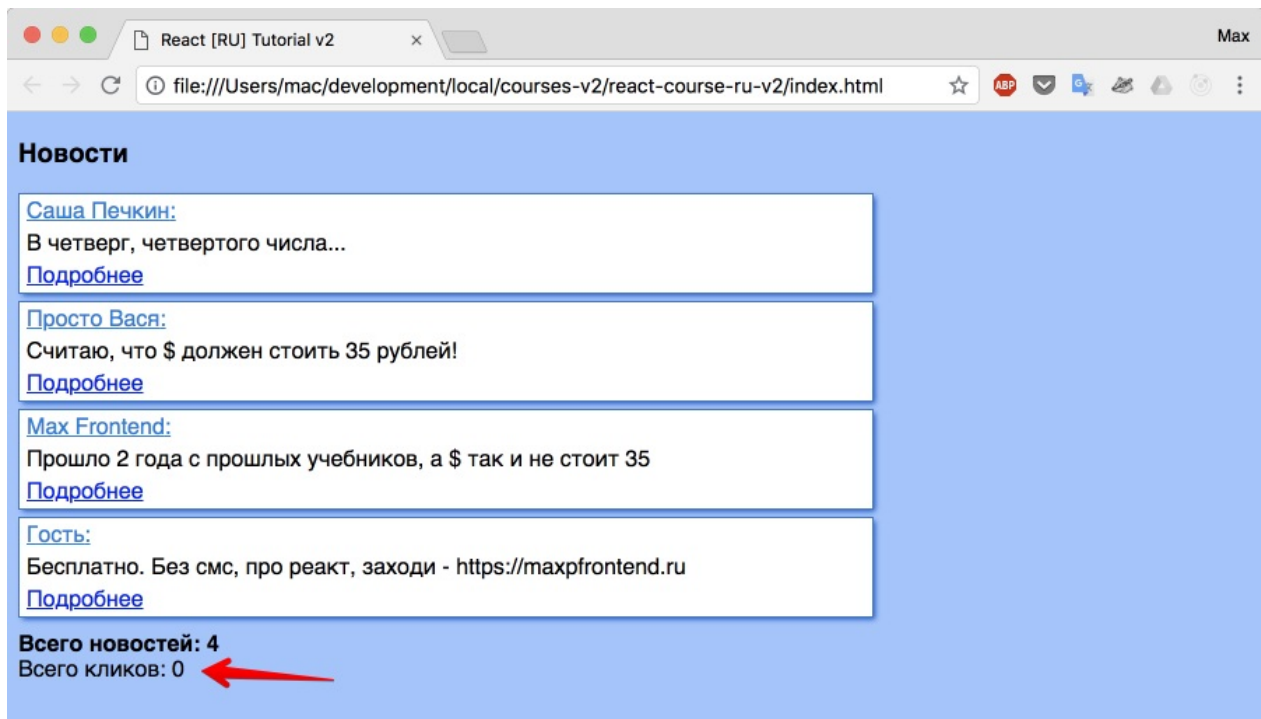
- **нельзя** вызывать `setState` внутри `render`: "реакт бдит", и если изменилось состояние - начинает перерисовывать компонент - видит что изменилось состояние - начинает перерисовывать компонент...
- `render` - **дорогостоящая** операция, поэтому внимательно относитесь к тому, где вы вызываете `setState`, и что это за собой влечет. Банальные `console.log` могут вам в этом помочь.

Очевидно, что если перерисовывается родительский компонент, то будут перерисованы и все дочерние компоненты.

В дальнейшем мы изучим разные "стадии жизни" компонента, и убедимся, что во время его "перерисовки" могут выполняться разные дорогостоящие операции и даже аякс-запросы. Пока что, просто убедимся, что вызов `setState` родителя - перерисует дочерние компоненты. Для этого предлагаю создать обработчик `onClick` на фразе "Всего новостей".

Попробуйте сами.

**Задача:** Необходимо добавить компоненту `<News />` свойство состояния - `counter`, в котором будет храниться количество кликов по фразе "всего новостей". То есть обычный автоинкремент. Это свойство нужно выводить после количества новостей в обычном параграфе ( `<p>` ). Будет выглядеть так:



В решении важно использовать `this.setState({counter: ++this.state.counter})`, об этом мы подробнее поговорим после решения, которое представлено ниже в виде подсказок и полностью.

Подсказка **#1**: добавьте свойство `state` в компонент `<News />` для создания начального состояния.

```
...
state = {
  counter: 0
}
...
```

Подсказка **#2**: добавьте обработчик *onClick* с функцией, которая будет увеличивать счетчик (следовательно, изменять *state*, следовательно, вызывать *this.setState...*).

**Решение:** Полный код компонента `<News />`

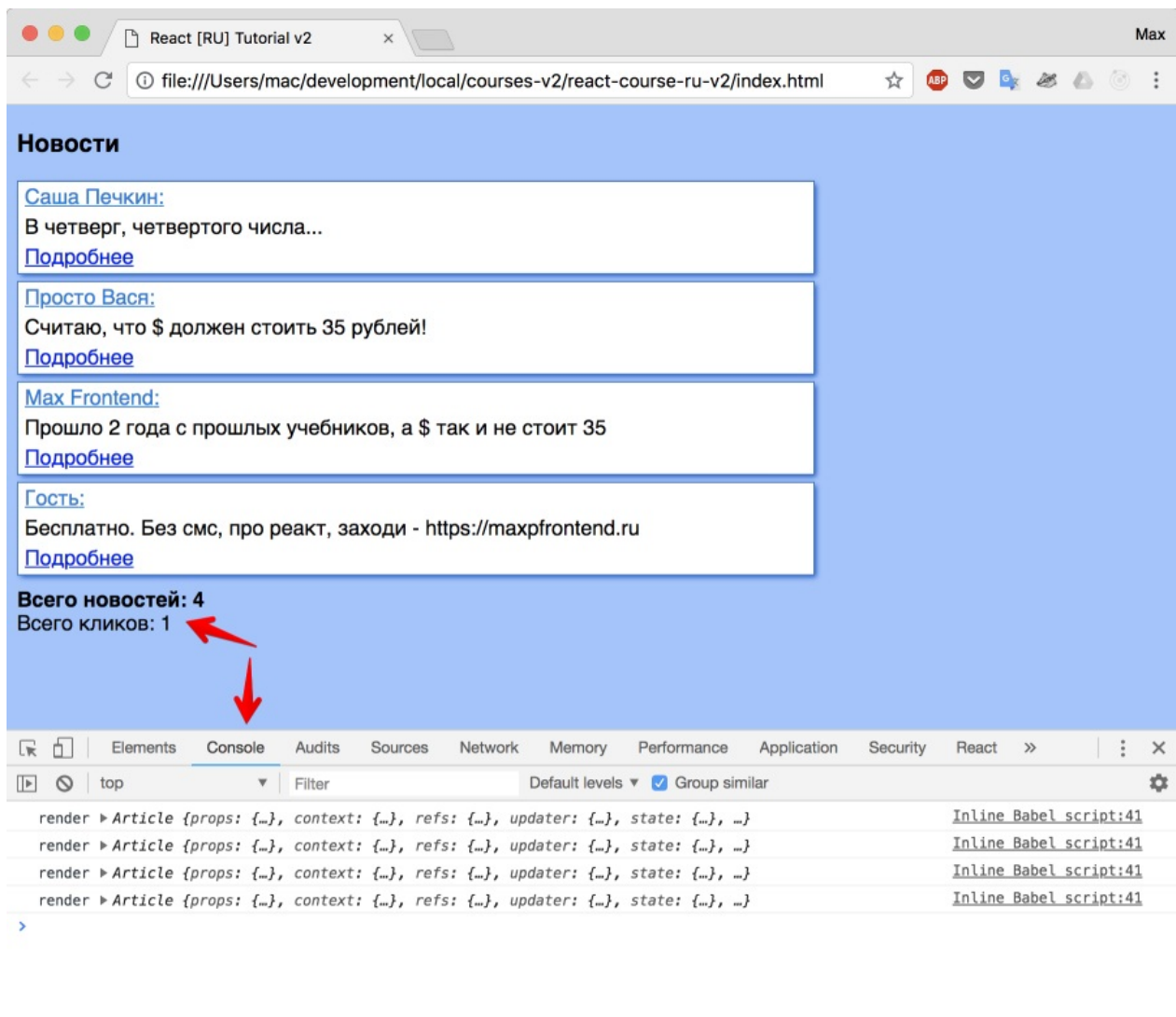
```
class News extends React.Component {
  state = {
    counter: 0, // добавили свойство counter (счетчик)
  }
  handleCounter = () => { // добавили новый метод
    this.setState({ counter: ++this.state.counter }) // в котором увеличиваем счетчик
  }
  renderNews = () => {
    const { data } = this.props
    let newsTemplate = null

    if (data.length) {
      newsTemplate = data.map(function(item) {
        return <Article key={item.id} data={item}/>
      })
    } else {
      newsTemplate = <p>К сожалению новостей нет</p>
    }

    return newsTemplate
  }
  render() {
    const { data } = this.props
    const { counter } = this.state // вытащили counter

    return (
      <div className='news'>
        {this.renderNews()}
        { /* добавили onClick */
          data.length ? <strong onClick={this.handleCounter} className='news__count'>
Всего новостей: {data.length}</strong> : null
          }
        <p>Всего кликов: { counter }</p>
      </div>
    );
  }
}
```

Проверьте в браузере. Если вы не удаляли `console.log` из компонента `<Article />` - на каждый клик по фразе "Всего новостей", в консоли будет появляться по 4 "перерисовки".



Поговорим о: `this.setState({counter: ++this.state.counter })`

Почему же, было важно использовать именно префиксную запись `++`, а не постфиксную? Сначала вспомним теорию:

- `++` перед переменной (префикс) - сначала увеличивает ее на 1, а потом возвращает значение;
- `++` после переменной (постфикс) - сначала вернет значение, а потом увеличит значение переменной;

В таком случае, мы должны были бы потерять всего 1 клик, не так ли? Проверьте в консоли следующим образом: откройте вкладку React в консоли, выберите компонент `<News />`, начните кликать на фразу "Всего новостей"

- Изменяется ли значение `counter`, если используется префиксная запись?

Да, изменяется

- Изменяется ли значение `counter`, если используется постфиксная запись?

Нет, не изменяется вообще. (убедитесь сами)

Ответ кроется в [официальной документации](#):

*setState() - не изменяет this.state немедленно, а создает очередь изменений состояния. Доступ к this.state после вызова метода, потенциально может вернуть имеющееся (что равносильно - бывшее) значение.*

Самое время крикнуть - верните мои деньги назад, и уйти... Но, не все так плачевно. Теперь вы знаете об этой особенности, и будете если что вооружены. Зачем так сделано? Вероятно, для оптимизации работы библиотеки в целом.

---

Вообще *state* у компонентов используется не часто. С появлением [flux](#) - подхода (здесь я проследил и не стал переписывать в 2018м году), комьюнити стало перемещаться на сторону *stateless* подхода, когда *state* не используется вообще (за исключением редких моментов). Мой любимый игрок данного лагеря - **Redux**, о котором я тоже написал [подробное руководство](#) на русском (руководство еще не переписано на современный лад, но теорию почитать можно).

Почему сейчас мы не изучаем Redux, стоит ли бросить все и прочитать другой tutorial? Определенно - нет. Продолжайте изучение данного курса.

---

## Ситуация в 2018м году:

- Есть не только Redux, но и MobX и прочие игроки. Однако Redux до сих пор остается моим любимым;
  - React для управления состоянием приложения в целом, добавил Context API, о котором мы еще поговорим;
  - Чистый flux не используется;
- 

[Исходный код](#) с console.log'ами и обработчиком кликов на фразе "Всего новостей".

# Работа с input

Сперва приберемся:

Удалим лишние `console.log`'и, удалим обработчик `handleCounter` и параграф, который выводил количество кликов.

Затем создадим компонент - `<TestInput />`, который будет просто отрисовывать (render) input перед списком новостей.

```
...
// --- добавили test input ---
class TestInput extends React.Component {
  render() {
    return (
      <input className='test-input' value='введите значение' />
    )
  }
}

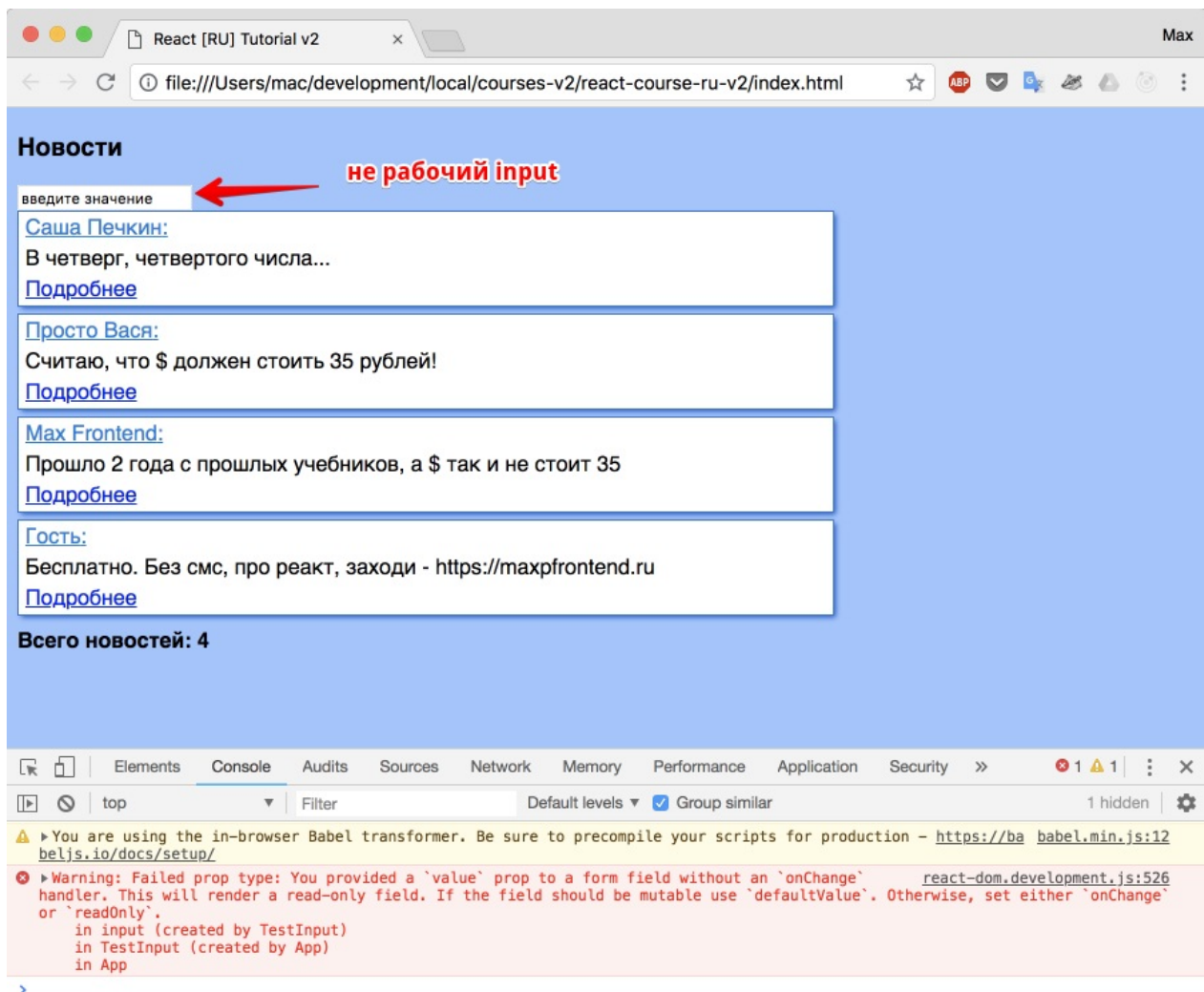
const App = () => {
  return (
    <React.Fragment>
      <h3>Новости</h3>
      <TestInput /> { /* добавили вывод компонента */ }
      <News data={myNews}/>
    </React.Fragment>
  )
}
...
```

Напомню про комментарии:

Первый комментарий, добавлен с помощью `//`, так как данный комментарий не находится внутри JSX. А второй - находится, следовательно имеет вид `{/* комментарий */}`. То есть, JSX - это не весь код вашего сценария, а только те части, где мы миксуем верстку и js (обычно в `return` методе у `render`'а).

Вообще, код сейчас не работает (но это не из-за комментария). Давайте посмотрим на ошибку внимательно:





Вы предоставили свойство `value` для поля, у которого нет `onChange` обработчика. Поэтому отрисовано поле только для чтения. Если поле должно быть изменяемо, используйте `defaultValue`. Либо установите `onChange` или `readOnly`. Проверьте `render` метод компонента `TestInput`.

Не могу не любить react за такие подробные сообщения об ошибках.

А вы кстати попробуйте сейчас изменить значение инпута. Ничего не выйдет. Здесь у нас есть два пути, и первый нам известный - использовать какое-нибудь свойство `state` в качестве динамически изменяемого значения инпута.

## Controlled components (контролируемые компоненты)

Для вызова `setState`, будем использовать событие `onChange`. Работа с ним не отличается от работы с `onClick` или другими любыми событиями. Главное - передать функцию-обработчик.

Не торопитесь, давайте подумаем еще раз:



1. Нам нужно передать функцию обработчик, которая будет изменять какую-то переменную состояния (с помощью *setState*, разумеется).
2. Значит нам нужно создать начальное состояние (*state*).
3. Если у нас есть переменная состояния компонента, значит мы хотим, чтобы именно она была в качестве *value* у нашего инпута.

Сможете сделать сами? Если да - отлично, если нет - решение ниже.

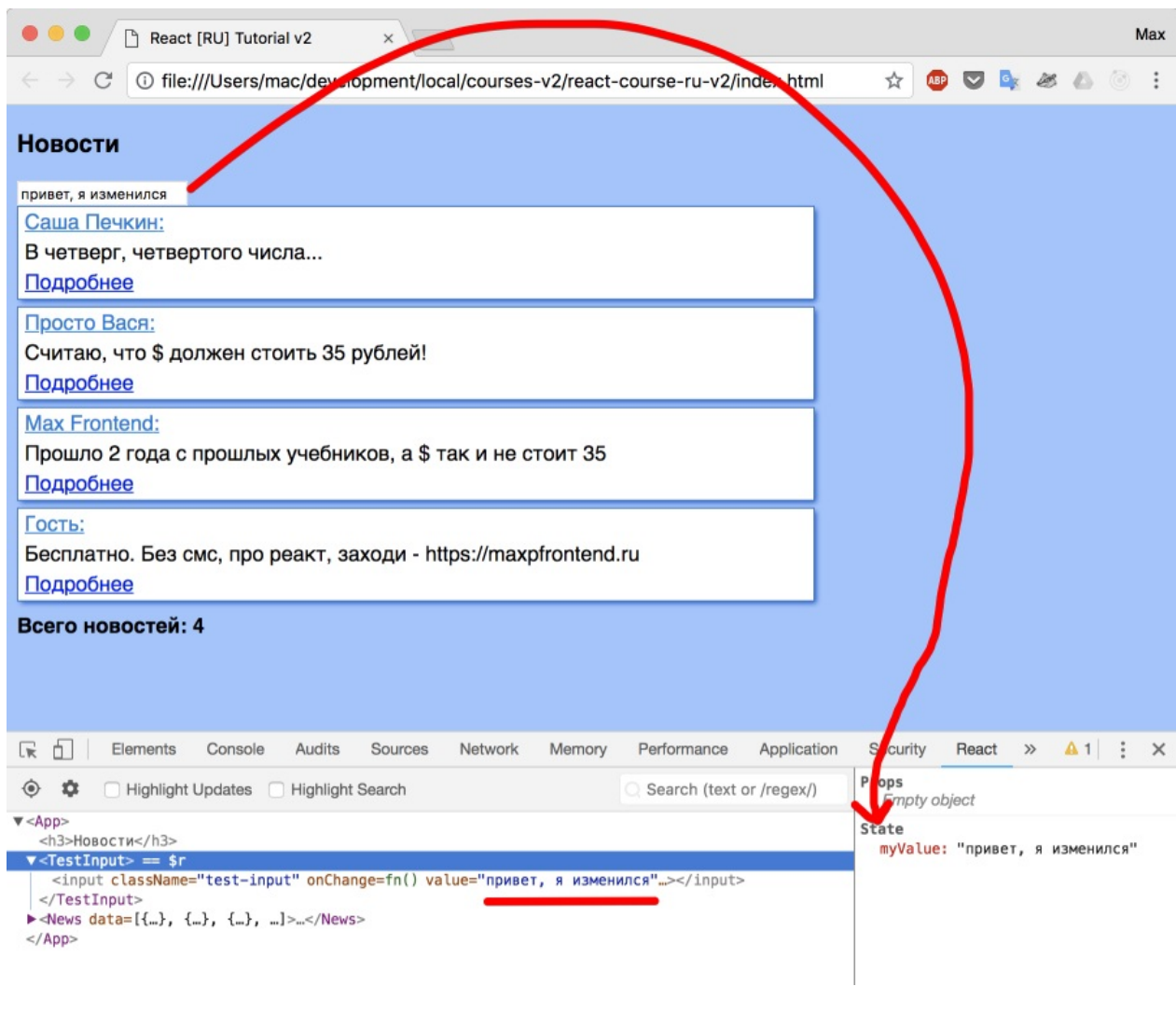
Подсказка **#1**: так может выглядеть состояние + функция-обработчик

```
...
state = { myValue: '' }
...
onChangeListener = (e) => {
  this.setState({ myValue: e.target.value })
},
...
```

### Решение:

```
class TestInput extends React.Component {
  state = {
    myValue: '',
  }
  // используется e.currentTarget.value
  onChangeHandler = (e) => {
    this.setState({ myValue: e.currentTarget.value })
  }
  render() {
    return (
      <input
        className='test-input'
        onChange={this.onChangeHandler}
        value={this.state.myValue}
        placeholder='введите значение' />
    )
  }
}
```

У нас есть *placeholder* - "введите значение", который будет показываться в момент загрузки страницы, так как наше начальное состояние input'a - пустая строка. При изменении, мы устанавливаем в переменную *myValue* - то что введено в input. Следовательно - input корректно изменяется.



## e.currentTarget vs e.target

Документация, что за свойство [currentTarget](#) (MDN)

Представьте ситуацию: у вас будет `onClick` стоять на `div`, внутри которого будет текст в параграфе. В итоге, при клике на текст в параграфе:

- `e.target` будет равно параграфу (и это верно, по спецификации)
- `e.currentTarget` будет равно `div`'у (и это то, что нам обычно и нужно)

Кто вообще не понял о чем идет речь - нужно читать [основы про объект события](#) или весь раздел [основы работы с событиями](#) целиком (Кантор).

Обычно, мы хотим по клику отправлять значения инпута...

**Задача:** По клику на кнопку - показывать `alert` с текстом инпута.

Попробуйте сами.

**Подсказка #1:**

Вам необходимо:

- добавить кнопку в `<TestInput />` ;
- на кнопку "повесить" обработчик `onClick`;
- в функции обработчике считывать значение `this.state.myValue` ;

**Подсказка #2:**

Так как нам необходимо рендерить больше одного элемента, нужно обернуть их в родительский элемент, например в `<div></div>` или `React.Fragment`

**Решение:**

```
class TestInput extends React.Component {
  state = {
    myValue: '',
  }
  onChangeHandler = (e) => {
    this.setState({ myValue: e.currentTarget.value })
  }
  onBtnClickHandler = (e) => {
    alert(this.state.myValue);
  }
  render() {
    return (
      <React.Fragment>
        <input
          className='test-input'
          onChange={this.onChangeHandler}
          value={this.state.myValue}
          placeholder='введите значение' />
        <button onClick={this.onBtnClickHandler}>Показать alert</button>
      </React.Fragment>
    )
  }
}
```

Предлагаю добавить отступы для `.test-input`:

`css/app.css`

```
...
.test-input {
  margin: 0 5px 5px 0;
}
...
```

После добавления отступа в данном коде ничего не раздражает. Или нет? Как думаете, что здесь может расстроить борца за оптимизацию?

**Ответ:** каждый раз, после любого изменения у нас вызывается `setState`, а значит - полная перерисовка компонента. Не очень приятно. Опять же, чуть больше логики в момент `render`'а компонента и в пору будет расстроиться от "отзывчивого" поля ввода.

У нас логики в `render`-методе никакой нет, поэтому для нас это лишняя оптимизация. Однако, рассмотреть способ создания "неконтролируемых компонентов" нужно обязательно.

Документация по `react`, предлагает вам использовать в большинстве случаев **контролируемые** компоненты.

---

## Uncontrolled Components (неконтролируемый компонент)

Главное отличие **неконтролируемого** компонента от **контролируемого** в том, что у него нет обработчика изменений, а значит нет постоянных вызовов `setState` и перерисовок.

Для того чтобы считать значение неконтролируемого компонента используется механизм `refs`.

Для неконтролируемого компонента в момент начальной загрузки можно указывать `defaultValue`.

Начнем по порядку:

1. Удалим обработчик `onChange`
2. Удалим `state`
3. Укажем `defaultValue` = пустая строка ( `defaultValue=''` ) вместо `value`
4. В `constructor` методе компонента (оп-па) укажем `this.input = React.createRef();`
5. Добавим атрибут `ref` инпуту, равный `this.input` (который в конструкторе создали)

```
class TestInput extends React.Component {
  constructor(props) {
    super(props)
    this.input = React.createRef()
  }
  onBtnClickHandler = (e) => { // эта запись сейчас не работает
    alert(this.state.myValue);
  }
  render() {
    return (
      <React.Fragment>
        <input
          className='test-input'
          defaultValue=''
          placeholder='введите значение'
          ref={this.input}
        />
        <button onClick={this.onBtnClickHandler}>Показать alert</button>
      </React.Fragment>
    )
  }
}
```

Обновите страницу, попробуйте ввести значение. Работает? Работает!

Теперь нам нужно, научиться считывать значение: перепишите *onBtnClickHandler* следующим образом:

```
onBtnClickHandler = () => {
  alert(this.input.current.value)
},
```

В [документации](#) так же есть пример использования refs и доступа к файлу, загружаемому через `<input type="file" />`.

За этот урок, мы научились с вами не вызывать дорогой *setState* и *render* на "каждый чих".

P.S. конечно, в данном случае никакого выигрыша в производительности нет. Оба подхода хорошо работают.

Вариант с контролируемыми и неконтролируемыми компонентами, работа с *defaultValue* и *state* являются одинаковыми для всех элементов форм.

Очень рекомендую посмотреть страницу документации на англ. языке по [элементам форм](#)

[Исходный код](#) на данный момент (включая *alert* и *console.log*). Пока что оставлен *input* с *ref*.

# Жизненный цикл компонента

Любимая фраза этого учебника "давайте представим задачу":

Мы отрисовали компонент, в котором есть `input`, и хотим чтобы фокус сразу же установился в него. Когда я первый раз подумал "как это сделать", даже не придумал что и ответить.

Хорошо, допустим я знаю, что могу достучаться до DOM элемента через `this.input.current` и вызвать нативный метод `focus()`, но в какой момент стучаться?

Какие вообще "моменты" есть?

---

## Lifecycle methods

У каждого компонента, есть жизненный цикл (*lifecycle*): компонент будет примонтирован, компонент отрисовался, компонент будет удален и так далее...

У всех этих фаз есть методы, так называемые *lifecycle-methods*. Полный [список](#) как всегда в документации. Предлагаю вам в конце урока еще раз его посмотреть, а пока хватит информации и здесь.

*Маленькая ремарка: здесь я перечислю часть самых популярных методов, причем будут и старые методы, так как еще многие их не вычистили. Старые методы будут отмечены биркой [DEPRECATED]*

Стартуем:

- `componentWillMount` - компонент будет примонтирован. В данный момент у нас нет возможности посмотреть DOM элементы. **[DEPRECATED]**
- `componentDidMount` - компонент примонтировался. В данный момент у нас есть возможность использовать `refs`, а следовательно это то самое место, где мы хотели бы указать установку фокуса. Так же, таймауты, аякс-запросы и взаимодействие с другими библиотеками стоит обрабатывать **здесь**.

Этот метод подходит для решения нашей задачи:

```
class TestInput extends React.Component {
  constructor(props) {
    super(props)
    this.input = React.createRef()
  }
  componentDidMount() {
    // ставим фокус в input
    this.input.current.focus()
  }
  onBtnClickHandler = () => {
    alert(this.input.current.value)
  }
  render() {
    return (
      <React.Fragment>
        <input
          className='test-input'
          defaultValue=''
          placeholder='введите значение'
          ref={this.input}
        />
        <button onClick={this.onBtnClickHandler}>Показать alert</button>
      </React.Fragment>
    )
  }
}
```

Принцип прежний: мы находим DOM-узел, считываем его свойство / вызываем его нативный метод, в данном случае - вызываем метод `focus()`. Обращаться к DOM-элементам напрямую - **очень редкая практика** в React.

- *componentWillReceiveProps* - компонент получает новые *props*. Этот метод не вызывается в момент первого render'a. В официальной документации очень хороший пример, пожалуй скопирую его: **[DEPRECATED]**

```
componentWillReceiveProps (nextProps) {
  this.setState({
    likesIncreasing: nextProps.likeCount > this.props.likeCount
  });
}
```

Обратите внимание: в этот момент, старые *props* доступны как *this.props*, а новые *props* доступны в виде **nextProps** аргумента функции.

Так же, если вы вызываете *setState* внутри этого метода - **не будет** вызван дополнительный *render*.



- *shouldComponentUpdate* - должен ли компонент обновиться? На самом деле, обычно реакт сам отлично разбирается. Но иногда ручное управление позволяет существенно ускорить работу в "узких местах". С этим методом нужно работать очень аккуратно.
- *componentWillUpdate* - вызывается прямо перед *render*, когда новые *props* и *state* получены. В этом методе нельзя вызывать *setState*. **[DEPRECATED]**
- *componentDidUpdate* - вызывается сразу после *render*. Не вызывается в момент первого *render*'а компонента.
- *componentWillUnmount* - вызывается перед тем, как компонент будет удален из DOM.

Конечно, в [документации](#) все описано намного подробнее. Я рекомендую с ней ознакомиться.

Здесь **хочу заострить внимание**, что чаще всего в старом коде или старых туториалах будет попадаться метод [componentWillReceiveProps](#), который Facebook-команда предлагает заменять на [getDerivedStateFromProps](#).

---

**Итого:** главная мысль данного урока: у компонента есть стадии жизни, "в которые можно писать код". Да, пусть я выступаю здесь как "плохой программист", который советует вам писать свои велосипеды на разных стадиях жизни компонента, но именно таким образом вы быстро освоитесь. Ставьте *console.log* и смотрите когда он срабатывает. Думайте как это можно использовать в своих целях.

Если же вы принадлежите к "правильному" типу программистов - пожалуйста, вот [все lifecycle-методы](#). Выучите, перечитайте, осознайте - и пишите код без багов ;)

**Итого:** существует несколько lifecycle-методов, благодаря которым мы уже почти перестали "лазить" в DOM, а если и делаем это, то осознанно.

[Исходный код](#) на данный момент.

## Работа с формой

В данном уроке мы превратим наш *input* в форму добавления новости. Научимся работать с чекбоксами, `disabled` атрибутом кнопки и прочими стандартными для такой задачи вещами.

Результатом добавления новости, пока что, вновь, будет *alert* с текстом новости.

---

Переименуйте `<TextInput />` в `<Add />`, и рендерите в нем следующую форму: автор (*input*), текст новости (*textarea*), "я согласен с правилами" (*checkbox*), "показать alert" (*button*).

Попутно изменим названия классов, удалим автофокус, удалим лишние обработчики и переместим компонент `<Add />` перед заголовком "Новости".

Итого, заготовка для "добавления новости" будет выглядеть следующим образом:

```
class Add extends React.Component {
  onBtnClickHandler = (e) => {
    e.preventDefault()
  }
  render() {
    return (
      <form className='add'>
        <input
          type='text'
          className='add__author'
          placeholder='Ваше имя'
        />
        <textarea
          className='add__text'
          placeholder='Текст новости'
        ></textarea>
        <label className='add__checkrule'>
          <input type='checkbox' /> Я согласен с правилами
        </label>
        <button
          className='add__btn'
          onClick={this.onBtnClickHandler}>
          Показать alert
        </button>
      </form>
    )
  }
}
```

Если вы не против моего оформления, можете взять стили для компонента `<Add />` :

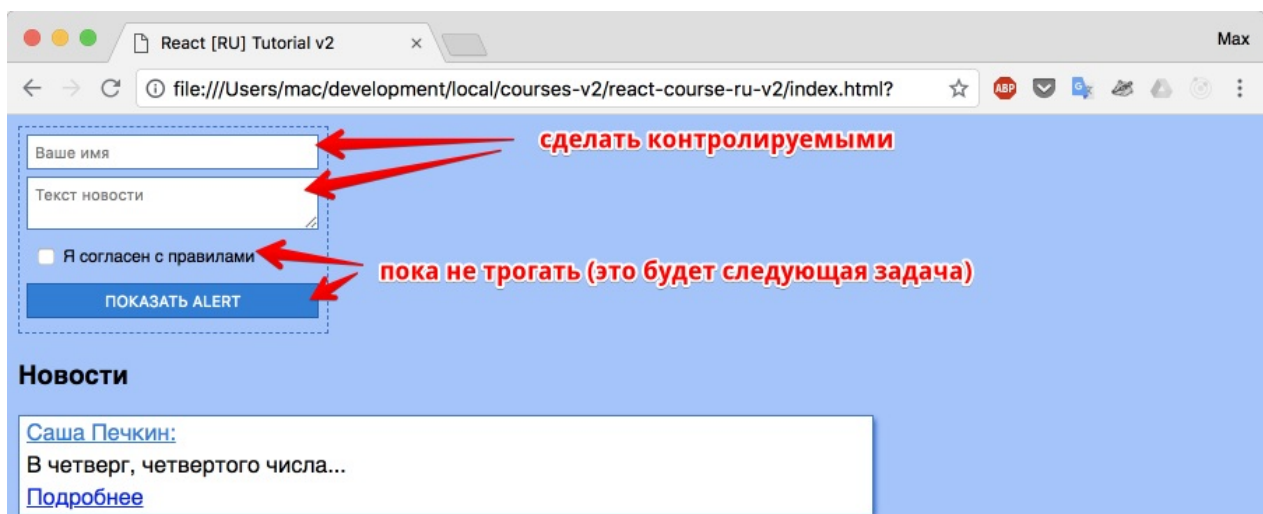
```

.add {
  margin: 0 5px 5px 0;
  width: 210px;
  border: 1px dashed rgba(0, 89, 181, 0.82);
  padding: 5px;
}
.add__author, .add__text, .add__btn, .add__checkrule {
  display: block;
  margin: 0 0 5px 0;
  padding: 5px;
  width: 94%;
  border: 1px solid rgba(0, 89, 181, 0.82);
}
.add__checkrule {
  border: none;
  font-size: 12px;
}
.add__btn {
  box-sizing: content-box;
  color: #FFF;
  text-transform: uppercase;
  background: #007DDC;
}
.add__btn:disabled {
  background: #CCC;
  color: #999;
}

```

Так как мы близки к финалу, я бы хотел нагрузить вас работой.

**Задача:** сейчас инпут "ваше имя" и text area - просто "болванка". Нужно сделать их контролируемыми.



**Подсказка:**

- создайте state (начальное состояние)
- добавьте обработчики на изменение имени и текста новости
- в value элементов записывайте значение переменной из состояния

**Решение:** (код сейчас не идеальный, но понятный. Рефакторить будем в конце раздела)

```
class Add extends React.Component {
  state = { // добавили начальное состояние
    name: '',
    text: '',
  }
  onBtnClickHandler = (e) => {
    e.preventDefault()
  }
  handleNameChange = (e) => { обработчик, в котором обновляем name
    this.setState({ name: e.currentTarget.value })
  }
  handleTextChange = (e) => { обработчик, в котором обновляем text
    this.setState({ text: e.currentTarget.value })
  }
  render() {
    const { name, text } = this.state // вытащили значения из стейта

    // добавили value для name и для textarea
    return (
      <form className='add'>
        <input
          type='text'
          onChange={this.handleNameChange}
          className='add__author'
          placeholder='Ваше имя'
          value={name}
        />
        <textarea
          onChange={this.handleTextChange}
          className='add__text'
          placeholder='Текст новости'
          value={text}
        ></textarea>
        <label className='add__checkrule'>
          <input type='checkbox' /> Я согласен с правилами
        </label>
        <button
          className='add__btn'
          onClick={this.onBtnClickHandler}>
          Показать alert
        </button>
      </form>
    )
  }
}
```

Инпуты начали работать и у нас есть очень приятный бонус: имя и текст новости хранится в `this.state`. Удобно? Разумеется. Представьте, что мы будем делать валидацию формы. У нас в любой момент времени будут **актуальные значения**

имени и текста новости! Все еще не в восторге? Немного терпения и мы доберемся до валидации...

Однако, для начала заставим работать связку чекбокс + кнопка отправки новости.

---

Давайте отключим кнопку "показать alert", если не отмечен чекбокс. Здесь есть 2 варианта - использовать *state* или не использовать. Для нашей задачи никаких проблем с производительностью не будет, если мы будем использовать *state*. Лазить в DOM (даже с помощью refs) в React-приложениях - не лучший вариант.

Разобьем задачу на этапы, необходимо:

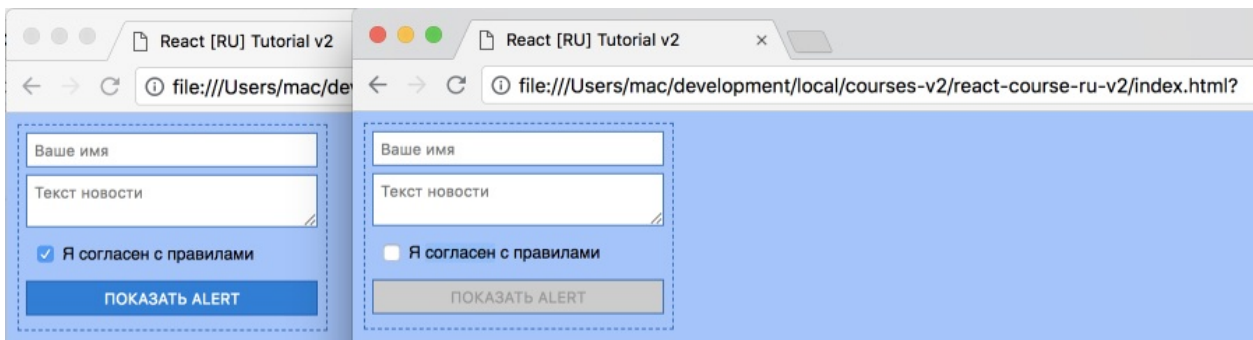
- добавить значение в *state* для чекбокса (*true/false*);
- добавить атрибут *disabled* у кнопки равным значению из *state*;
- добавить функцию обработчик;

Попробуйте сами, либо посмотрите решение:

```
class Add extends React.Component {
  state = {
    name: '',
    text: '',
    agree: false, // новое значение состояния - agree (булево)
  }
  onBtnClickHandler = (e) => {
    e.preventDefault()
  }
  handleNameChange = (e) => {
    this.setState({ name: e.currentTarget.value })
  }
  handleTextChange = (e) => {
    this.setState({ text: e.currentTarget.value })
  }
  handleCheckboxChange = (e) => { // обработчик кликов по чекбоксу
    // чтобы установить true/false считываем свойство checked
    this.setState({ agree: e.currentTarget.checked })
  }
  render() {
    const { name, text, agree } = this.state
    return (
      <form className='add'>
        <input
          type='text'
          onChange={this.handleNameChange}
          className='add__author'
          placeholder='Ваше имя'
          value={name}
        />
        <textarea
          onChange={this.handleTextChange}
          className='add__text'
          placeholder='Текст новости'
          value={text}
        ></textarea>
        <label className='add__checkrule'>
          <input type='checkbox' onChange={this.handleCheckboxChange} /> Я согласен с
правилами
        </label>
        { /* кнопке добавили disabled равный (НЕ agree) */ }
        <button
          className='add__btn'
          onClick={this.onBtnClickHandler}
          disabled={!agree}>
          Показать alert
        </button>
      </form>
    )
  }
}
```



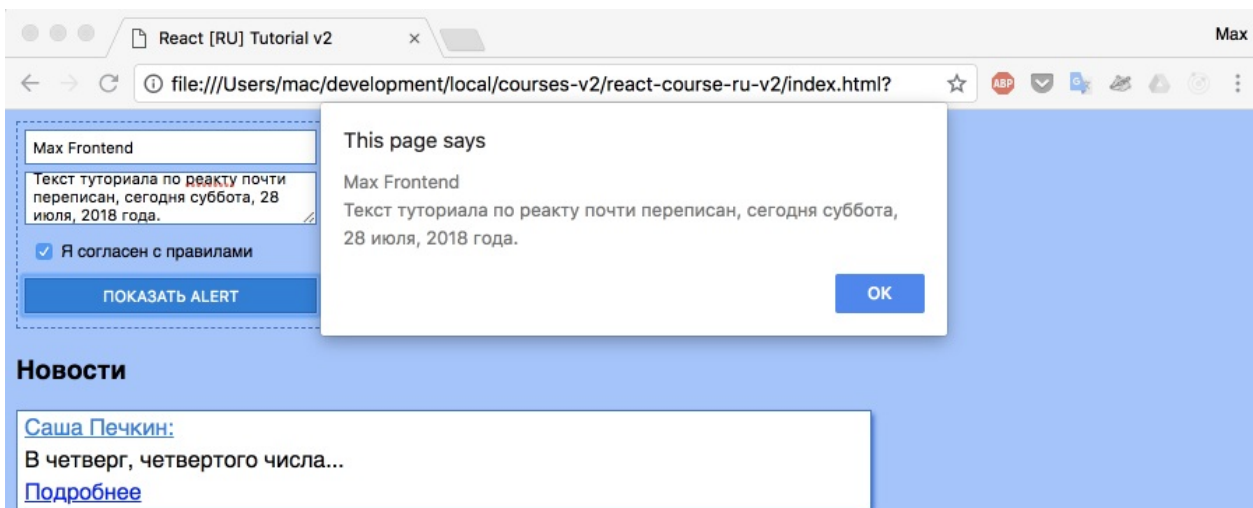
Так как по клику в чекбокс происходит изменение state, то вызывается перерисовка (метод `render` отработает). Значит, у нас всегда будет актуальное значение `agree` в `disabled` атрибуте.



Мне нечего более здесь комментировать для тех, кто знает основы JavaScript. Для тех, кто не знает:

- `checked` (Кантор);
- Изменение: `change`, `input`, `cut`, `copy`, `paste` (Кантор)
- `disabled = true` будет означать, что кнопка выключена. Кнопка должна быть выключена тогда, когда `agree = false` (то есть, чекбокс не отмечен), значит мы делаем отрицание (НЕ) с помощью знака восклицания;

Для добавления новости нам осталось сформировать код, который будет выводить в *alert* имя и текст новости. Я думаю, эта задача вам точно под силу.



**Решение:**

```
...
onBtnClickHandler = (e) => {
  e.preventDefault()
  const { name, text } = this.state
  alert(name + '\n' + text) // \n = перенос строки
}
```

## Блокировка кнопки, если не все поля заполнены

Приехала обещанная валидация. Точнее, *"Валидация часть 1: Начало"*.

Почему часть 1? Потому что, валидация форм и вообще работа с формой - одна из самых скрупулезных задач. Нужно показывать понятные сообщения об ошибках, подсвечивать неправильно заполненное поле, блокировать кнопку отправки, валидировать поля по определенным правилам и так далее.

В данный момент, мы добавим к условию (что чекбокс отмечен) только одно простое условие: поля `name` и `text` должны быть заполнены. И не пробелами.

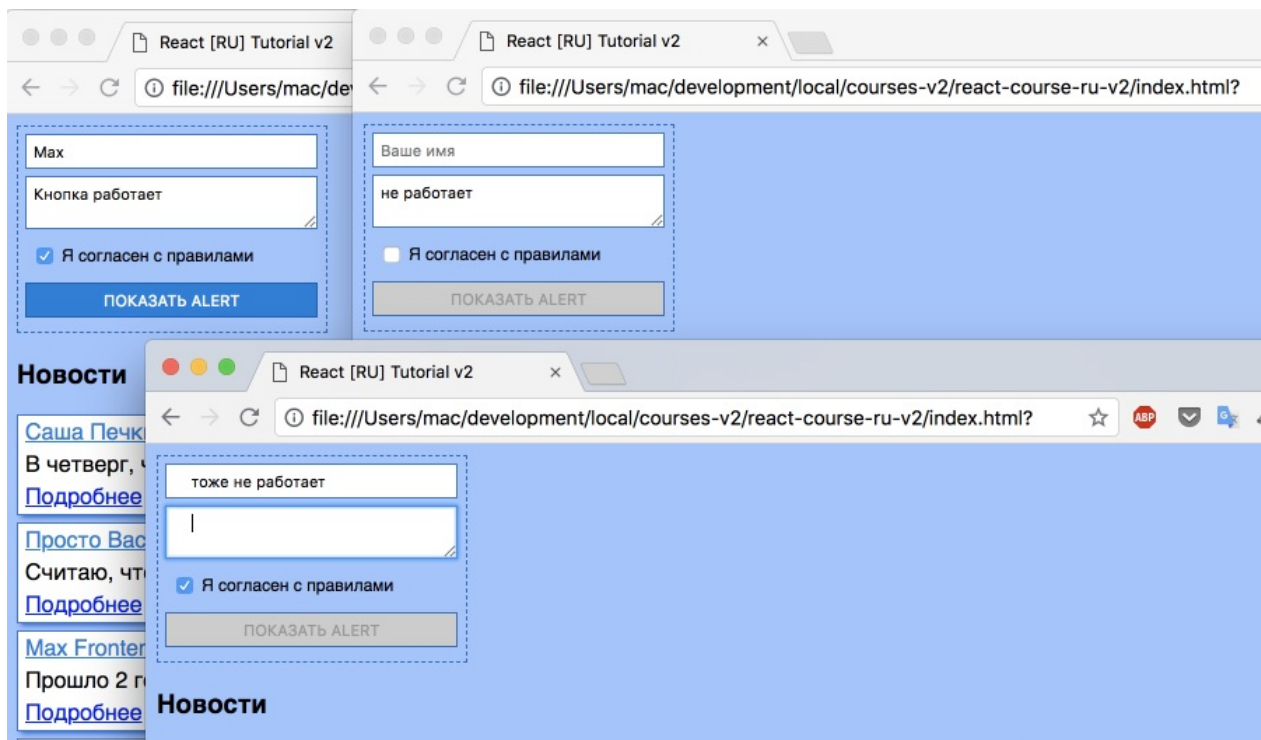
Как бы решалась такая задача без `react`? Вероятно у нас была бы функция *validate*, которая вызывалась бы на каждое изменение в проверяемых полях. Нужно было бы генерировать и прослушивать событие...

Думаю, вы поняли намек. Здесь без *state* не обойтись, и это точно то место, где удобнее использовать именно **состояние**, а не `refs`.

Попробуйте сами, а потом сверьтесь с решением.

---

**Задача:** если в поле "имя" или "текст" не введено ничего (либо пробелы) - кнопка "показать alert" должна быть недоступной.



Подсказка #1: для удаления пробелов используйте стандартный метод `trim()`

Подсказка #2: вам потребуется в атрибут `disabled` передавать результат работы функции `validate` (которую нужно создать в качестве метода компонента, чтобы был доступ к `this.state`). Пример:

```
validate = () => {
  // какие то условия
  // возвращает true или false
}
...
<button disabled={this.validate()} ... >
...
```

**Решение:**

```
class Add extends React.Component {
  state = {
    name: '',
    text: '',
    agree: false,
  }
  ...
  validate = () => {
    const { name, text, agree } = this.state
    if (name.trim() && text.trim() && agree) {
      return true
    }
    return false
  }
  render() {
    const { name, text, agree } = this.state
    return (
      <form className='add'>
        ...
        <button
          className='add__btn'
          onClick={this.onBtnClickHandler}
          disabled={!this.validate()}>
            Показать alert
          </button>
        </form>
      )
    )
  }
}
```

Ну как? Не очень весело? Если да - значит у вас проблемы с основами JavaScript и вам нужно их подтягивать. А если в целом порядочек - я вас поздравляю, мы почти у цели.

---

## Порефакторим копипасту

Есть проблемка:

```
handleNameChange = (e) => {
  this.setState({ name: e.currentTarget.value })
}
handleTextChange = (e) => {
  this.setState({ text: e.currentTarget.value })
}
```

Очень похожие методы. Можно ли их унифицировать? Конечно.

- необходимо добавить вычисляемое значение ключа;
- понять откуда будем считывать ключ;

Ключ будем считывать из id элемента. Идея такова: в id записываем такую же строку, что и значение ключа в state, то есть для name - инпуту дадим `id='name'`, а для textarea - `id='text'`

Готовы?

```
class Add extends React.Component {
  ...
  handleChange = (e) => {
    const { id, value } = e.currentTarget
    this.setState({ [id]: e.currentTarget.value })
  }
  ...
  render() {
    const { name, text, agree } = this.state
    return (
      <form className='add'>
        <input
          id='name'
          type='text'
          onChange={this.handleChange}
          className='add__author'
          placeholder='Ваше имя'
          value={name}
        />
        <textarea
          id='text'
          onChange={this.handleChange}
          className='add__text'
          placeholder='Текст новости'
          value={text}
        ></textarea>
        ...
      </form>
    )
  }
}
```

Из `e.currentTarget` мы можем считать `id` и `value`. Далее записываем в стейт по нужному ключу - значение.

Вычисляемое значение ключа - это одна из самых больных тем новичков.

Внимательно читаем (и перечитываем) урок из учебника Кантора: [Объекты как ассоциативные массивы](#)

**Итого:** мы научились работать с формой. Положили начало досерверной валидации. Поняли и осознали, что знание основ JavaScript - это невероятный "буст" в изучении React. Да, без основ можно читать tutorиалы и выполнять задачи, но поверьте, если вы сначала подтянете основы, то React станет вам другом гораздо быстрее.

[Исходный код](#) на данный момент.

# Добавить новость

Что такое добавление новости?

1. Это форма, в которую мы вводим необходимые данные.
2. Это "лента новостей", которая отображает наши данные.

У данной задачи есть масса вариантов решения. Мы начнем с канонического варианта: в общем родителе ( `<App/>` ) будем хранить state с новостями. В компонент `<Add />` будем передавать функцию (так как в props мы можем передавать что угодно), которая будет иметь доступ к state с новостями и которая в свою очередь будет добавлять новость в этот state.

Так как state у `<App />` будет изменяться, все дети (в том числе и новостная лента `<News />` ) будут перерисованы, а следовательно - мы увидим добавленную новость.

## Взаимодействие из ребенка с родителем

**Шаг 1:** создадим состояние с новостями в `<App />` (а следовательно, переделаем App из stateless в statefull):

```
class App extends React.Component {
  state = {
    news: myNews, // в начальное состояние положили значение из переменной
  }

  render() {
    return (
      <React.Fragment>
        <Add />
        <h3>Новости</h3>
        { /* считали новости из this.state */ }
        <News data={this.state.news} />
      </React.Fragment>
    )
  }
}
```

Что примечательно, мы изменили источник данных для `<News />` , но компонент работает как ни в чем не бывало. Удобно!

**Шаг 2:** передадим функцию-обработчик в Add

```
class App extends React.Component {
  state = {
    news: myNews,
  }
  handleAddNews = () => {
    console.log('я вызвана из Add, но имею доступ к this.state у App!', this.state)
  }
  render() {
    return (
      <React.Fragment>
        <Add onAddNews={this.handleAddNews} />
        <h3>Новости</h3>
        <News data={this.state.news} />
      </React.Fragment>
    )
  }
}
```

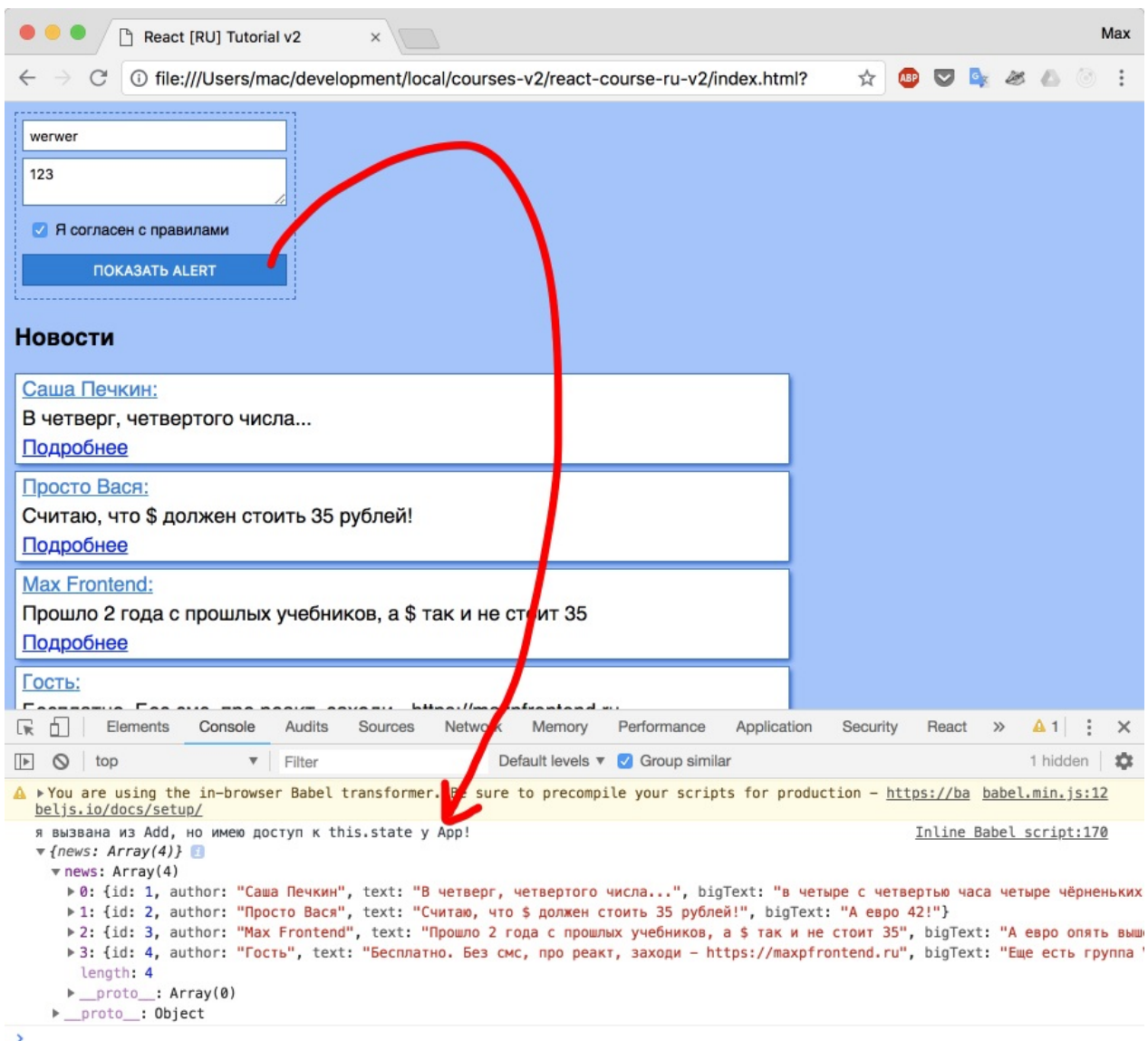
**Шаг 3:** вызовем функцию из `<Add />`, не забудем про *PropTypes*.



```
class Add extends React.Component {
  state = {
    name: '',
    text: '',
    agree: false,
  }
  onBtnClickHandler = (e) => {
    e.preventDefault()
    const { name, text } = this.state
    // alert(name + '\n' + text)
    // вызываем вместо alert
    this.props.onAddNews()
  }
  ...
  render() {
    const { name, text, agree } = this.state
    return (
      <form className='add'>
        ...
        <button
          className='add__btn'
          onClick={this.onBtnClickHandler}
          disabled={!this.validate()}>
            Показать alert
          </button>
        </form>
      )
    )
  }
}

Add.propTypes = {
  onAddNews: PropTypes.func.isRequired, // func используется для проверки передачи fun
  ction
}
```

Проверим:



**Шаг 4:** из `<Add />` будем передавать объект с новостью.

```
class Add extends React.Component {
  ...
  onBtnClickHandler = (e) => {
    e.preventDefault()
    const { name, text } = this.state
    // передаем name и text
    // big text у нас отсутствует :(
    this.props.onAddNews({ name, text })
  }
  ...
}
```

**Шаг 5:** полученный объект будем записывать на первое место в массиве с новостями в `<App />`. Разумеется, массив будем обновлять через `setState`.

```

class App extends React.Component {
  ...
  handleAddNews = (data) => {
    // сначала мы формируем массив, на основе
    // всего того, что уже было в новостях
    // и кладем это все в новый массив +
    // новую новость кладем в начало массива
    const nextNews = [data, ...this.state.news]

    // затем обновляем новый массив новостей в this.state.news
    this.setState({ news: nextNews })
  }
  ...
}

```

Проверим?

Ученик

Новая новость

☒ Я согласен с правилами

ПОКАЗАТЬ ALERT

### Новости

Новая новость  
[Подробнее](#)

Саша Печкин:  
В четверг, четвертого числа...  
[Подробнее](#)

Просто Вася:  
Считаю, что \$ должен стоить 35 рублей!  
[Подробнее](#)

Max Frontend:  
Просто Вася считает, что \$ должен стоить 35...

Elements Console Audits Sources Network Memory Performance Application Security >> 2 1 1

top Filter Default levels Group similar 1 hidden

Warning: You are using the in-browser Babel transformer. Be sure to precompile your scripts for production - <https://babel.min.js:12>

Warning: Failed prop type: The prop `data.author` is marked as required in `Article`, but its value is `undefined`.  
in Article (created by News)  
in News (created by App)  
in App **внимательно читаем текст ошибки**

Warning: Each child in an array or iterator should have a unique "key" prop.  
Check the render method of `News`. See <https://fb.me/react-warning-keys> for more information.  
in Article (created by News)  
in News (created by App)  
in App

Окей, перед нами отличный кейс (реальный рабочий случай).

*Во-первых:* мы были почти прилежными учениками и сделали propTypes, для `<Article />`. Из ошибки сразу понятно: мы не передаем значение *author* (потому что, мы передаем *name*).

*Во-вторых:* мы все же накосячили, и забыли добавить в propTypes для `<Article />` свойство `id`. Хорошо, что кода мало и ошибка сразу нашлась. Не забывайте про перечисление всех свойств в `propTypes` - это супер шпаргалка.

*В-третьих:* мы из `<Add />` не передаем `id` и `bigText`.

Тем не менее, нельзя не отметить, что мы добавили динамики в наше приложение! Новость добавляется, причем счетчик новостей работает (а мы его вообще не трогали). Кто уже празднует - молодец, но кто хочет пофиксить все ошибки и сдать работу на пять - милости прошу в заключительный пункт основного курса.

---

## Работа над ошибками

Я бы хотел, чтобы все это вы пофиксили сами. Но, чтобы не быть автором, который "оп-па, косяк, давайте-ка я отдам это вам на домашку", я все сделаю и опишу. Просто попробуйте. У вас получится. Практика решает. Обязательно всегда практикуйтесь для закрепления материала.

### Задача:

- добавить в форму добавления textarea для `bigText`;
- передавать `author`;
- передавать `id` (можно сделать через `timestamp` (отметку времени в ms): `+new Date()`). Для обучающего примера будет достаточно;
- исправить propTypes в `<News />`

The screenshot shows a web application with a form to add a news item and a list of news items. The form has fields for author, main text, and a checkbox for agreement. The news list shows two items: one by 'Максим Пацианский' and one by 'Саша Печкин'. Red arrows indicate the mapping from the form fields to the rendered news items. The bottom part of the image shows the React DevTools component inspector with the 'Application' tab selected, displaying the component tree and props for the selected article.

## Решение:

Полный код того, что находится в `<body />` (с комментариями о последних изменениях)

```
<body>
  <div id="root"></div>
  <script type="text/babel">

    const myNews = [
      {
        id: 1,
        author: 'Саша Печкин',
        text: 'В четверг, четвертого числа...',
        bigText: 'в четыре с четвертью часа четыре чёрненьких чумазеньких чертёнка чер
тили чёрными чернилами чертёж.'
      },
      {
        id: 2,
        author: 'Просто Вася',
        text: 'Считаю, что $ должен стоить 35 рублей!',
        bigText: 'А евро 42!'
      }
    ]
```

```

    },
    {
      id: 3,
      author: 'Max Frontend',
      text: 'Прошло 2 года с прошлых учебников, а $ так и не стоит 35',
      bigText: 'А евро опять выше 70.'
    },
    {
      id: 4,
      author: 'Гость',
      text: 'Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru',
      bigText: 'Еще есть группа VK, telegram и канал на youtube! Вся инфа на сайте,
не реклама!'
    }
  ];

class Article extends React.Component {
  state = {
    visible: false,
  }
  handleReadMoreClick = (e) => {
    e.preventDefault()
    this.setState({ visible: true })
  }
  render() {
    const { author, text, bigText } = this.props.data
    const { visible } = this.state
    return (
      <div className='article'>
        <p className='news__author'>{author}</p>
        <p className='news__text'>{text}</p>
        {
          !visible && <a onClick={this.handleReadMoreClick} href="#" className='news__readmore'>Подробнее</a>
        }
        {
          visible && <p className='news__big-text'>{bigText}</p>
        }
      </div>
    )
  }
}

Article.propTypes = {
  data: PropTypes.shape({
    id: PropTypes.number.isRequired, // добавили id, это число, обязательно
    author: PropTypes.string.isRequired,
    text: PropTypes.string.isRequired,
    bigText: PropTypes.string.isRequired
  })
}

class News extends React.Component {

```

```

// удалили старое состояние counter: 0 (старый ненужный код)
renderNews = () => {
  const { data } = this.props
  let newsTemplate = null

  if (data.length) {
    newsTemplate = data.map(function(item) {
      return <Article key={item.id} data={item}/>
    })
  } else {
    newsTemplate = <p>К сожалению новостей нет</p>
  }

  return newsTemplate
}

render() {
  const { data } = this.props

  return (
    <div className='news'>
      {this.renderNews()}
      {
        data.length ? <strong className={'news__count'}>Всего новостей: {data.length}</strong> : null
      }
    </div>
  );
}

News.propTypes = {
  data: PropTypes.array.isRequired
}

class Add extends React.Component {
  state = {
    name: '',
    text: '',
    bigText: '', // добавлен bigText
    agree: false,
  }
  onBtnClickHandler = (e) => {
    e.preventDefault()
    const { name, text, bigText } = this.state // вытащили так же и bigText
    this.props.onAddNews({
      id: +new Date(), // в id сохраняется количество миллисекунд прошедших с 1 января 1970 года в часовом поясе UTC
      author: name, // name сохраняем в поле author
      text,
      bigText,
    })
  }
  handleChange = (e) => {

```

```

    const { id, value } = e.currentTarget
    this.setState({ [id]: e.currentTarget.value })
  }
  handleCheckboxChange = (e) => {
    this.setState({ agree: e.currentTarget.checked })
  }
  validate = () => {
    const { name, text, agree } = this.state
    if (name.trim() && text.trim() && agree) {
      return true
    }
    return false
  }
  render() {
    const { name, text, bigText, agree } = this.state
    return (
      <form className='add'>
        <input
          id='name'
          type='text'
          onChange={this.handleChange}
          className='add__author'
          placeholder='Ваше имя'
          value={name}
        />
        <textarea
          id='text'
          onChange={this.handleChange}
          className='add__text'
          placeholder='Текст новости'
          value={text}
        ></textarea>
        { /* добавили bigText */ }
        <textarea
          id='bigText'
          onChange={this.handleChange}
          className='add__text'
          placeholder='Текст новости подробно'
          value={bigText}
        ></textarea>
        <label className='add__checkrule'>
          <input type='checkbox' onChange={this.handleCheckboxChange} /> Я согласен с правилами
        </label>
        <button
          className='add__btn'
          onClick={this.onBtnClickHandler}
          disabled={!this.validate()}>
          Показать alert
        </button>
      </form>
    )
  }
}

```



```
}

Add.propTypes = {
  onAddNews: PropTypes.func.isRequired,
}

class App extends React.Component {
  state = {
    news: myNews,
  }
  handleAddNews = (data) => {
    const nextNews = [data, ...this.state.news]
    this.setState({ news: nextNews })
  }
  render() {
    return (
      <React.Fragment>
        <Add onAddNews={this.handleAddNews}/>
        <h3>Новости</h3>
        <News data={this.state.news}/>
      </React.Fragment>
    )
  }
}

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

</script>

</body>
```

Готово!

[Исходный код](#) на данный момент.

# Итоги

Чему вы научились на данный момент:

- Создавать компоненты (с помощью `function` и с помощью `class`);
  - Однако в чем разница, кроме сокращенной и подробной записи? [1]
- Передавать свойства (`props`) и считывать их;
- Общаться из ребенка с родителем (через функцию, переданную в `props`);
- Делать развилки в шаблонах (`if/else`, `&&` и т.д.)
- Работать с изменяемым состоянием компонента (`state`)
- Работать с формой (контролируемые и не контролируемые компоненты)
- Познакомились с методами жизненного цикла
  - На самом деле только с `componentDidMount` и `render` [2]

[1] - разница в том, что `stateless` компонент, имеет встроенную "легкую" проверку в `shouldComponentUpdate`. Она невидимая, но она есть. Этот пункт можно отнести к теме оптимизации перерисовок ваших компонентов.

[2] - этот пункт раскрыт во второй серии (про `Redux`), но так как `Redux`-тutorial еще не переписан, я думаю стоит "прокачать" пример с новостями здесь и сейчас.

Так же, есть неудобство, что вы слышали про [create-react-app](#) импорты и прочее, а здесь в руководстве всего этого нет. Конечно, это сделано для того, чтобы руководство было максимально "сухим" и по теме. Тем не менее, мне бы хотелось добавить это в обновленной версии.

Поэтому, я не прощаюсь с вами и приглашаю вкусить основ в дополнительных главах, в которых я буду предельно краток, чтобы не раздувать объем.

---

Напоминаю, что масса бесплатных и не очень материалов выходит в моих "соц.пространствах":

- [Расписание стримов и вебинаров](#) (на сайте есть текстовые версии вебинаров)
- [Youtube канал](#) с записями вебинаров и стримов
- Группа [vkontakte](#)
- Канал в [telegram](#)
- [Twitter](#)
- [Facebook](#)



# Create-react-app

Здесь я буду предельно краток: facebook выкатили удобный [инструмент для старта приложения](#). Поддерживается новый синтаксис, импорты, тестирование, перезагрузка страницы при изменениях, линтер и многое другое.

Во всем разнообразии мы сейчас разбираться не будем. Цель: разбить index.html на компоненты, подключить их, навести порядок.

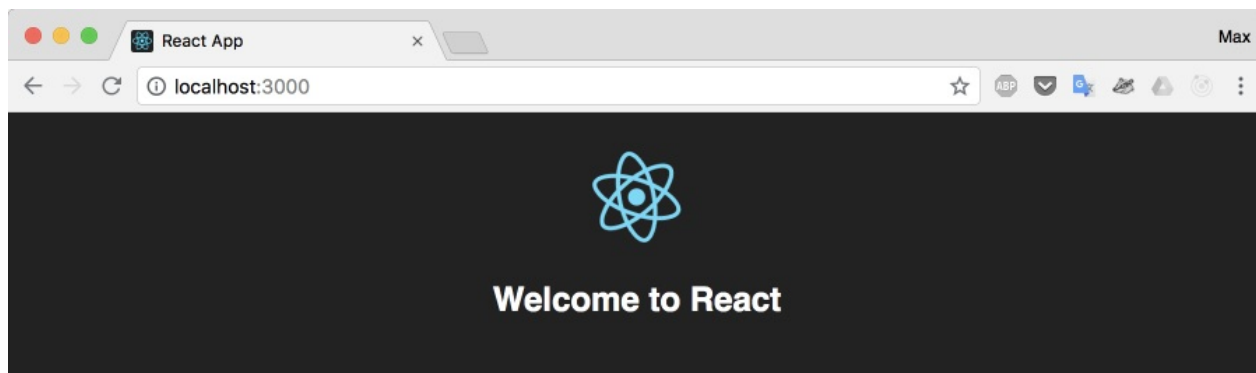
Скопируйте index.html куда-нибудь на память, скоро мы разобьем его на мелкие удобные компоненты.

Установка и запуск create-react-app

```
npx create-react-app my-app  
cd my-app  
npm start
```

Если вы не знакомы с данными командами, значит вам нужно поставить себе [node.js](#) и ввести их в терминале после.

После запуска мы получим следующую картину в браузере:



To get started, edit `src/App.js` and save to reload.



И следующую файловую структуру:

```
+-- node_modules (здесь расположены пакеты для работы приложения)
+-- public (здесь расположены публичные файлы, такие как index.html и favicon)
+-- src (здесь сейчас уже живет компонент App)
+-- .gitignore (файл для гита)
+-- package.json (файл с зависимостями проекта)
+-- README.md (описание проекта)
+-- yarn.lock (может быть, а может и не быть - тоже относится к теме зависимостей прое
кта)
```

Восстановим баланс в src

src/App.css (копируем все наши стили)

```
.none {
  display: none;
}

body {
  background: rgba(0, 102, 255, 0.38);
  font-family: sans-serif;
}

p {
  margin: 0 0 5px;
}

.article {
  background: #FFF;
  border: 1px solid rgba(0, 89, 181, 0.82);
  width: 600px;
  margin: 0 0 5px;
  box-shadow: 2px 2px 5px -1px rgb(0, 81, 202);
  padding: 3px 5px;
}

.news__author {
  text-decoration: underline;
  color: #007DDC;
}

.news__count {
  margin: 10px 0 0 0;
  display: block;
}

.test-input {
  margin: 0 5px 5px 0;
}

.add {
  margin: 0 5px 5px 0;
```

```

width: 210px;
border: 1px dashed rgba(0, 89, 181, 0.82);
padding: 5px;
}
.add__author, .add__text, .add__btn, .add__checkrule {
display: block;
margin: 0 0 5px 0;
padding: 5px;
width: 94%;
border: 1px solid rgba(0, 89, 181, 0.82);
}
.add__checkrule {
border: none;
font-size: 12px;
}
.add__btn {
box-sizing: content-box;
color: #FFF;
text-transform: uppercase;
background: #007DDC;
}
.add__btn:disabled {
background: #CCC;
color: #999;
}

```

**src/App.js** (копируем почти все из тэга script)

```

import React from 'react'; // подключение библиотеки React
import './App.css'; // подключение файла стилей

// далее скопировано из тэга script

const myNews = [
  {
    id: 1,
    author: "Саша Печкин",
    text: "В четверг, четвертого числа...",
    bigText:
      "В четыре с четвертью часа четыре чёрненьких чумазеньких чертёнка чертили чёрным  
и чернилами чертёж."
  },
  {
    id: 2,
    author: "Просто Вася",
    text: "Считаю, что $ должен стоить 35 рублей!",
    bigText: "А евро 42!"
  },
  {
    id: 3,
    author: "Max Frontend",

```

```

    text: "Прошло 2 года с прошлых учебников, а $ так и не стоит 35",
    bigText: "А евро опять выше 70."
  },
  {
    id: 4,
    author: "Гость",
    text: "Бесплатно. Без смс, про реакт, заходи - https://maxpfrontend.ru",
    bigText:
      "Еще есть группа VK, telegram и канал на youtube! Вся инфа на сайте, не реклама!"
  }
];

class Article extends React.Component {
  state = {
    visible: false
  };
  handleReadMoreClick = e => {
    e.preventDefault();
    this.setState({ visible: true });
  };
  render() {
    const { author, text, bigText } = this.props.data;
    const { visible } = this.state;
    return (
      <div className="article">
        <p className="news__author">{author}</p>
        <p className="news__text">{text}</p>
        {!visible && (
          <a
            onClick={this.handleReadMoreClick}
            href="#"
            className="news__readmore"
          >
            Подробнее
          </a>
        )}
        {visible && <p className="news__big-text">{bigText}</p>}
      </div>
    );
  }
}

Article.propTypes = {
  data: PropTypes.shape({
    id: PropTypes.number.isRequired, // добавили id, это число, обязательно
    author: PropTypes.string.isRequired,
    text: PropTypes.string.isRequired,
    bigText: PropTypes.string.isRequired
  })
};

class News extends React.Component {

```

```
renderNews = () => {
  const { data } = this.props;
  let newsTemplate = null;

  if (data.length) {
    newsTemplate = data.map(function(item) {
      return <Article key={item.id} data={item} />;
    });
  } else {
    newsTemplate = <p>К сожалению новостей нет</p>;
  }

  return newsTemplate;
};

render() {
  const { data } = this.props;

  return (
    <div className="news">
      {this.renderNews()}
      {data.length ? (
        <strong className={"news__count"}>
          Всего новостей: {data.length}
        </strong>
      ) : null}
    </div>
  );
}

News.propTypes = {
  data: PropTypes.array.isRequired
};

class Add extends React.Component {
  state = {
    name: "",
    text: "",
    bigText: "",
    agree: false
  };

  onBtnClickHandler = e => {
    e.preventDefault();
    const { name, text, bigText } = this.state;
    this.props.onAddNews({
      id: +new Date(),
      author: name,
      text,
      bigText
    });
  };

  handleChange = e => {
    const { id, value } = e.currentTarget;
```



```

    this.setState({ [id]: e.currentTarget.value });
  };
  handleCheckboxChange = e => {
    this.setState({ agree: e.currentTarget.checked });
  };
  validate = () => {
    const { name, text, agree } = this.state;
    if (name.trim() && text.trim() && agree) {
      return true;
    }
    return false;
  };
  render() {
    const { name, text, bigText, agree } = this.state;
    return (
      <form className="add">
        <input
          id="name"
          type="text"
          onChange={this.handleChange}
          className="add__author"
          placeholder="Ваше имя"
          value={name}
        />
        <textarea
          id="text"
          onChange={this.handleChange}
          className="add__text"
          placeholder="Текст новости"
          value={text}
        />
        <textarea
          id="bigText"
          onChange={this.handleChange}
          className="add__text"
          placeholder="Текст новости подробно"
          value={bigText}
        />
        <label className="add__checkrule">
          <input type="checkbox" onChange={this.handleCheckboxChange} /> Я
            согласен с правилами
        </label>
        <button
          className="add__btn"
          onClick={this.onBtnClickHandler}
          disabled={!this.validate()}
        >
          Показать alert
        </button>
      </form>
    );
  }
}

```

```
Add.propTypes = {
  onAddNews: PropTypes.func.isRequired
};

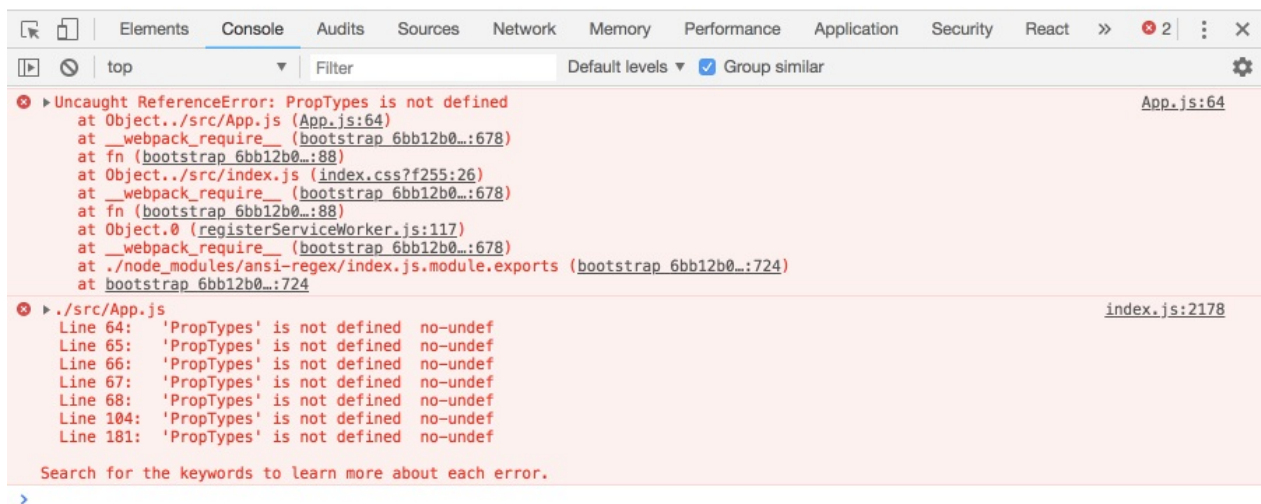
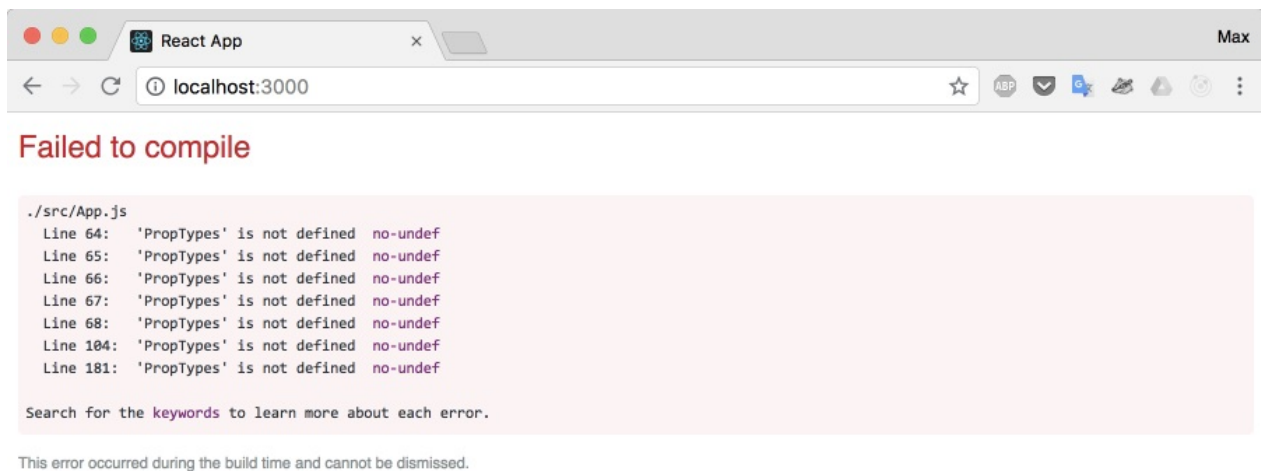
class App extends React.Component {
  state = {
    news: myNews
  };
  handleAddNews = data => {
    const nextNews = [data, ...this.state.news];
    this.setState({ news: nextNews });
  };
  render() {
    return (
      <React.Fragment>
        <Add onAddNews={this.handleAddNews} />
        <h3>Новости</h3>
        <News data={this.state.news} />
      </React.Fragment>
    );
  }
}

// скопировано все кроме ReactDOM.render

// добавился export
export default App;
```

Удалите файл `src/Logo.svg` , остальное не трогайте, но можете посмотреть :)

Create-react-app (CRA) при каждом изменении в файлах внутри директории `src` - перезагружает страницу в браузере.



У нас нет `PropTypes` в проекте. Так как раньше это был тэг `script`, а теперь нам нужен `npm`-пакет.

Остановите работу `create-react-app` в терминале и добавьте пакет [prop-types](#)

```
npm install --save prop-types
```

Я не привожу сюда примеров для `yarn`, так как если у вас `yarn` вы итак в курсе, как ставить пакеты через него.

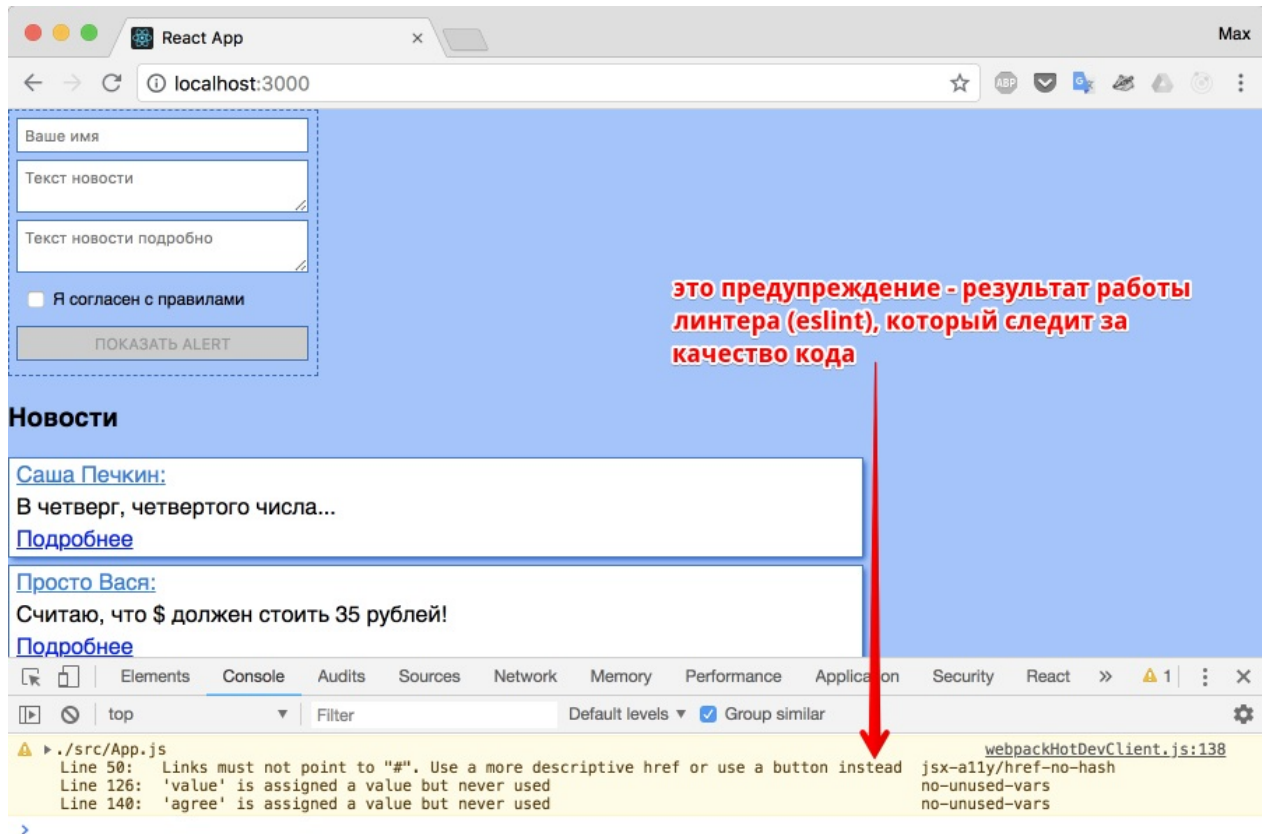
Подключите `PropTypes` в начале файла:

`src/App.js`

```
import React from 'react'
import PropTypes from 'prop-types'
import './App.css'
...
```

Запустите приложение еще раз:

```
npm start
```



**Итого:** мы перевели приложение на CRA.

**Исходный код** на данный момент (добавлен конфиг для преттири - .prettierrc, не обращайтесь внимание).

# Приборка

Запустите приложение, если оно у вас не запущено ( `npm start` ).

Как вы помните на прошлом скрине, у нас были предупреждения от [eslint](#). eslint - вспомогательный инструмент, который помогает поддерживать код в чистоте, на данный момент у нас следующие проблемы:

```
./src/App.js
  Line 50:  Links must not point to "#". Use a more descriptive href or use a button
instead  jsx-a11y/href-no-hash
  Line 126:  'value' is assigned a value but never used
              no-unused-vars
  Line 140:  'agree' is assigned a value but never used
              no-unused-vars
```

Строка 50: у ссылки должен быть атрибут href не #, а что-то более вразумительное (замените на '#readmore').

Строка 126 - value - не используется, исправьте:

```
// было
const { id, value } = e.currentTarget
this.setState({ [id]: e.currentTarget.value })

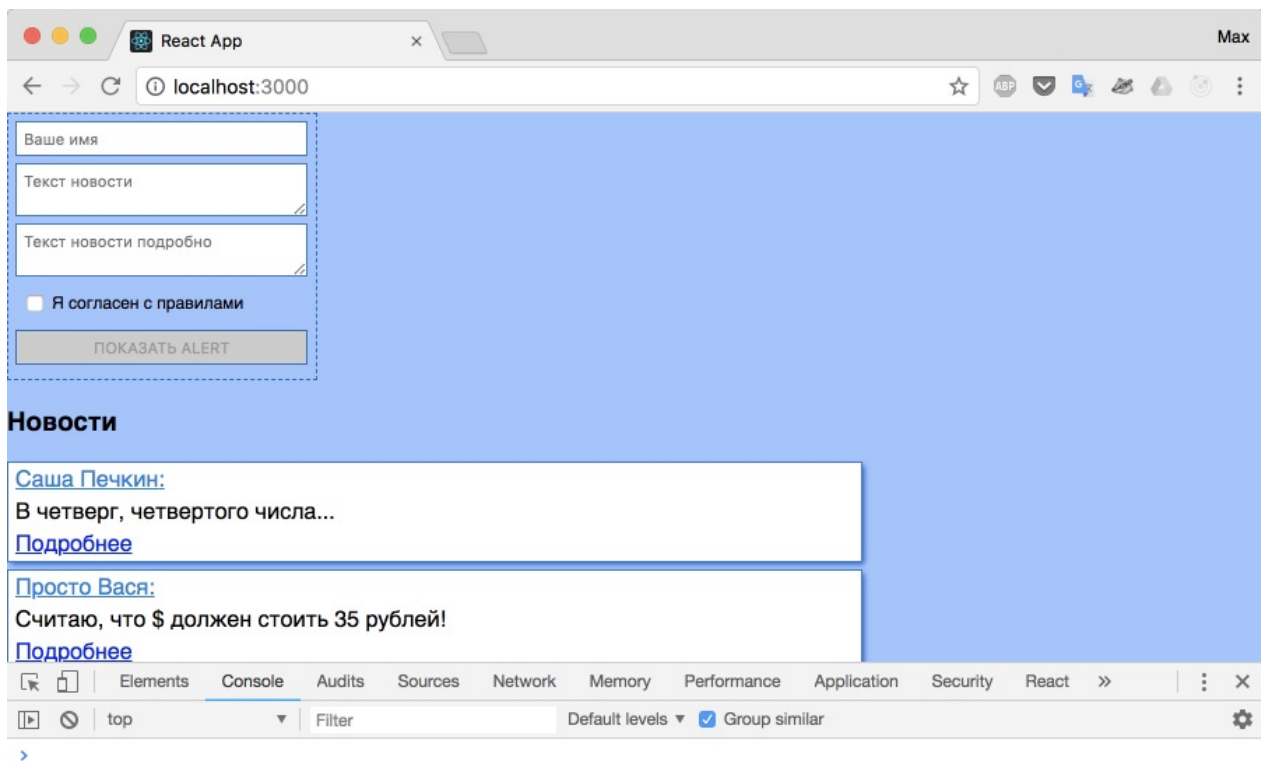
// стало
const { id, value } = e.currentTarget
this.setState({ [id]: value })
```

Строка 140 - agree - не используется, исправьте:

```
// было
const { name, text, bigText, agree } = this.state

// стало
const { name, text, bigText } = this.state
```

Поддерживайте свой код без предупреждений. Если вы не знаете о чем речь, например, сработало правило [no-unused-vars](#), не ленитесь. Изучайте документацию на [eslint.org](#)



**чистая консоль - залог здорового приложения**

## Импорты

Наша задача - разбить огромный файл `src/App.js` на компоненты.

Поддерживаются следующие импорты:

```
import A from 'A' // импорт по дефолту
import { B } from 'B' // именованный импорт
import * as C from 'C' // импорт "всего" в namespace C
```

С другой стороны поддерживаются следующие экспорты:

```
export default A // экспорт по дефолту
export const B // именованный экспорт
```

Мы создадим несколько файлов, в каждом из которых будем пользоваться именованным экспортом. Затем в App импортируем.

Создайте директорию `src/components` и в ней создайте файлы для каждого компонента, кроме `<App />` .

Рассмотрим создание файла, на примере `<Article />` :

*src/components/Article.js*

```
import React from 'react' // мы обязаны импортировать необходимые пакеты в каждом файле

import PropTypes from 'prop-types' // у Article это react и prop-types

// далее просто скопировано все что было, кроме последней строки

class Article extends React.Component {
  state = {
    visible: false,
  }
  handleReadMoreClick = e => {
    e.preventDefault()
    this.setState({ visible: true })
  }
  render() {
    const { author, text, bigText } = this.props.data
    const { visible } = this.state
    return (
      <div className="article">
        <p className="news__author">{author}</p>
        <p className="news__text">{text}</p>
        {!visible && (
          <a
            onClick={this.handleReadMoreClick}
            href="#readmore"
            className="news__readmore"
          >
            Подробнее
          </a>
        )}
        {visible && <p className="news__big-text">{bigText}</p>}
      </div>
    )
  }
}

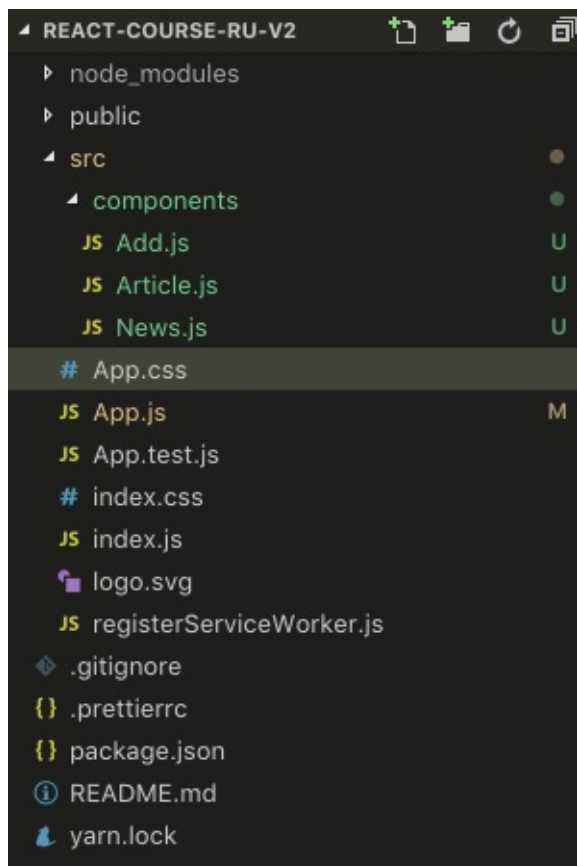
Article.propTypes = {
  data: PropTypes.shape({
    id: PropTypes.number.isRequired, // добавили id, это число, обязательно
    author: PropTypes.string.isRequired,
    text: PropTypes.string.isRequired,
    bigText: PropTypes.string.isRequired,
  }),
}

export { Article } // именованный экспорт
```

Остальные файлы оформите аналогично.

Должна получиться следующая структура:





При этом, необходимо добавить `import`'ы в *App.js* и в *News.js*

*src/App.js*

```
import React from 'react'
// импорт пакета prop-types удален, так как в этом файле prop-types не используется
import { Add } from './components/Add' // ./ = текущая директория,
import { News } from './components/News' // далее мы идем в директорию components и в
нужный компонент
import './App.css'

const myNews = [
  {
    id: 1,
    author: 'Саша Печкин',
    ...
```

*src/components/News.js*

```
import React from 'react'
import PropTypes from 'prop-types'
import { Article } from './Article' // идти в components не нужно, так как мы уже в эт
ой директории

class News extends React.Component {
  renderNews = () => {
    ...
  }
}
```

Я думаю принцип понятен: в каждом файле мы импортируем то, что нам нужно относительно этого файла.

- чтобы подняться на уровень выше - `'../'`
  - чтобы на два уровня выше - `'../../'`
  - чтобы начать поиск в текущей директории - `'./'`
  - чтобы найти компонент Name в текущей директории в components - `'./components/Name'`
  - чтобы импортировать из библиотеки - `'название библиотеки'`
- 

Наш *App.js* заметно похудел. Ориентироваться в коде стало удобнее.

*myNews* тоже можно выкинуть из этого файла.

*src/data/newsData.json* (экспорт json структуры по дефолту)

```
[
  {
    "id": 1,
    "author": "Саша Печкин",
    "text": "В четверг, четвертого числа...",
    "bigText":
      "в четыре с четвертью часа четыре чёрненьких чумазеньких чертёнка чертили чёрным
и чернилами чертёж."
  },
  {
    "id": 2,
    "author": "Просто Вася",
    "text": "Считаю, что $ должен стоить 35 рублей!",
    "bigText": "А евро 42!"
  },
  {
    "id": 3,
    "author": "Max Frontend",
    "text": "Прошло 2 года с прошлых учебников, а $ так и не стоит 35",
    "bigText": "А евро опять выше 70."
  },
  {
    "id": 4,
    "author": "Гость",
    "text": "Бесплатно. Без смс, про реакт, заходи - https: //maxpfrontend.ru",
    "bigText":
      "Еще есть группа VK, telegram и канал на youtube! Вся инфа на сайте, не реклама!"
  }
]
```

**Итого:** наш *App.js* нарядный:

*src/App.js*

```
import React from 'react'
import { Add } from './components/Add'
import { News } from './components/News'
import newsData from './data/newsData' // импорт по умолчанию
import './App.css'

class App extends React.Component {
  state = {
    news: newsData,
  }
  handleAddNews = data => {
    const nextNews = [data, ...this.state.news]
    this.setState({ news: nextNews })
  }
  render() {
    return (
      <React.Fragment>
        <Add onAddNews={this.handleAddNews} />
        <h3>Новости</h3>
        <News data={this.state.news} />
      </React.Fragment>
    )
  }
}

export default App
```

**Итого:** мы не плохо прибрались, разобрались в импортах. Читать App.js стало удобнее, каждый компонент живет в отдельном файле.

Есть несколько подходов к организации файлов в больших проектах, мы лишь сделали первый шаг в этом направлении.

[Исходный код.](#)

# Асинхронные запросы

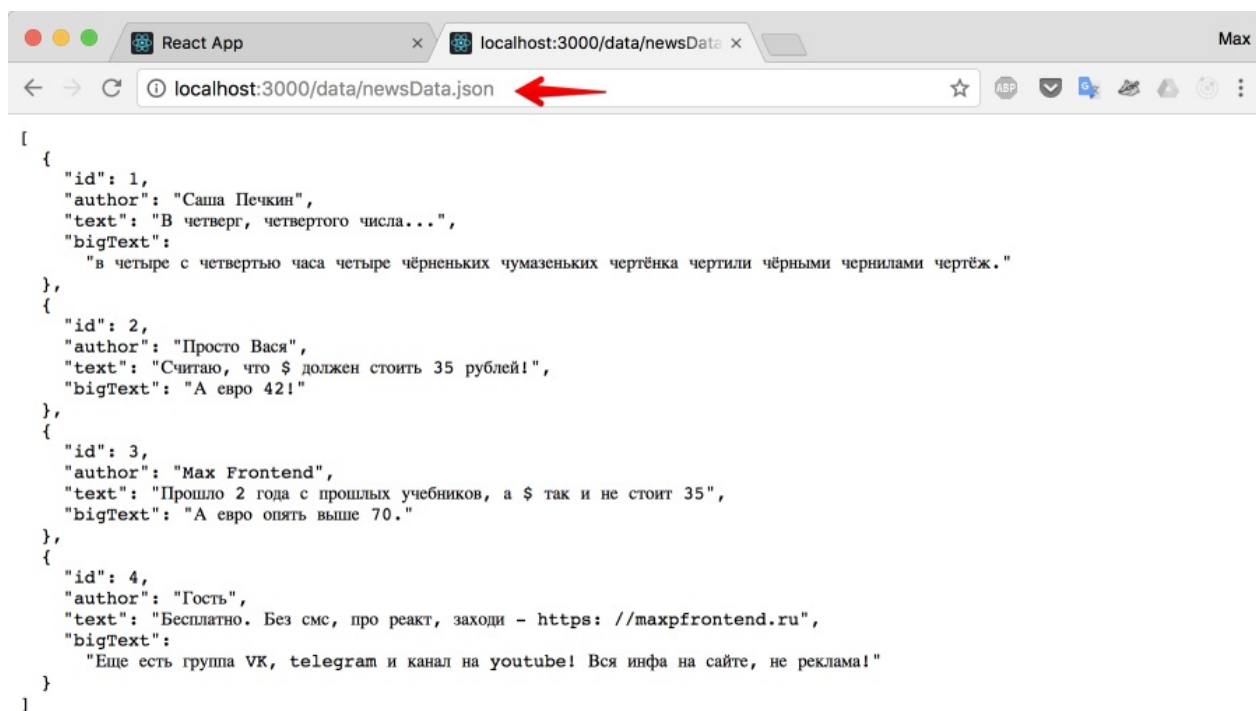
Нам все еще не нужен `redux`, ничего подобного.

CRA так устроен, что если вы положите что-нибудь в `public` директорию, это будет доступно по пути:

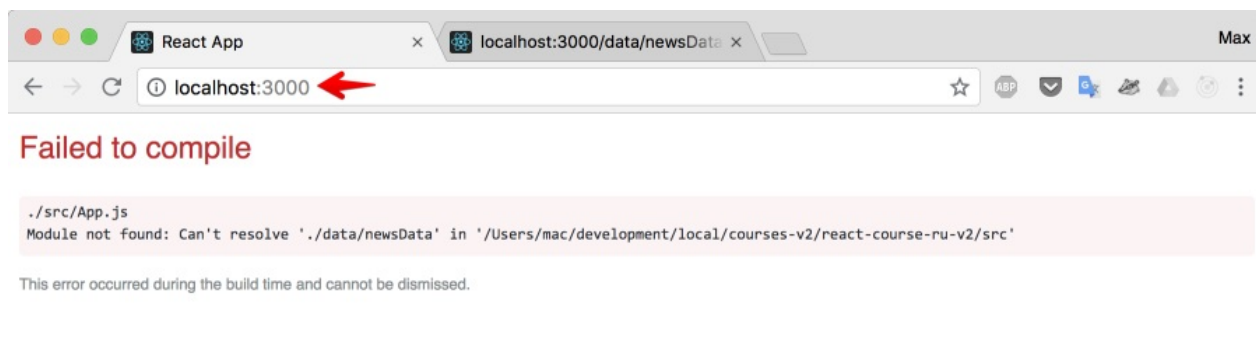
```
http://localhost:3000/название-директории/название-файла (но без слова public в пути)
```

Переместим наш `json` с новостями в `src/public/data/newsData.json`

Теперь его можно открыть GET-запросом на [localhost:3000/data/newsData.json](http://localhost:3000/data/newsData.json)



Конечно, при этом у нас сломался импорт в `App.js` (так как такого файла по старому пути нет):



Так как у нас есть доступ к файлу через GET-запрос, мы можем запросить его.

"Давайте представим задачу" (с)

У нас есть данные на сервере (новости в json), нам нужно их запросить и отобразить в списке. Пока запрос выполняется, мы хотим показывать юзеру надпись: "Загружаю..." вместо списка новостей, чтобы он не нервничал. Когда новости загружены - мы хотим отобразить их как раньше.

Что нового в этой задаче:

- как сделать асинхронный запрос (вопрос не про react) [1];
- где делать асинхронный запрос (про react) [2];

[1] - это вопрос про нативный js. Запрос будем делать с помощью [fetch](#).

[2] - запрос за данными следует начать в *componentDidMount*

---

Начнем с подготовки "состояния" и шаблона.

*src/App.js*

```
import React from 'react'
import { Add } from './components/Add'
import { News } from './components/News'
// удален импорт newsData
import './App.css'

class App extends React.Component {
  state = {
    news: null, // было newsData
    isLoading: false, // статус для манипуляций "прелоадером" ("Загружаю..." в нашем случае)
  }
  ...
}

export default App
```

В данный момент наше приложение не работает, но мы приготовили несколько важных вещей:

- во-первых, мы сможем на основе данных в `newsData` сказать:
  - если `newsData: null` :
    - если `isLoading: false` - значит данные еще не были загружены или произошла ошибка;

- если `isLoading: true` - данные еще загружаются
- если `newsData: []` (пустой массив) - значит новостей нет;
- если `newsData: [данные про новости]` - значит новости есть и они загружены;

В реакт-приложениях, все начинается с представления (описания) данных. Рисуйте в голове или на листочке, так как на основе такой шпаргалки, нам не составит труда сделать шаблон.

Начнем с составления выражений для шаблона. Первое:

```
{Array.isArray(news) && <News data={news} />}
```

То есть мы проверяем, если в `this.state.news` - массив - то рисуем компонент новости, он уже умеет рисовать "новостей нет" или список новостей.

Документация про [isArray](#) (MDN)

Второе условие:

```
{isLoading && <p>Загружаю...</p>}
```

Оформим все это в компоненте:

*src/App.js*

```
class App extends React.Component {
  state = {
    news: null,
    isLoading: false,
  }
  handleAddNews = data => {
    const nextNews = [data, ...this.state.news]
    this.setState({ news: nextNews })
  }
  render() {
    const { news, isLoading } = this.state // все необходимое взяли из state

    return (
      <React.Fragment>
        <Add onAddNews={this.handleAddNews} />
        <h3>Новости</h3>
        {isLoading && <p>Загружаю...</p>}
        {Array.isArray(news) && <News data={news} />}
      </React.Fragment>
    )
  }
}
```

Осталось сделать асинхронный вызов и установить правильное состояние.

Помните, мы с вами один раз описали, что количество новостей отображает цифры в зависимости от данных и когда стали добавлять новости - мы это место вообще не трогали, но счетчик работал корректно. Это **декларативный** подход. Так же и сейчас - мы в силу того, что я вижу всю картину, описали шаблон и как ему себя вести, а состояние разгружать будем на последнем шаге. Такой трюк может быть недоступен вам некоторое время, пока вы обучаетесь, поэтому пишите код как будет удобно, например делайте шаг за шагом что-то и воюйте с ошибками, главное - практика.

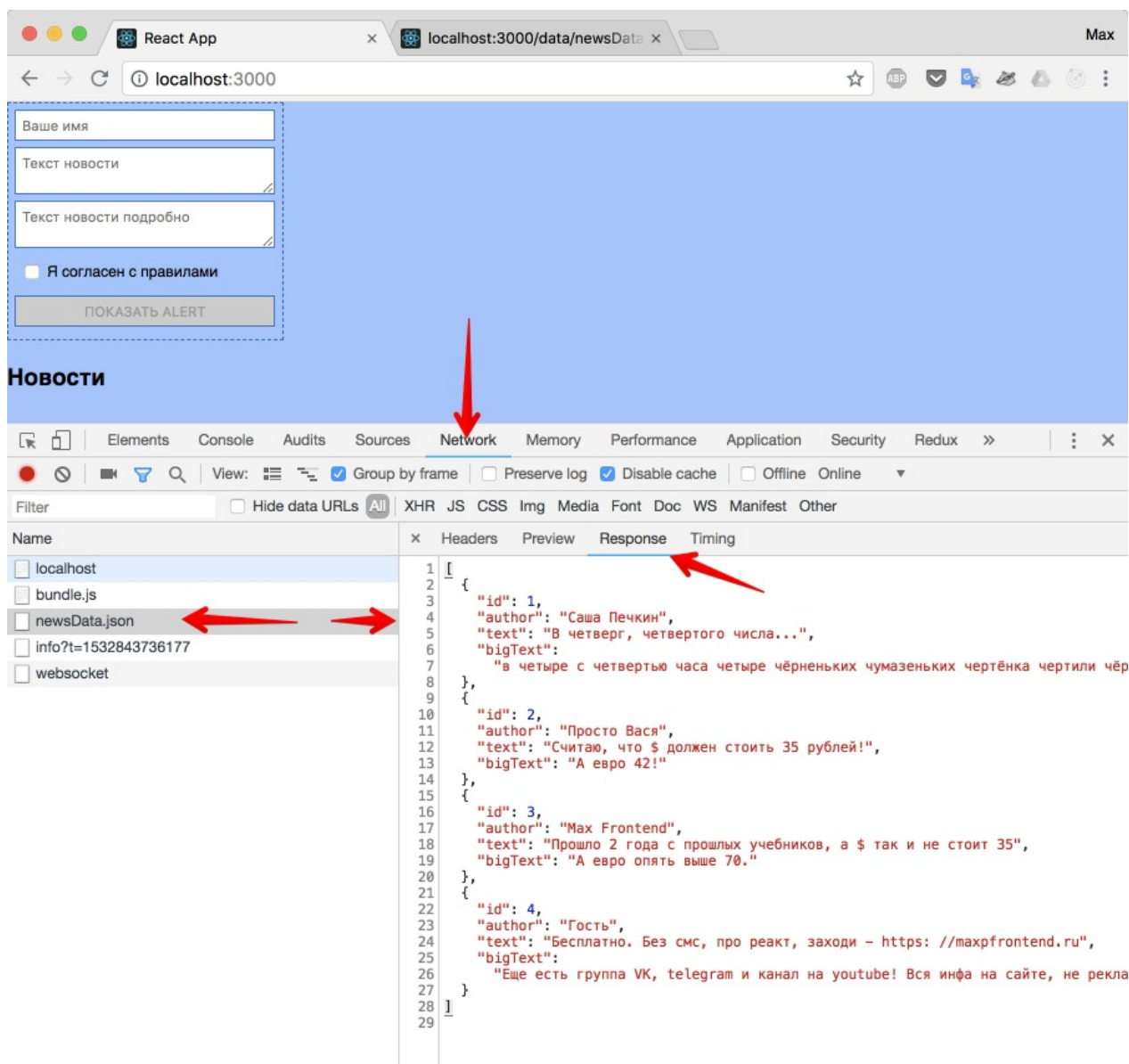
Вернемся к коду и сделаем fetch-запрос + console.log'и. Как я уже говорил, запрос за данными будем делать в момент, когда компонент уже примонтирован (то есть появился на странице, то есть нам нужен метод жизненного цикла - componentDidMount):

*src/App.js*

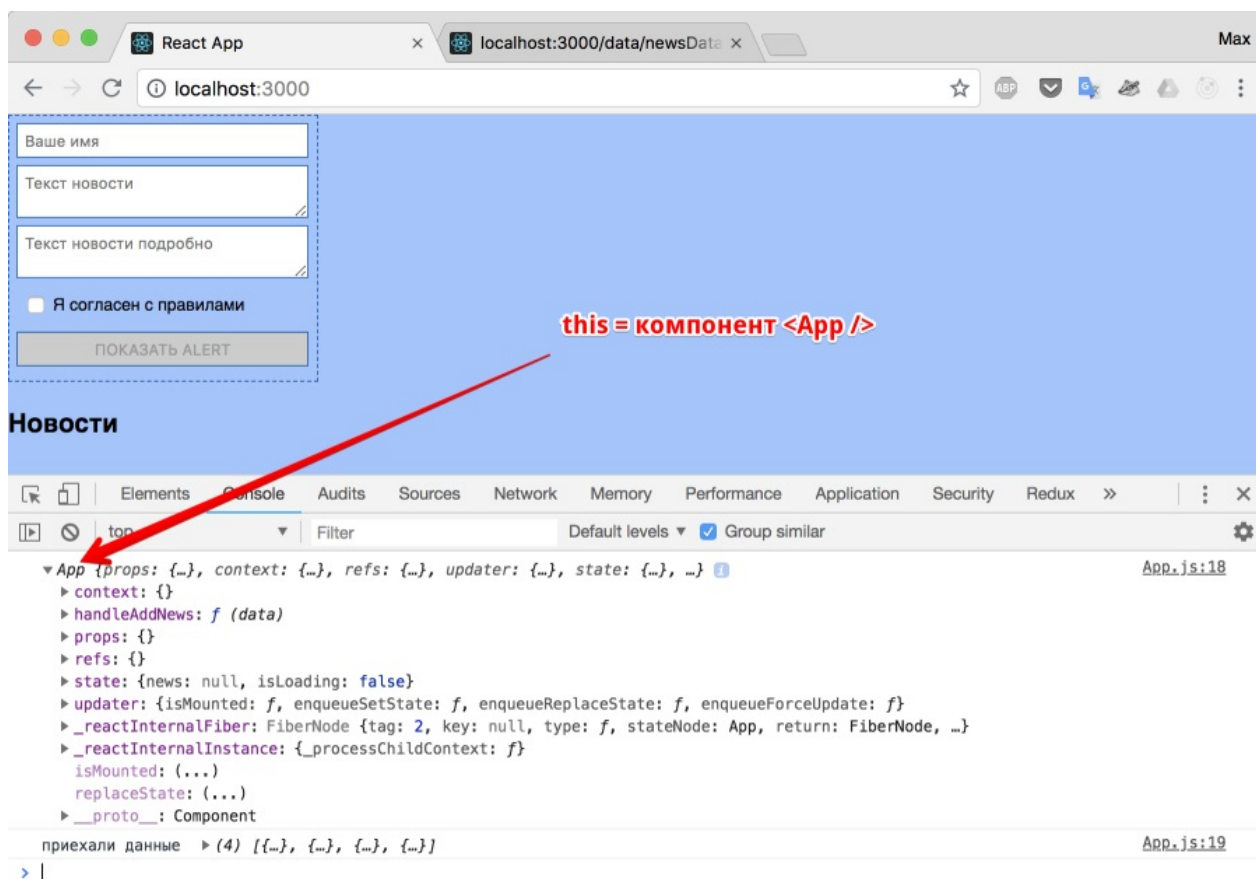
```
class App extends React.Component {  
  ...  
  componentDidMount() {  
    fetch('http://localhost:3000/data/newsData.json')  
      .then(response => {  
        return response.json()  
      })  
      .then(data => {  
        console.log(this)  
        console.log('приехали данные ', data)  
      })  
  }  
  ...  
}
```

Посмотрите в network, все работает:





Заглянем в console, и увидим, что так как мы используем стрелочные функции - мы не потеряли `this`.



Рассказывать про то как работает promise я не буду, но если у вас есть вопросы, вот мои любимые материалы:

- [Promise](#) (Кантор)
- [У нас проблемы с промисами](#) (перевод статьи на хабре)

Урок подходит к логическому завершению. Осталось лишь корректно обновлять состояние компонента.

**Задача:** до запроса - сделать `isLoading: true`, после завершения запроса - обновить `isLoading: false`, и в `news` положить данные, пришедшие с сервера.

(так как решение в пару строк, я сделаю отступ. Очень хочу чтобы вы попробовали сами)

.....

**Решение:**

`src/App.js`

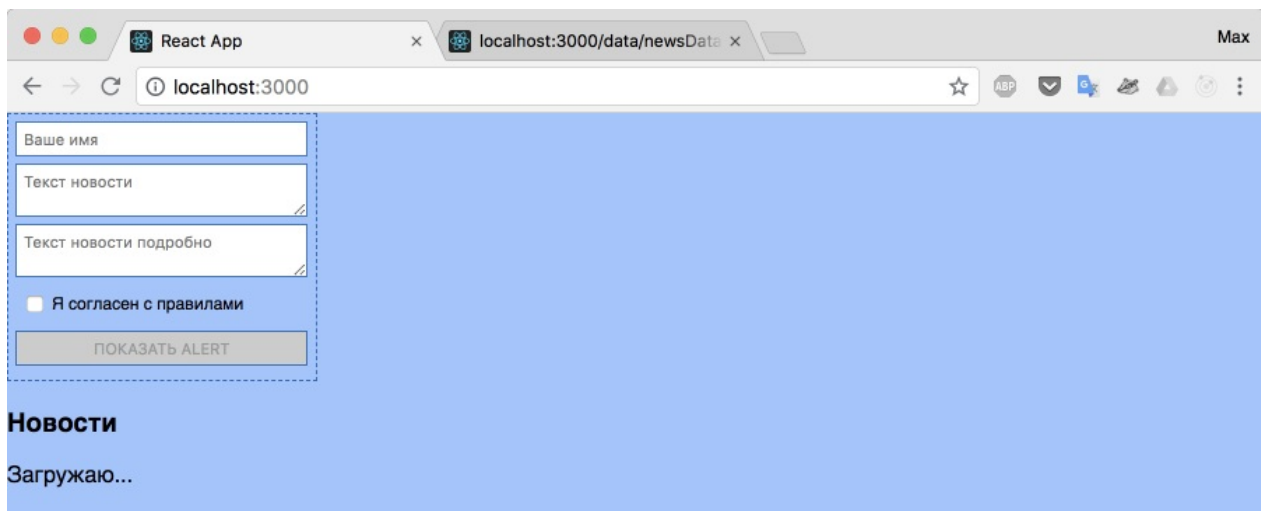
```
...
componentDidMount() {
  // ставим isLoading true,
  // то есть запрос за данными начался
  // фактически он начнется в строке с fetch,
  // но на переход от одной строки к другой
  // пройдут миллисекунды
  this.setState({ isLoading: true })
  fetch('http://localhost:3000/data/newsData.json')
    .then(response => {
      return response.json()
    })
    .then(data => {
      // запрос завершился успешно,
      // делаем isLoading: false
      // в news кладем пришедшие данные
      this.setState({ isLoading: false, news: data })
    })
}
...
```

Что интересно, у нас опять все работает. Мы не трогали компонент `<News />`, так как мы вновь просто изменили "источник данных".

Так как запрос за данными происходит на localhost, данные прилетают мгновенно. Давайте искусственно затормозим этот момент, чтобы увидеть как они "загружаются". Добавим таймаут, конечно же.

*src/App.js*

```
...
componentDidMount() {
  this.setState({ isLoading: true })
  fetch('http://localhost:3000/data/newsData.json')
    .then(response => {
      return response.json()
    })
    .then(data => {
      setTimeout(() => { // добавили задержку
        this.setState({ isLoading: false, news: data })
      }, 3000) // в три секунды
    })
}
...
```



Подождите три секунды и увидите как появится список новостей. Причем, что хочется отметить - опять нам помогает React. Изменился state -> вызвался render. Никаких дополнительных манипуляций ;)

И никакого Redux/Mobx и прочего. Задача решена без "дичи" в виде кучи библиотек, которые здесь не уместны. Поздравляю.

**Итого:** научились выполнять асинхронные запросы и показывать прелоадер.

[Исходный код.](#)

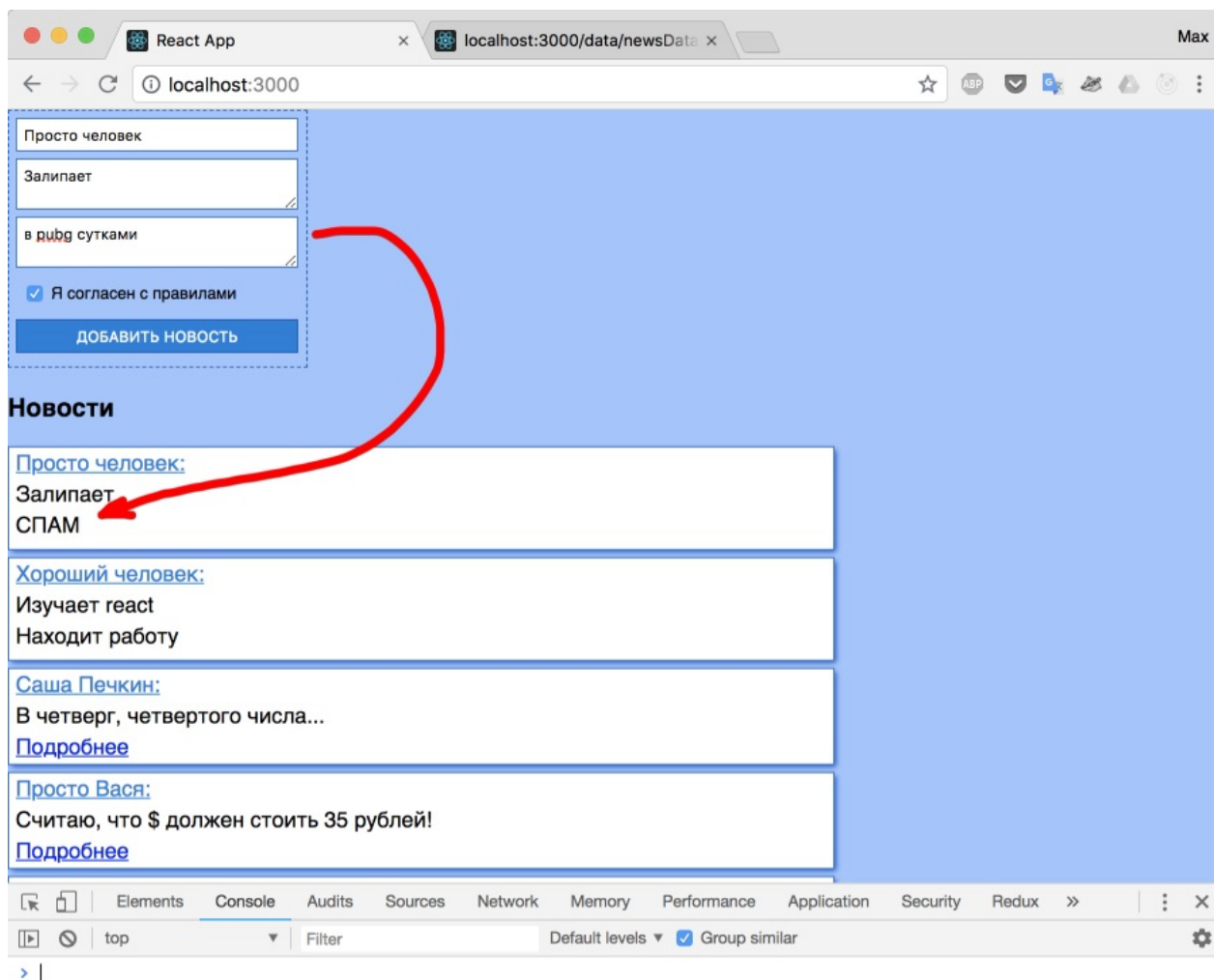
## Делаем спам-фильтр

Мне осталось осветить момент обновления данных. Ранее обработка происходила в `componentWillReceiveProps`, а сейчас в `getDerivedStateFromProps` (еще и `static`).

Для этого мне пришлось выдумать задачу, которая на самом деле решается в момент валидации новости на бэкэнде. Но представим, что наш бэкэндер очень занят, а менеджер говорит - пожалуйста, сделай как-нибудь, потом доделаем (ага!).

Задача: если пользователь, в добавленной новости в `bigText` ввел 'pubg' - будем пометать такую новость как СПАМ (то есть, в `bigText` вырезаем все, и вставляем строку "СПАМ").

Выглядит следующим образом:



p.s. я не против pubg :) просто соблюдайте баланс между отдыхом и развитием.



## ComponentWillReceiveProps (CWRP)

Давайте начнем со старого метода жизненного цикла ([componentWillReceiveProps](#)), который будет поддерживаться до React 17й версии. Нам нужно это знать, потому что много (очень много) кода уже написано и вам, наверняка, достанется такой проект.

*src/components/News.js*

```
class News extends React.Component {
  componentWillReceiveProps(nextProps) {
    console.log({ nextProps })
    console.log({ oldProps: this.props })
  }
  ...
}

...

export { News }
```

Скриншот веб-приложения и Redux DevTools. В веб-приложении есть форма с тремя текстовыми полями (содержащими 'qweqwe', '123' и '44123123'), чекбоксом 'Я согласен с правилами' и кнопкой 'ПОКАЗАТЬ ALERT'. Ниже формы находится секция 'Новости' с несколькими элементами. Один элемент новости выделен красной рамкой, и на него указывает красная стрелка с надписью 'свежая новость'. В Redux DevTools видны состояния 'nextProps' и 'oldProps'. Красные стрелки указывают на эти объекты с надписями 'новые props' и 'старые props' соответственно.

```

{nextProps: {}} ← новые props
  nextProps:
    data: Array(5)
      0: {id: 1532852281911, author: "qweqwe", text: "123", bigText: "44123123"} ← свежая новость
      1: {id: 1, author: "Саша Печкин", text: "В четверг, четвертого числа...", bigText: "в четыре с четвертью часа четыре чёрненьки"}
      2: {id: 2, author: "Просто Вася", text: "Считаю, что $ должен стоить 35 рублей!", bigText: "А евро 42!"}
      3: {id: 3, author: "Max Frontend", text: "Прошло 2 года с прошлых учебников, а $ так и не стоит 35", bigText: "А евро опять в"}
      4: {id: 4, author: "Гость", text: "Бесплатно. Без смс, про реакт, заходи - https://maxpffrontend.ru", bigText: "Еще есть груп"}
      length: 5
      __proto__: Array(0)
      __proto__: Object
      __proto__: Object

{oldProps: {}} ← старые props
  oldProps:
    data: Array(4)
      0: {id: 1, author: "Саша Печкин", text: "В четверг, четвертого числа...", bigText: "в четыре с четвертью часа четыре чёрненьки"}
      1: {id: 2, author: "Просто Вася", text: "Считаю, что $ должен стоить 35 рублей!", bigText: "А евро 42!"}
      2: {id: 3, author: "Max Frontend", text: "Прошло 2 года с прошлых учебников, а $ так и не стоит 35", bigText: "А евро опять в"}
      3: {id: 4, author: "Гость", text: "Бесплатно. Без смс, про реакт, заходи - https://maxpffrontend.ru", bigText: "Еще есть груп"}
      length: 4
      __proto__: Array(0)
      __proto__: Object
      __proto__: Object

```

Давайте, мы отвлечемся на секунду и исправим старую ошибку - в компоненте `<Add />` у кнопки, сделайте текст "Добавить новость", вместо "Показать Alert".

Продолжим: CWRP в первом аргументе принимает "будущие props", значит мы можем по ним пробежаться, найти новость с "rubg" фрагментом, если она есть и пометить ее как "СПАМ".

Одно НО, так как мы хотим что-то изменять, значит у нас у компонента появляется состояние! **Обращаю внимание:** данная задача сейчас решается не оптимальным способом, мы просто учимся следующим моментам:

- как сделать state на основе props?
- как изменить состояние, на основе вновь пришедших props?

`src/components/News.js`



```
...
class News extends React.Component {

  state = { // создали состояние
    filteredNews: this.props.data,
  }

  componentWillReceiveProps(nextProps) {
    console.log({ nextProps })
    console.log({ oldProps: this.props })
  }

  renderNews = () => {
    const { filteredNews } = this.state // используем состояние
    let newsTemplate = null

    if (filteredNews.length) { // везде data заменена на filteredNews
      newsTemplate = filteredNews.map(function(item) {
        return <Article key={item.id} data={item} />
      })
    } else {
      newsTemplate = <p>К сожалению новостей нет</p>
    }

    return newsTemplate
  }

  render() {
    const { filteredNews } = this.state // аналогично, используем состояние

    return (
      <div className="news">
        {this.renderNews()}
        {filteredNews.length ? (
          <strong className={'news__count'}>
            Всего новостей: {filteredNews.length}
          </strong>
        ) : null}
      </div>
    )
  }
}
```

Изменений немного, просто внимательно пробежитесь по файлу - теперь данные берем из `filteredNews`, но при этом они нам изначально приходят в `props` (поэтому `propTypes` оставили без изменений).

Внутри *CWRP* можно безопасно использовать `setState`, так как это не приведет к дополнительной (лишней) перерисовке.

*src/components/News.js*

```
componentWillReceiveProps(nextProps) {  
  let nextFilteredNews = [...nextProps.data]  
  
  nextFilteredNews.forEach((item, index) => {  
    if (item.bigText.toLowerCase().indexOf('pubg') !== -1) {  
      item.bigText = 'СПАМ'  
    }  
  })  
  
  this.setState({ filteredNews: nextFilteredNews })  
}
```

В данном фрагменте ничего необычного, для тех кто в теме основ (я понимаю, что фраза "знать основы" набила оскомину, но их действительно нужно знать).

В `nextFilteredNews` клонировали весь массив новостей из "будущих" пропс, затем пробежались по нему, в `if` выяснили, есть ли `pubg` в введенном в `bigText` тексте, и если есть, заменили на "СПАМ".

Для тех, у кого есть пробелы:

- [toLowerCase](#) (MDN)
- [indexOf](#) (MDN)

---

### Итого:

- научились создавать *state* на основе *props*;
- научились обновлять *state* на основе новых *props* без лишней перерисовки;

[Исходный код](#)

## static getDerivedStateFromProps

Сигнатура метода: `static getDerivedStateFromProps(props, state)` и [документация](#), в которой описано, что данный метод нужен для очень редких случаев. Мы сами выдумали себе задачу и ограничения для решения, поэтому он нам пригодится.

Этот метод жизненного цикла в целом похож на *CWRP*, но есть отличия:

- так как метод `static` - нет доступа к `this`
- из метода должно возвращаться новое состояние или `null`, если "обновлений" не планируется.
- для первого рендера (*initial render*) - тоже вызывается

Один из лучших способов понять, что происходит - `console.log`

*src/components/News.js*

```
...
static getDerivedStateFromProps(props, state) {
  console.log(props)
  console.log(state)
  return {
    filteredNews: props.data,
  }
}

// удалите componentWillReceiveProps
...
```

Попробуйте добавить новость.

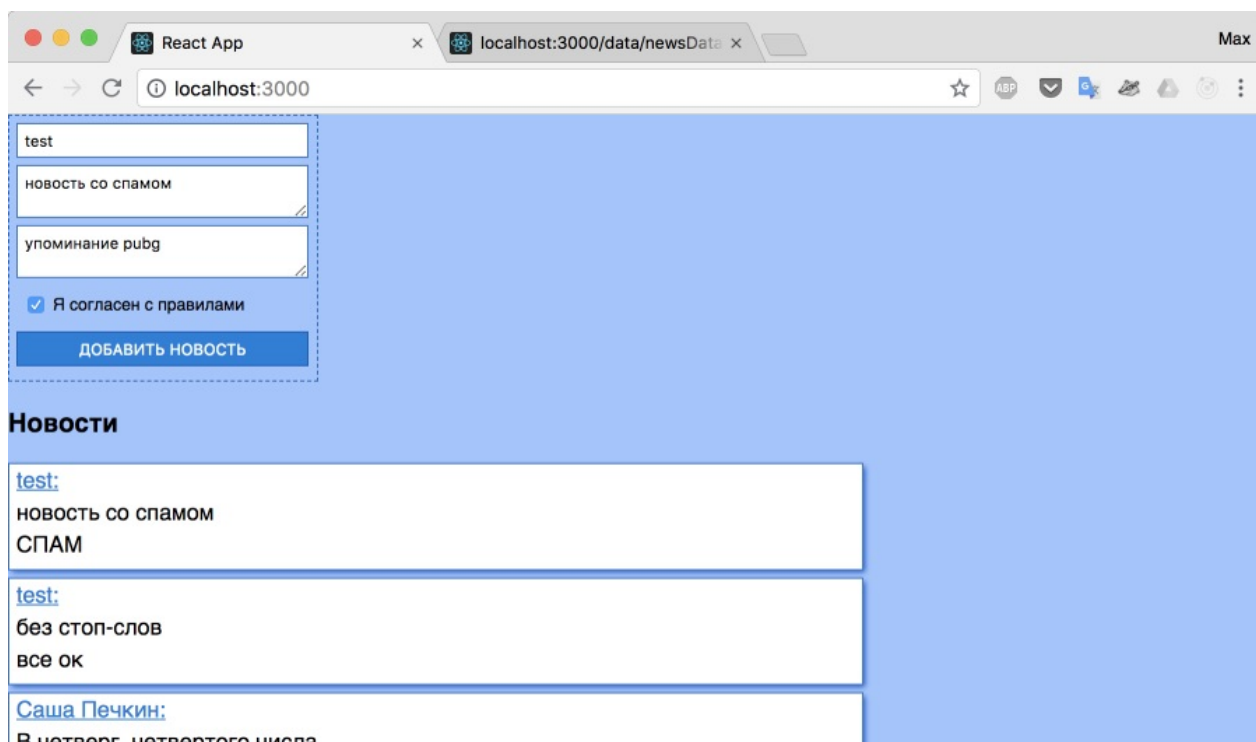
Восстановим код, который был в *CWRP*, но уже для *getDerivedStateFromProps* (*gDSFR*)

*src/components/News.js*

```
static getDerivedStateFromProps(props, state) {
  let nextFilteredNews = [...props.data] // было nextProps - переименовали

  nextFilteredNews.forEach((item, index) => {
    if (item.bigText.toLowerCase().indexOf('pubg') !== -1) {
      item.bigText = 'СПАМ'
    }
  })

  return { // возвращаем новое состояние
    filteredNews: nextFilteredNews,
  }
}
```



**Итого:** научились использовать современный `getDerivedStateFromProps`

[Исходный код](#)

# Порефакторим

В течении всего учебника я учил вас думать о данных, а потом в CWP и gDSFR взял и сделал из образно *stateless* (хоть он и был через *class* - состояния у него не было) компонента - *statefull*. Это было сделано для удобства объяснений.

Смотрите, если мы откатим наш `<News />` на два урока назад (когда компонент просто получал props), то мы сможем в `<App />` использовать gDSFR и там "рубить спам". Таким образом, мы бы опять решили задачу без изменения *stateless* компонента.

Было: компонент `<News />` умел отображать данные. Стало: компонент `<News />` умеет отображать данные и помечать спам.

**Задача:** обрабатывать данные в `<App />`, вернуть `<News />` к прежнему "тупому" образу жизни.

**Подсказка:** вы запросто можете сделать все что нужно в `<App />`, так как мы только что отработали этот прием.

**Подсказка:** в `<App />` новости лежат в *state*, а не в *props*.

Напоминаю, как выглядел `<News />`:

*src/components/News.js*

```
import React from 'react'
import PropTypes from 'prop-types'
import { Article } from './Article'

class News extends React.Component {
  renderNews = () => {
    const { data } = this.props
    let newsTemplate = null

    if (data.length) {
      newsTemplate = data.map(function(item) {
        return <Article key={item.id} data={item} />
      })
    } else {
      newsTemplate = <p>К сожалению новостей нет</p>
    }

    return newsTemplate
  }
  render() {
    const { data } = this.props

    return (
      <div className="news">
        {this.renderNews()}
        {data.length ? (
          <strong className={'news__count'}>
            Всего новостей: {data.length}
          </strong>
        ) : null}
      </div>
    )
  }
}

News.propTypes = {
  data: PropTypes.array.isRequired,
}

export { News }
```

## Решение

Полный код компонента `<App />`

`src/App.js`

```
import React from 'react'
```

```

import { Add } from './components/Add'
import { News } from './components/News'
import './App.css'

class App extends React.Component {
  state = {
    news: null,
    isLoading: false,
  }
  static getDerivedStateFromProps(props, state) {
    let nextFilteredNews

    // смотрим в state.news (ранее смотрели в props)
    // и проверяем, чтобы не клоинировать null
    // например, в момент первой отрисовки
    if (Array.isArray(state.news)) {
      nextFilteredNews = [...state.news]

      nextFilteredNews.forEach((item, index) => {
        if (item.bigText.toLowerCase().indexOf('pubg') !== -1) {
          item.bigText = 'СПАМ'
        }
      })
    }

    return {
      filteredNews: nextFilteredNews,
    }
  }

  return null
}

componentDidMount() {
  this.setState({ isLoading: true })
  fetch('http://localhost:3000/data/newsData.json')
    .then(response => {
      return response.json()
    })
    .then(data => {
      setTimeout(() => {
        this.setState({ isLoading: false, news: data })
      }, 1000) // изменил таймер на 1000, чтобы не ждать долго
    })
}

handleAddNews = data => {
  const nextNews = [data, ...this.state.news]
  this.setState({ news: nextNews })
}

render() {
  const { news, isLoading } = this.state

  return (
    <React.Fragment>
      <Add onAddNews={this.handleAddNews} />
    </React.Fragment>
  )
}

```



```
    <h3>Новости</h3>
    {isLoading && <p>Загружаю...</p>}
    {Array.isArray(news) && <News data={news} />}
  </React.Fragment>
)
}
}

export default App
```

---

**Итого:**

[Исходный код](#)

## Заключение

Что сказать? Вы молодцы. Основы React'a у вас в голове. Практиковались пока читали? Если нет - еще раз прочитайте + задачки поделайте.

Если да - **welcome** (добро пожаловать). Дальше больше: реакт не заботится о том, как вы будете хранить данные в вашем приложении. Гонять все из `state` родителя вниз через props'ы не удобно.

Я сторонник Redux. По нему есть старое руководство, которое в данный момент я уже переписываю.

Подписывайтесь на сообщества, где я провожу вебинары и стримы:

- [Расписание стримов и вебинаров](#) (на сайте есть текстовые версии вебинаров)
- [Youtube канал](#) с записями вебинаров и стримов
- Группа [vkontakte](#)
- Канал в [telegram](#)
- [Twitter](#)
- [Facebook](#)

---

Вы можете [поддержать проект](#), мне будет очень приятно.

Спасибо, что уделили время этому учебнику. Посоветуйте его другу.

---

Максим Пацианский, Июль 2018г.