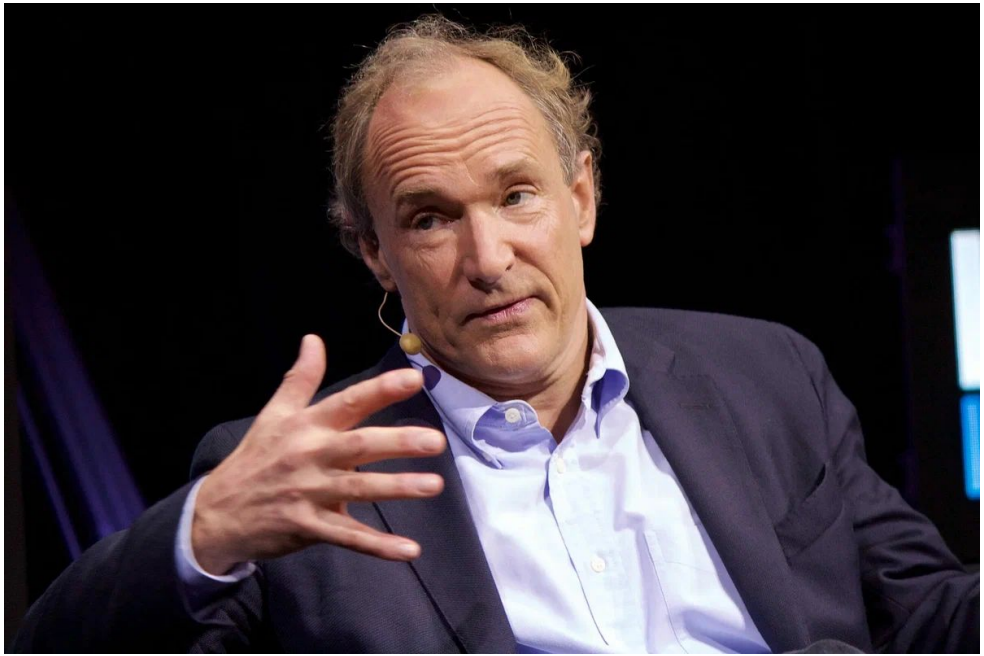
React.js

Развитие фронтенд-технологий

Зарождение концепции WorldWideWeb (WWW)

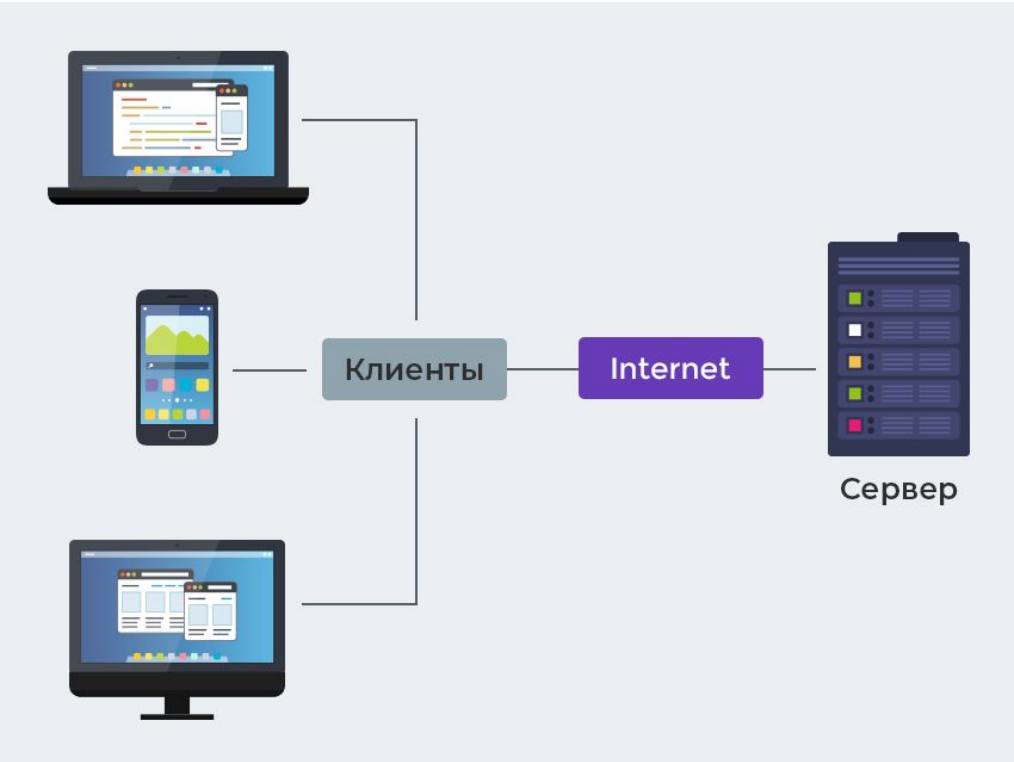


1989 - доклад о необходимости создания распределенной гипертекстовой системе (компания CERN)

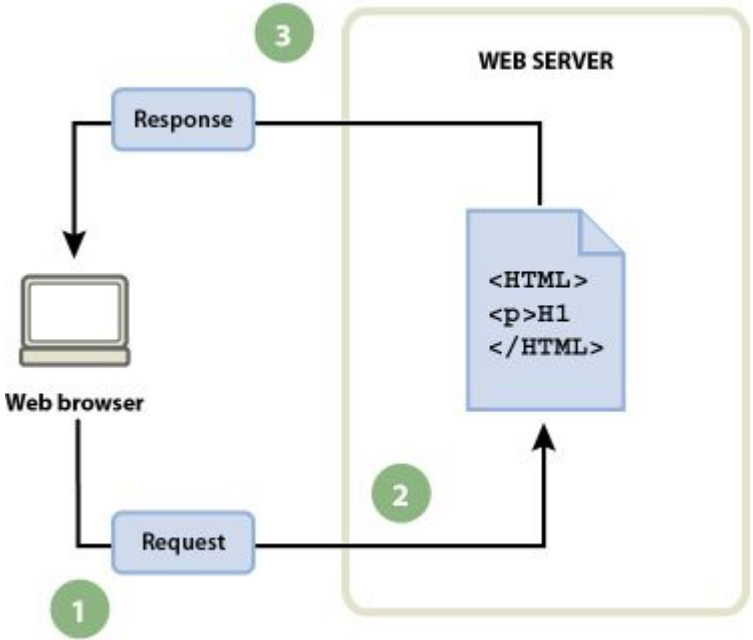
1991 - появление языка HTML и первого браузера WorldWideWeb

1993 - публикация исходного кода WorldWideWeb

Общая структура веб-приложений



Структура веб-приложения Первого этапа



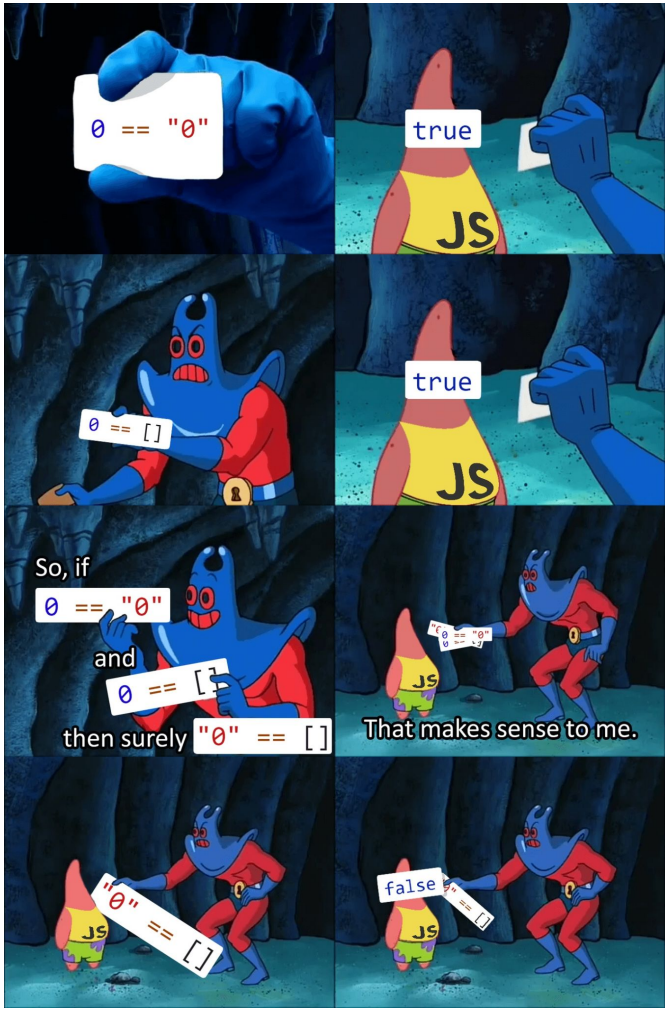
Клиент

- отправляет запрос на получение некоторого ресурса;
- отображает результат запроса.

Сервер

- хранит и отдает статические HTML-страницы.

Браузерные войны



В результате:

Разработчики сталкивались со сложностями проприетарных решений из-за чего приходилось поддерживать варианты веб-страниц

Стандартизация веб-технологий

1996 - стандартизация языка HTML

1997 - принятие спецификации ECMAScript на основе существующих технологий JavaScript и JScript

Развитие фронтенд-технологий. Шаблонизаторы

Шаблонизатор ERB (Ruby)

```
<h1><%= @name %> (<%= @code %>)</h1>

<ul>
    <% @features.each do |f| %>
      <li><b><%= f %></b></li>
    <% end %>
  </ul>

<p>
   <% if @cost < 10 %>
      <b>Всего <%= @cost %>!!!</b>
   <% end %>
</p>
```

Шаблонизатор Mustache

```
<p>Hello</p> {{{name}}}
{{#if_else}}
  <h2> - IF -</h2>
{{/if_else}}

{{#listt}}
    <p>{{name}} | {{email}}</p>
{{/listt}}
{{> footer}}
```

Развитие фронтенд-технологий. Шаблонизаторы

Какие возможности давала технология:

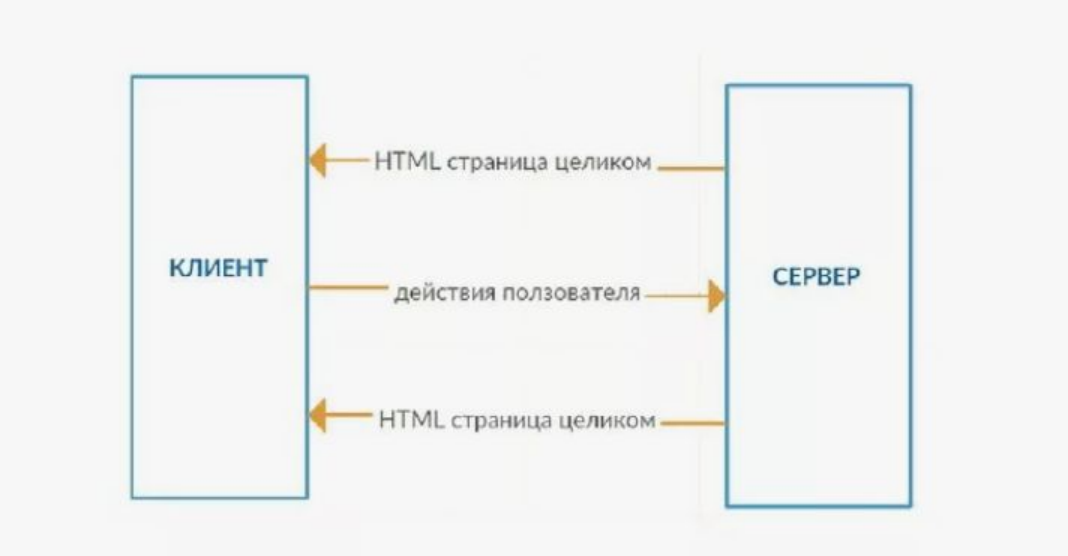
- отделение представления от данных;
- разбиение фрагментов разметки на отдельные компоненты;
- переиспользование шаблонов в различных проектах;
- облегчение процесса рефакторинга.

Развитие фронтенд-технологий. JQuery

Какие возможности давала технология:

- написание кода, работающего во всех браузерах;
- удобное манипулирование DOM;
- выполнение асинхронных запросов (AJAX);
- применение анимаций к элементам страницы;
- добавление обработчиков событий для элементов страницы.

Структура классического веб-приложения Второго этапа



Клиент

- отрисовывает HTML-страничку, переданную сервером;
- асинхронные запросы;
- манипуляция DOM;
- взаимодействие с сервисом посредством форм.

Сервер

- генерация динамических страниц;
- обработка пользовательских запросов;
- взаимодействие с хранилищем данных.

Структура классического веб-приложения Второго этапа

Недостатки:

- большая нагрузка на сервер;
- низкая отзывчивость приложения;
- долгий период ожидания ответа и перезагрузка страницы;
- лишний трафик в сети.

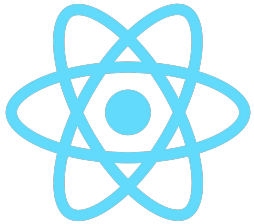
Предпосылки к появлению современных фронтенд-фреймворков

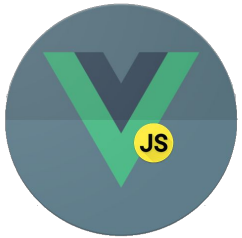
Сложность разработки приложений с использованием JQuery

- логика приложения разбросана по файлу в виде обработчиков;
- появление состояния у приложения и сложности манипулирования над ним;
- прямое манипулирование DOM-деревом;
- невозможность переиспользования фрагментов кода.

Кроме того, отрисовкой пользовательского интерфейса может заниматься клиентское приложение!

Современные веб-фреймворки и их концепции

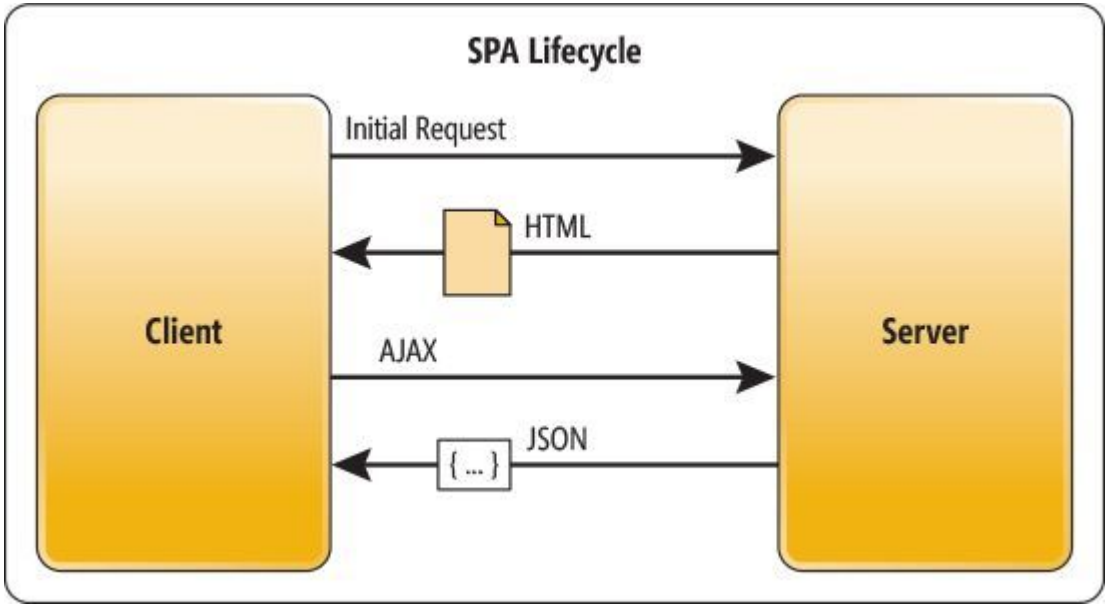




Компонентный подход

- любое приложение разбивается на независимые переиспользуемые кусочки (компоненты);
- компоненты объединяются в иерархии, образуя более сложные элементы интерфейса;
- приложение в определенный момент времени представляет собой некоторое состояние;
- взаимодействие с пользователем приводит изменяет состояние;
- компонент реагирует на изменение состояния и обновляет пользовательский интерфейс.

Структура классического веб-приложения Третьего этапа. SPA



Клиент

- хранение статических файлов;
- отрисовка пользовательского интерфейса;
- отправка асинхронных запросов;
- обновление пользовательского интерфейса

Сервер

- обработка клиентских запросов;
- взаимодействие с хранилищем данных.

Знакомство с библиотекой React

Назначение библиотеки React

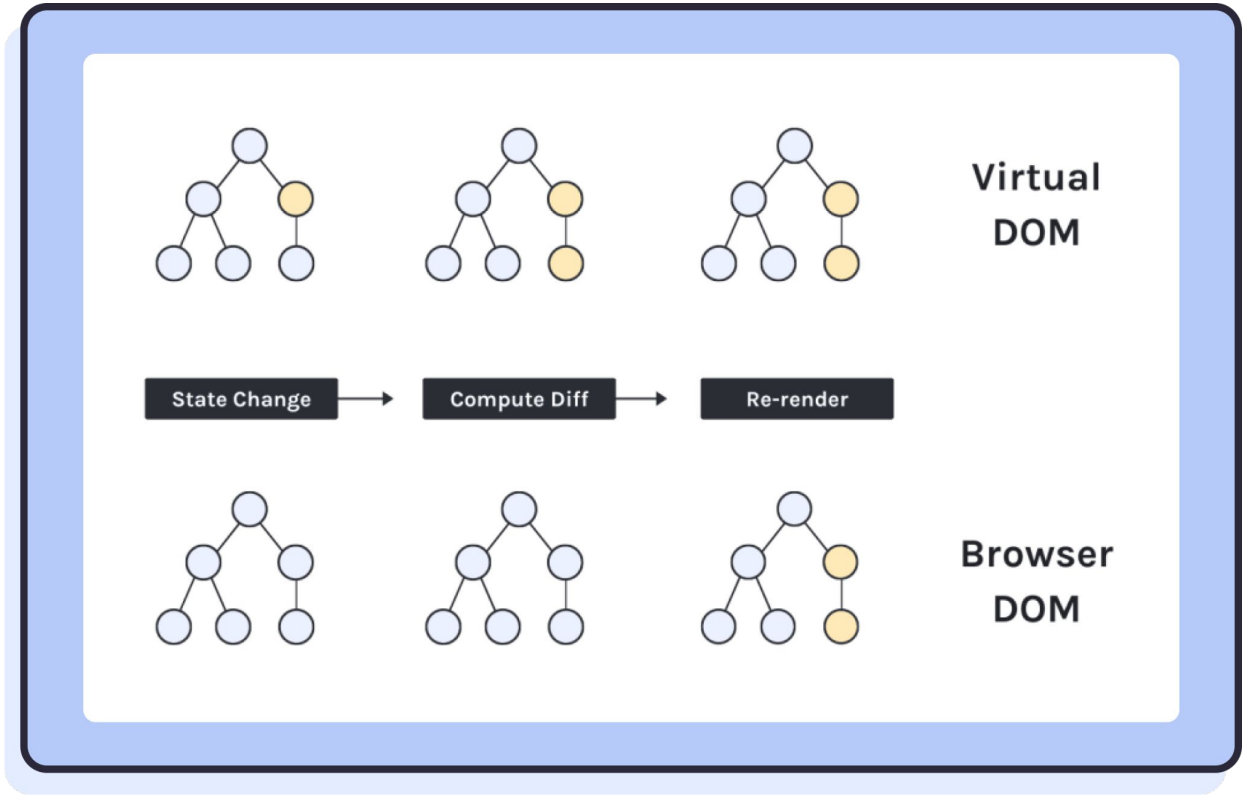
Что мне как разработчику дает React?

- движок рендеринга пользовательского интерфейса;
- декларативный подход описания компонент и формирования связей между ними;
- близкий к HTML синтаксис;
- инструменты для работы с состоянием приложения;
- различные подходы к написанию компонент (классовые и функциональные);
- виртуальный DOM (Virtual Dom).

Пример React компонента

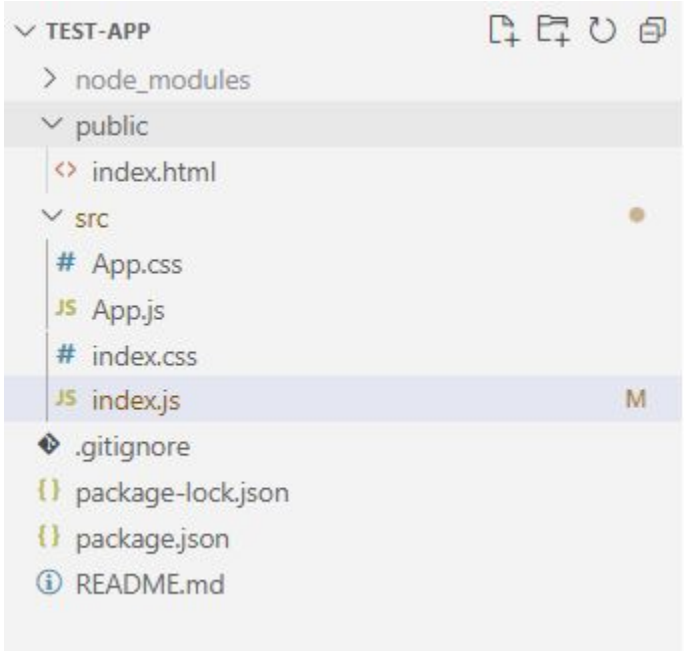
```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Привет, {this.props.name}
      </div>
    );
  }
}
```

Virtual DOM


Virtual
DOM
State Change
Compute Diff
Re-render
Browser
DOM

API библиотеки React.js

Структура базового проекта



Структура базового проекта

index.html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

App.js

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

export function App() {
  return (
    <h1>Hello, World!</h1>
  );
}
```

index.js

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

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Результат в браузере

Синтаксис JSX

```jsx
import { Component } from 'react'

export class Example1 extends Component {
    constructor(props) {
        super(props);
    }

    render() {
        return (
            <article>
                <h1>My First Component</h1>
                <ol>
                    <li>Components: UI Building Blocks</li>
                    <li>Defining a Component</li>
                    <li>Using a Component</li>
                </ol>
            </article>
        );
    }
}
```

Синтаксис JSX

```jsx
import { Component } from 'react'
import "./style.css";

export class Example2 extends Component {
    render() {
        return (
            <div
                className='bg-black'
                style={{
                    width: '400px',
                    height: '200px',
                    color: 'white'
                }}
                onClick={function () { console.log('Клик') }}>
                Какой-то текст...
            </div>
        );
    }
}
```

```jsx
import { Component } from 'react'
import "./style.css";

export class Example2 extends Component {
    render() {
        return (
            <div
                className='bg-black'
                style={{
                    width: '400px',
                    height: '200px',
                    color: 'white'
                }}
                onClick={this.handleClick}>
                Какой-то текст...
            </div>
        );
    }

    handleClick() {
        console.log('click')
    }
}
```

```html
<div class="bg-black" style="width: 400px; height: 200px; color: white" onclick="() => { console.log('click') }">
  Какой-то текст...
</div>
```

JSX является объектом

Синтаксис JSX

```jsx
const element = (
  <h1 className="greeting">
    Привет, мир!
  </h1>
);
```

Результат компиляции

```jsx
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Привет, мир!'
);
```

Соответствующая HTML-разметка

```html
<h1 class="greeting">Привет, мир!</h1>
```

JSX является объектом

```
import { Component } from 'react';

export class Example1 extends Component {
    constructor(props) {
        super(props);
    }

    render() {
        return React.createElement(
            'article',
            null,
            React.createElement(
                'h1',
                null,
                'My First Component'
            ),
            React.createElement(
                'ol',
                null,
                React.createElement(
                    'li',
                    null,
                    'Components: UI Building Blocks'
                ),
                React.createElement(
                    'li',
                    null,
                    'Defining a Component'
                ),
                React.createElement(
                    'li',
                    null,
                    'Using a Component'
                )
            )
        );
    }
}
```

Анатомия JSX

```jsx
const element = (
  <h1 className="greeting">
    Привет, мир!
  </h1>
);
```

className - пропс

“Привет, мир!” - дочерний элемент

```jsx
<article>
    <h1>My First Component</h1>
    <ol>
        <li>Components: UI Building Blocks</li>
        <li>Defining a Component</li>
        <li>Using a Component</li>
    </ol>
</article>
```

h1 - дочерний элемент для article

Определение компонента

Определение функционального компонента

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

Определение классового компонента

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

Структура компоненты

```
1   import { Component } from 'react'
2   import "./style.css";
3
4   export class Example3 extends Component {
5       render() {
6           return <Welcome name="мир" />
7       }
8   }
9
10  class Welcome extends Component {
11      render() {
12          return <h1>Привет, {this.props.name}!</h1>;
13      }
14  }
15
```

Композиция компонентов

```
1   import { Component } from 'react'
2   import "./style.css";
3
4   export class Example3 extends Component {
5       render() {
6           return (
7               <div>
8                   <Welcome name="Боб" />
9                   <Welcome name="Стив" />
10                  <Welcome name="Фрэнк" />
11              </div>
12          )
13      }
14  }
15
16  class Welcome extends Component {
17      render() {
18          return <h1>Привет, {this.props.name}!</h1>;
19      }
20  }
```

Описание компоненты отображения

```
21  class ProfileCard extends Component {
22      render() {
23          const { avatarUrl, name, description, onClick, ...props } = this.props;
24
25          return (
26              <div className='profile-card' onClick={onClick} {...props}>
27                  <img className='profile-card__avatar' src={avatarUrl} alt='' />
28                  <div className='profile-card__personal-info'>
29                      <div className='profile-card__name'>
30                          {name}
31                      </div>
32                      <div className='profile-card__description'>
33                          {description}
34                      </div>
35                  </div>
36              </div>
37          )
38      }
39  }
```

Описание компоненты отображения

```
1   import { Component } from 'react'
2   import './style.css';
3
4   import img from '../images/mem-rabbit.jpg';
5
6   export class Example4 extends Component {
7       render() {
8           return (
9               <ProfileCard
10                  name='Bob'
11                  description='Some interesting description'
12                  avatarUrl={img}
13                  onClick={() => alert('Profile card')}
14              />
15          );
16      }
17  }
```

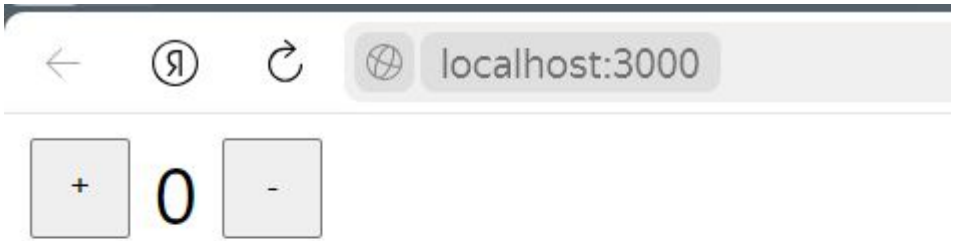
Описание компоненты отображения

Работа с состоянием. Определение компонента счетчика

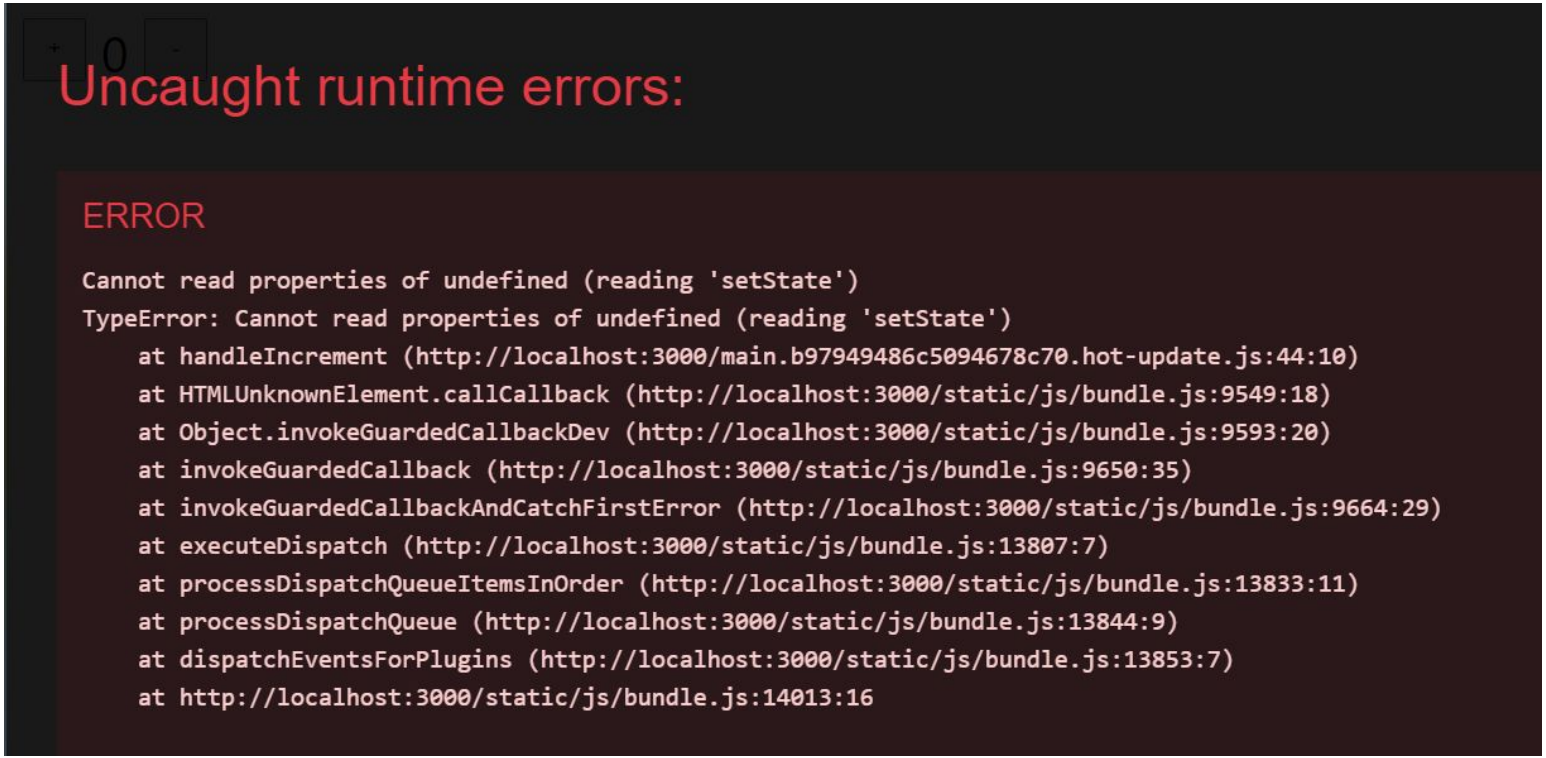
```
36  class Counter extends Component {
37      render() {
38          const { value, onIncrement, onDecrement, ...props } = this.props;
39
40          return (
41              <div className='counter' {...props}>
42                  <button className='counter__button counter__button_inc' onClick={onIncrement}>
43                      +
44                  </button>
45                  <span className='counter__input'>
46                      {value}
47                  </span>
48                  <button className='counter__button counter__button_dec' onClick={onDecrement}>
49                      -
50                  </button>
51              </div>
52          )
53      }
54  }
```

Работа с состоянием. Определение состояния

```
4   export class Example5 extends Component {
5       constructor(props) {
6           super(props);
7
8           this.state = {
9               value: 0
10          };
11      }
12
13      render() {
14          return (
15              <Counter
16                  value={this.state.value}
17                  onIncrement={this.handleIncrement}
18                  onDecrement={this.handleDecrement}
19              />
20          );
21      }
22
23      handleIncrement() {
24          this.setState({
25              value: this.state.value + 1
26          });
27      }
28
29      handleDecrement() {
30          this.setState({
31              value: this.state.value - 1
32          });
33      }
34  }
```



Работа с состоянием. Определение состояния



Потеря контекста this

```
handleIncrement() {
    // console.log(this) -> undefined

    this.setState({
        value: this.state.value + 1
    });
}
```

Пример 1

```
<script>
  function a() {
    console.log(this)
  }
</script>
```

Вывод

```
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
```

Потеря контекста this

Пример 2

```
<script>
  function greet() {
    console.log(`Hello, ${this.username} ${this.age}`)
  }

  greet();

  const obj = {
    username: 'Bob',
    age: 21,
    greet: greet
  }

  obj.greet()
```

Вывод

```
Hello, undefined undefined
Hello, Bob 21
```

Для обычных функций контекст this определяется в момент вызова функции.

Контекст представляет из себя объект слева от вызова функции

Если функция объявлена в глобальной области, то контекст исполнения Window

Потеря контекста this

Пример 3

```
<script>
  const obj2 = {
    username: 'Bob',
    age: 21,
    greet: () => {
      console.log(this.username);
    }
  }

  obj2.greet()

</script>
```

Вывод

```
undefined
```

Потеря контекста this

Пример 4

```
<script>
  function foo() {
    console.log(this);

    return function () {
      console.log(this);
    }
  }

  const test = foo();
  test();

  const obj = {
    username: 'Bob',
    age: 21,
    foo: test
  }

  obj.foo()
</script>
```

Вывод

```
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
▶ {username: 'Bob', age: 21, foo: ƒ}
```

Привязка контекста

Пример 5

```
<script>
  function foo() {
    console.log(`${this.username} ${this.age}`)
  }

  const obj = {
    username: 'Bob',
    age: 21
  }

  const boo = foo.bind(obj)
  boo()
</script>
```

Вывод

```
Bob 21
```

Привязка контекста

Пример 5

```
<script>
  class Person {
    #name = 'Bob'
    #age = 21

    toString() {
      console.log(this)
      console.log(`${this.#name} ${this.#age}`);
    }
  }

  function foo(boo) {
    boo();
  }

  const bob = new Person();
  foo(bob.toString)
</script>
```

Вывод

```
undefined
▶ Uncaught
TypeError: Cannot read properties of undefined (reading '#name')
    at toString ((индекс):20:29)
    at foo ((индекс):25:7)
    at (индекс):29:5
```

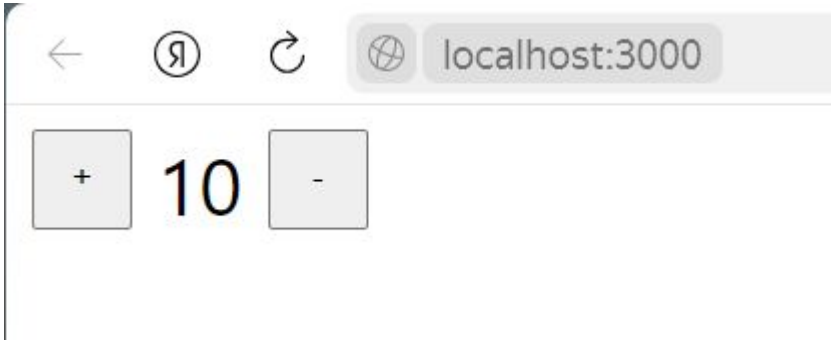
Возвращаемся к нашим компонентам

```
render() {
    return (
        <Counter
            value={this.state.value}
            onIncrement={this.handleIncrement.bind(this)}
            onDecrement={() => {
                this.handleDecrement()
            }}
        />
    );
}
```

Контролируемая и неконтролируемая компонента

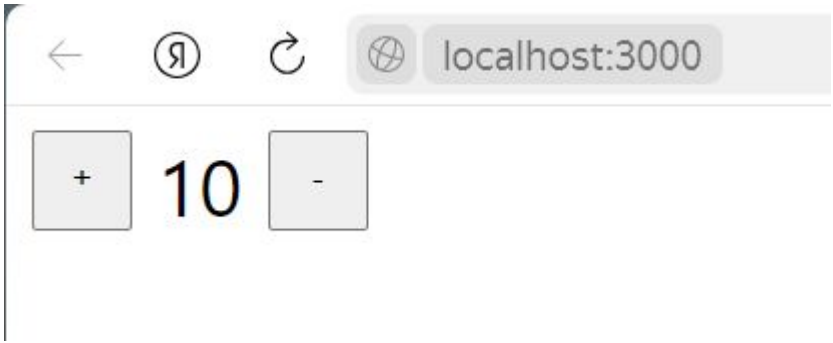
```
class Counter extends Component {
    constructor(props) {
        super(props);

        this.state = {
            value: this.props.initialValue || this.props.value || 0
        }
    }

    render() {
        const {
            value = this.state.value,
            onIncrement = this.onIncrementHandler.bind(this),
            onDecrement = this.onDecrementHandler.bind(this),
            ...props
        } = this.props;

        return (
            <div className='counter' {...props}>
                <button className='counter__button counter__button_inc' onClick={onIncrement}>
                    +
                </button>
                <span className='counter__input'>
                    {value}
                </span>
                <button className='counter__button counter__button_dec' onClick={onDecrement}>
                    -
                </button>
            </div>
        )
    }
    onIncrementHandler() {
        this.setState({ value: this.state.value + 1 });
    }
    onDecrementHandler() {
        this.setState({ value: this.state.value - 1 });
    }
}
```

```
render() {
    return <Counter initialValue={10} />;
}
```



Контролируемая и неконтролируемая компонента

```
render() {
    return (
        <Counter
            value={this.state.value}
            onIncrement={this.handleIncrement.bind(this)}
            onDecrement={this.handleDecrement.bind(this)}
        />
    );
}
```



Как не потерять изменение состояние при нескольких вызовах setState

```
    #userNameChange = this.#userNameChangeHandler.bind(this);
    #passwordChange = this.#passwordChangeHandler.bind(this);
    #submitForm = this.#submitFormHandler.bind(this);

    render() {
        return (
            <>
                <form className='login-form'>
                    <label className='login-form__username' style={{ display: 'block' }}>
                        Логин:
                        <input
                            type='text'
                            value={this.state.userName}
                            onChange={this.#userNameChange}
                        />
                    </label>
                    <label className='login-form__password' style={{ display: 'block' }}>
                        Пароль:
                        <input
                            type='password'
                            value={this.state.password}
                            onChange={this.#passwordChange}
                        />
                    </label>
                    <div
                        className='login-form__error-messages'
                        style={{ minHeight: '32px', color: 'red' }}>
                        {this.state.errorMessage}
                    </div>
                    <button
                        type='submin'
                        onClick={this.#submitForm}>
                        Войти
                    </button>
                </form>
                <button onClick={this.#handleIncrement.bind(this)}>{this.state.counter}</button>
            </>
        );
    }
```

```
    #userNameChangeHandler(event) {
        this.setState({
            userName: event.target.value
        });
    }

    #passwordChangeHandler(event) {
        this.setState({
            errorMessage: event.target.value.length < 8
                ? 'Пароль должен быть не менее 8 символов'
                : ''
        });

        this.setState({
            password: event.target.value
        });
    }

    #submitFormHandler(event) {
        event.preventDefault()
    }

    #handleIncrement() {
        this.setState({ counter: this.state.counter + 1 });
        this.setState({ counter: this.state.counter + 1 });
        this.setState({ counter: this.state.counter + 1 });
        this.setState({ counter: this.state.counter + 1 });
    }
}
```

Как не потерять изменение состояние при нескольких вызовах setState

```
#handleIncrement() {
    this.setState({ counter: this.state.counter + 1 });
    this.setState((state) => {
        return {
            counter: state.counter + 1
        };
    });
}
```

Условный рендеринг

```
4   export class Example9 extends Component {
5       constructor(props) {
6           super(props);
7           this.state = {
8               renderBlock: true
9           };
10      }
11
12      render() {
13          return (
14              <div>
15                  <button onClick={this.handleFormSwitch.bind(this)}>
16                      Переключить видимость блока
17                  </button>
18
19                  {this.state.renderBlock && <div>Первый блок</div> || <div>Второй блок</div>}
20              </div>
21          );
22      }
23
24      handleFormSwitch() {
25          this.setState({ renderBlock: !this.state.renderBlock });
26      }
27  }
```

Описание компоненты отображения со свойством children

```
7   export class Example20 extends Component {
8       constructor(props) {
9           super(props);
10          this.state = {};
11      }
12
13      render() {
14          return <StyledDiv>Какой-то стилизованный блок</StyledDiv>;
15          // return <StyledDiv>{() => { return <>Какой-то стилизованный блок</> }}</StyledDiv>;
16      }
17  }
```

Описание компоненты отображения со свойством children

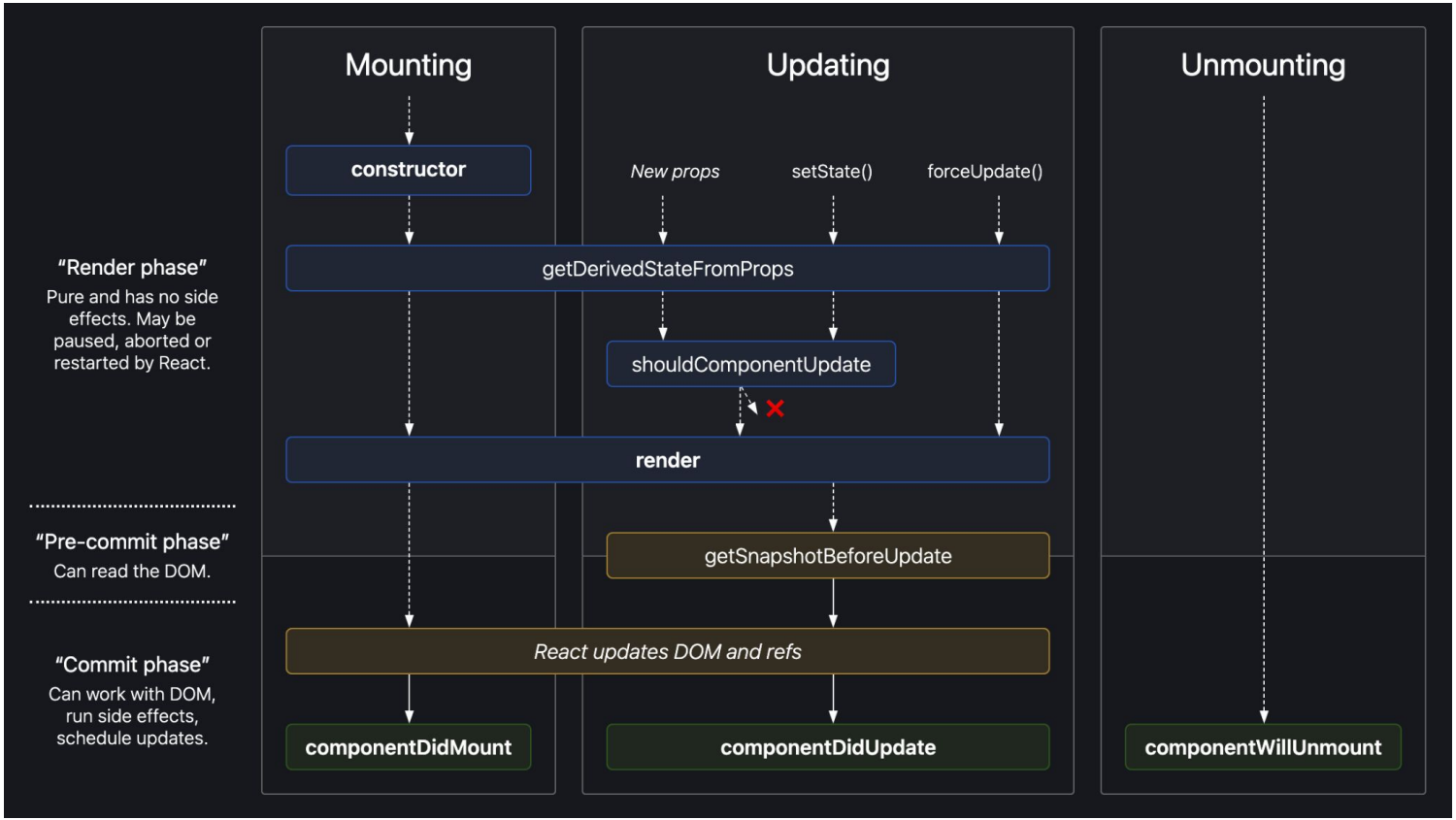
```
19  class StyledDiv extends Component {
20      render() {
21          return (
22              <div style={{
23                  padding: '20px',
24                  color: 'green',
25                  fontWeight: 'bold',
26                  fontSize: '2em',
27                  border: '1px solid red',
28                  margin: '20px'
29              }}>
30                  {
31                      typeof this.props.children === 'function'
32                          ? this.props.children()
33                          : this.props.children
34                  }
35              </div>
36          )
37      }
38  }
```

Ссылки на дочерние компоненты

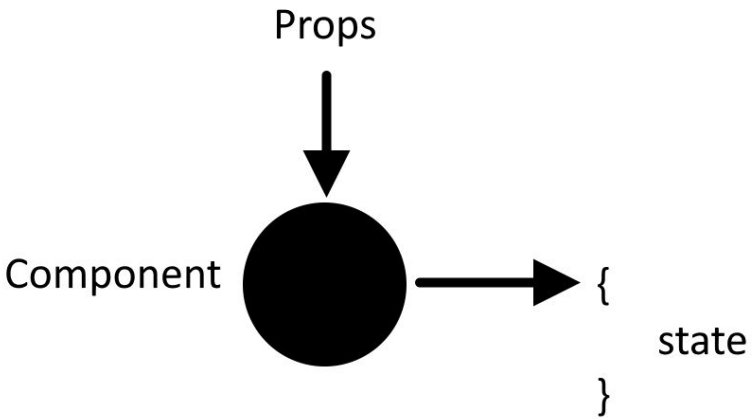
```
export class Example10 extends Component {
    constructor(props) {
        super(props);
        this.state = {
            renderInput: true
        };
    }

    render() {
        return (
            <div>
                <button onClick={this.handleFormSwitch.bind(this)}>
                    Переключить поля ввода
                </button>

                {this.state.renderInput && <FocusableInput />}
            </div>
        );
    }

    handleFormSwitch() {
        this.setState({ renderInput: !this.state.renderInput });
    }
}
```

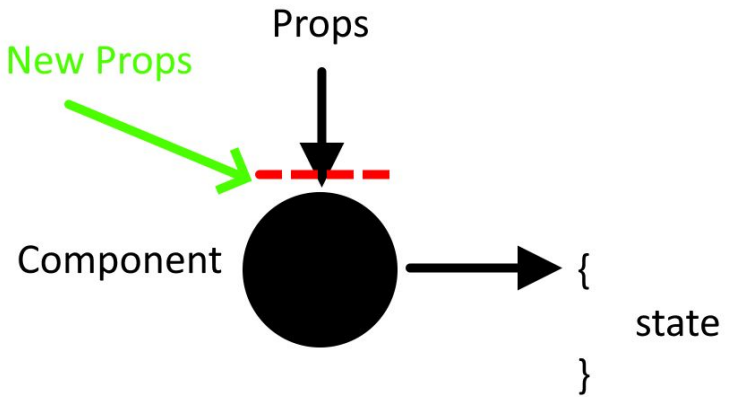
```
class FocusableInput extends Component {
    #inputRef = createRef();

    componentDidMount() {
        if (this.#inputRef.current) {
            this.#inputRef.current.focus();
        }
    }

    render() {
        return (
            <input ref={this.#inputRef} />
        );
    }
}
```

Жизненный цикл компонентов



Пару слов о getDerivedStateFromProps





Пару слов о getDerivedStateFromProps

```
static getDerivedStateFromProps(prevProps, state) {
    if (prevProps.property != state.property) {
        return {
            // Новое состояние
        }
    }


    // Игнорируем входящие пропсы
    return null;
}
```

Не рекомендуется так делать, поскольку усложняется процесс взаимодействия с компонентой

Лучше использовать либо полностью контролируемый компонент, либо не контролируемый

Пару слов о getDerivedStateFromProps. Как делать не стоит

Пример 1

```
4   export class Example11 extends Component {
5       constructor(props) {
6           super(props);
7           this.state = {
8               value: ''
9           };
10      }
11
12      render() {
13          return (
14              <>
15                  <Input value={this.state.value} />
16                  <button
17                      onClick={() => { this.setState({ value: this.state.value + 'a' }) }}>
18                      Изменить строку
19                  </button>
20              </>
21          );
22      }
23  }
```

Пару слов о getDerivedStateFromProps. Как делать не стоит

Пример 1

```
25  class Input extends Component {
26      constructor(props) {
27          super(props);
28
29          this.state = {
30              value: this.props.value
31          };
32
33      }
34
35      static getDerivedStateFromProps(nextProps, prevState) {
36          console.log('get derived state from props')
37
38          return {
39              value: nextProps.value
40          };
41      }
42
43      render() {
44          return (
45              <input
46                  value={this.state.value}
47                  onChange={(event) => { this.setState({value: event.target.value}); }}
48              />
49          );
50      }
51  }
52
```

Проблема

Проблема состоит в безусловном получении нового состояния из props, что приводит к некорректному поведению

Пару слов о getDerivedStateFromProps. Как делать не стоит

Пример 2

```
export class Example12 extends Component {
    constructor(props) {
        super(props);

        this.state = {
            userIndex: 0
        };
    }

    render() {
        const { email, userName } = users[this.state.userIndex];

        return (
            <>
                <EmailInput email={email} userName={userName} />
                <div>
                    <button onClick={() => this.setState({ userIndex: 0 })}>1</button>
                    <button onClick={() => this.setState({ userIndex: 1 })}>2</button>
                    <button onClick={() => this.setState({ userIndex: 2 })}>3</button>
                </div>
            </>
        );
    }
}
```

```
const users = [
    {
        email: 'vasya-pupkin@mail.ru',
        userName: 'Vasya Pupkin'
    },
    {
        email: 'vasya-pupkin@mail.ru',
        userName: 'Petya Petrov'
    },
    {
        email: 'zhenya-sidorov@mail.ru',
        userName: 'Zhenya Sidorov'
    }
];

```

Пару слов о getDerivedStateFromProps. Как делать не стоит

Пример 2

```
44  class EmailInput extends Component {
45      state = {
46        email: this.props.email,
47        userName: this.props.userName
48      };
49
50      static getDerivedStateFromProps(nextProps, prevState) {
51      //   if (prevState !== nextProps) {
52      //     return {
53
54      //     }
55      //   }
56
57          if (nextProps.email !== prevState.email) {
58              return {
59                  email: nextProps.email,
60                  userName: nextProps.userName
61              }
62          }
63
64          return null;
65      }
66
```

Пару слов о getDerivedStateFromProps. Более удачное применение

```jsx
export class Example13 extends Component {
    constructor(props) {
        super(props);

        this.state = {
            userIndex: 0
        };
    }

    render() {
        const {id,  email, userName } = users[this.state.userIndex];

        return (
            <>
                <EmailInput id={id} email={email} userName={userName} />
                <div>
                    <button onClick={() => this.setState({ userIndex: 0 })}>1</button>
                    <button onClick={() => this.setState({ userIndex: 1 })}>2</button>
                    <button onClick={() => this.setState({ userIndex: 2 })}>3</button>
                </div>
            </>
        );
    }
}
```

```jsx
const users = [
    {
        id: 1,
        email: 'vasya-pupkin@mail.ru',
        userName: 'Vasya Pupkin'
    },
    {
        id: 2,
        email: 'vasya-pupkin@mail.ru',
        userName: 'Petya Petrov'
    },
    {
        id: 3,
        email: 'zhenya-sidorov@mail.ru',
        userName: 'Zhenya Sidorov'
    }
];
```

Пару слов о getDerivedStateFromProps. Более удачное применение

```
47  class EmailInput extends Component {
48      state = {
49          id: this.props.id,
50          email: this.props.email,
51          userName: this.props.userName
52      };
53
54      static getDerivedStateFromProps(nextProps, prevState) {
55          if (nextProps.id !== prevState.id) {
56              return {
57                  id: nextProps.id,
58                  email: nextProps.email,
59                  userName: nextProps.userName
60              }
61          }
62
63          return null;
64      }
```

Отрисовка списков и prop key

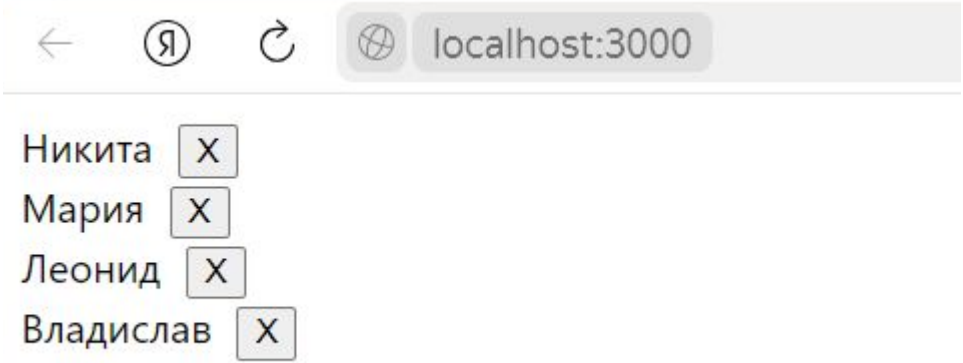
Инициализация состояния

```
constructor(props) {
    super(props);
    this.state = {
        users: [
            { id: 1, name: 'Никита'},
            { id: 2, name: 'Мария' },
            { id: 3, name: 'Леонид' },
            { id: 4, name: 'Владислав'}
        ]
    };
}
```

Отрисовка компонента и обработка удаления

```
render() {
    return (
        <div style={{ padding: '10px' }}>
            {this.state.users.map((user, index) => {
                return (
                    <div className='user-card' /* key={index} */>
                        <span>{user.name}</span>
                        <button
                            style={{ marginLeft: '10px' }}
                            onClick={this.#removeUserHandler.bind(this, index)}>X</button>
                    </div>
                );
            })}
        </div>
    );
}

#removeUserHandler(index) {
    const updatedUsersList = this.state.users.slice();
    updatedUsersList.splice(index, 1);

    this.setState({
        users: updatedUsersList
    });
}
```

Отрисовка списков и prop key



React понимает, что отрисовывает однотипные компоненты (список) и предупреждает, что для эффективного обновления DOM необходимо, чтобы элементы имели ключи

```
▶Warning: Each child in a list should have a unique "key" prop.

Check the render method of `Example15`. See https://reactjs.org/link/warning-keys for more information.
    at div
    at Example15 (http://localhost:3000/static/js/bundle.js:33:5)
```

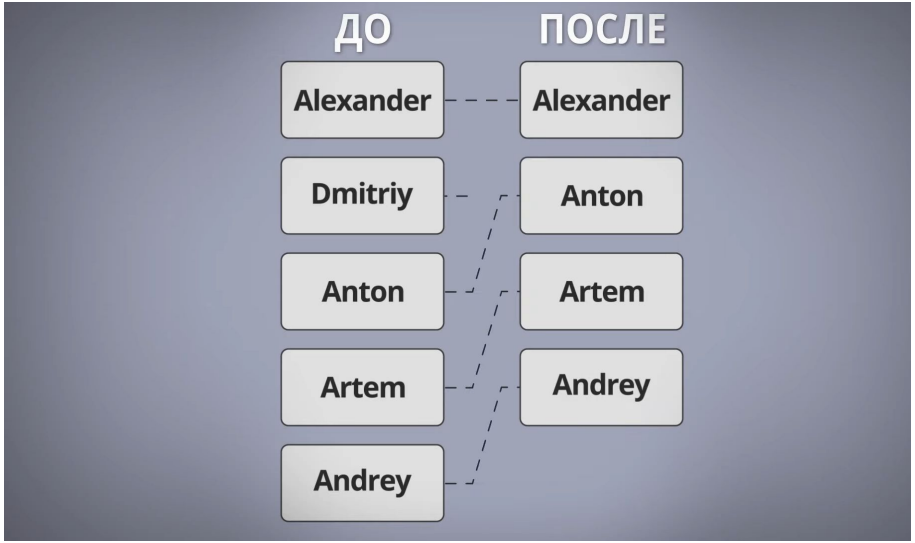
Отрисовка списков и prop key

Попробуем в качестве ключа использовать index элемента и посмотрим, что будет происходить с компонентами при удалении элемента из середины списка. В примере используются чистые компоненты.

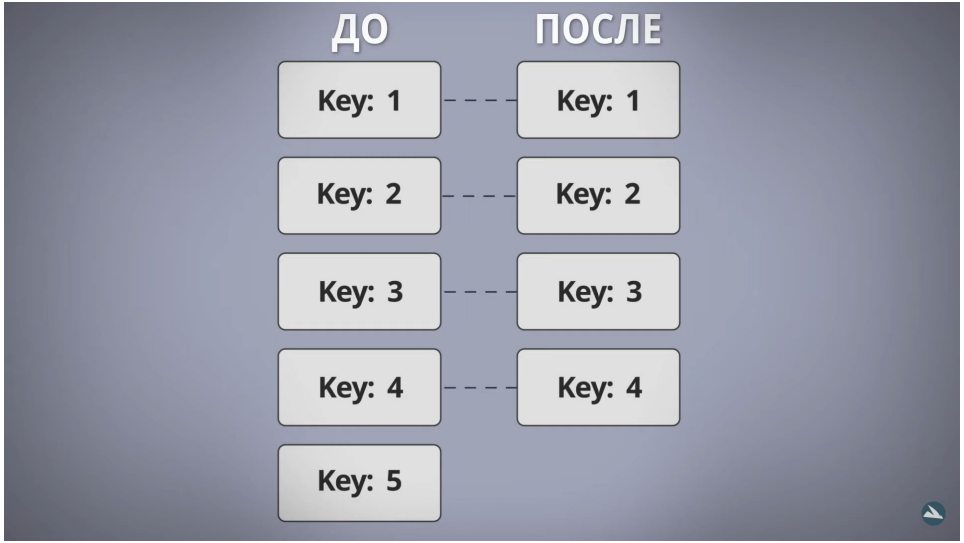
Чистый компонент — это компонент, который всегда рендерит одно и то же при одних и тех же значениях props. Это серьезно улучшает производительность.

Отрисовка списков и prop key

Пользовательское ожидание



Что происходит на самом деле



https://www.youtube.com/watch?v=OtAlPwW8DNU&t=47s

Правильная отрисовка списков

```
27
28  #handler = this.#removeUserHandler.bind(this);
29
30  render() {
31      return (
32          <div style={{ padding: '10px' }}>
33              {this.state.users.map((user, index) => {
34                  return (
35                      <UserCard
36                          id={user.id}
37                          name={user.name}
38                          onClickHandler={this.#handler}
39                          key={user.id}
40                      />
41                  );
42              })}
43          </div>
44      );
45  }
46
47  #removeUserHandler(id) {
48      const updatedUsersList = this.state.users.filter((user) => {
49          return user.id !== id;
50      });
51
52      this.setState({
53          users: updatedUsersList
54      });
55  }
```

```
return (
    <li className='user-card'>
        <span>{this.props.name}</span>
        <button
            style={{ marginLeft: '10px' }}
            onClick={() => { this.props.onClickHandler(this.props.id) }}>
                X
        </button>
    </li>
);
```

Prop key

```
22  export class Example14 extends Component {
23      constructor(props) {
24          super(props);
25
26          this.state = {
27              userIndex: 0
28          };
29      }
30
31      render() {
32          const {id,  email, userName } = users[this.state.userIndex];
33
34          return (
35              <>
36                  <label>
37                      Изменяем почту снаружи:
38                      <input
39                          onChange={(event) => {
40                              users[this.state.userIndex].email = event.target.value;
41                          }}
42                      />
43                  </label>
44                  <EmailInput key={id} email={email} userName={userName} />
45                  <div>
46                      <button onClick={() => this.setState({ userIndex: 0 })}>1</button>
47                      <button onClick={() => this.setState({ userIndex: 1 })}>2</button>
48                      <button onClick={() => this.setState({ userIndex: 2 })}>3</button>
49                  </div>
50              </>
51          );
52      }
53  }
```

Prop key используется для оптимизации процесса отрисовки узлов дерева документа. Как правило ключи используется для отрисовки списков, однако можно встретить и такое применение

Prop key . Подытожим

В качестве ключей нужно использовать уникальные идентификаторы. В том случае, если список не будет изменяться, то можно использовать индекс элемента в массиве. Во всех других случаях индексом должен быть какой-то props с уникальным значением

Если пропсы компонента при перерисовке всего списка не изменяются для чистого компонента (при условии, что ключ не изменился), то применяется оптимизация (элемент не перерисовывается). Отсюда следует, что ключом не может выступать индекс.

Если мы используем не чистые компоненты, а данные берутся только из состояния, то при неизменном ключе и измененных пропсах состояние сохронится.

Еще об обработке событий

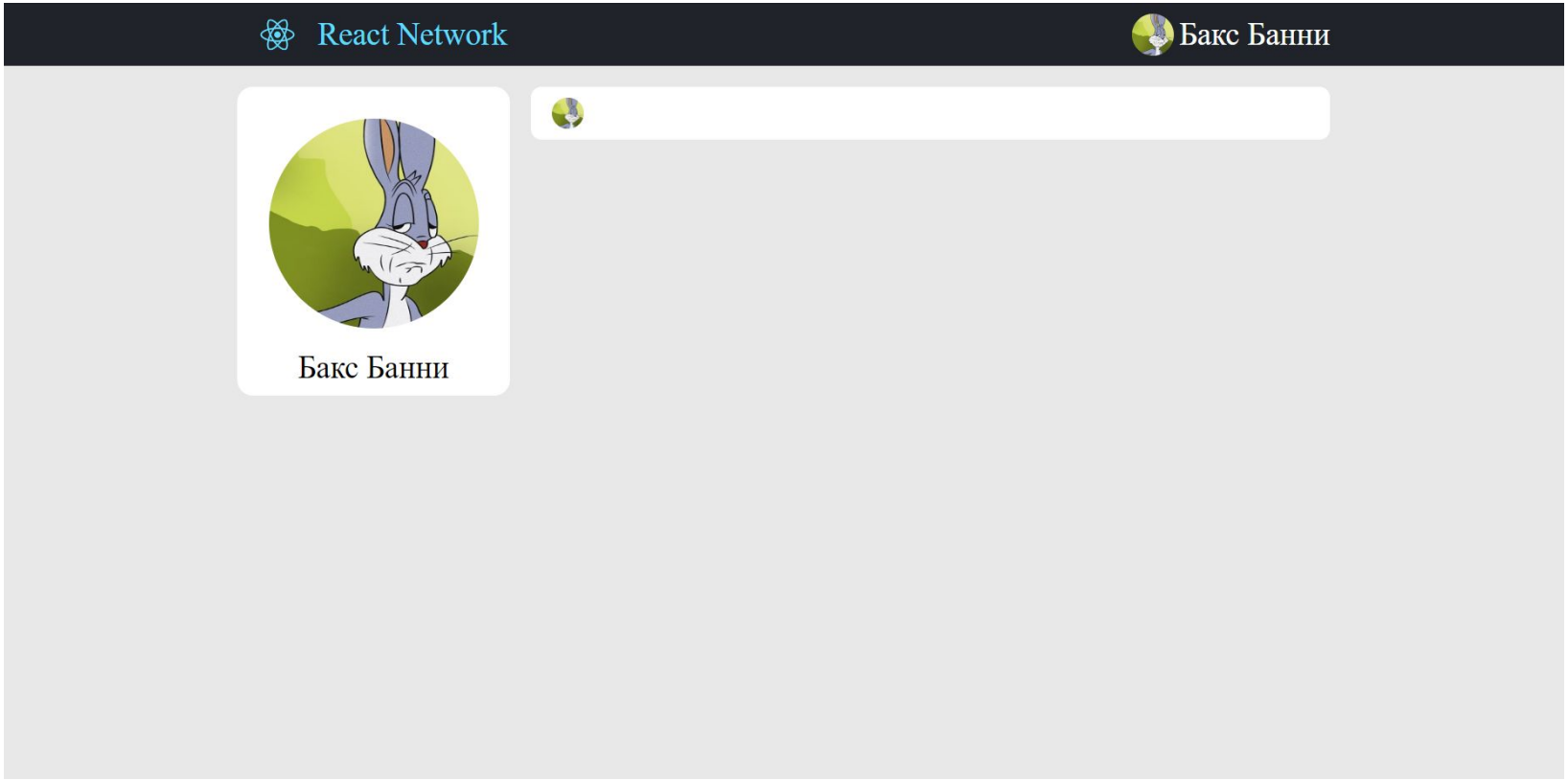
Чаще всего разработчики сталкиваются с событиями ChangeEvent для обработки пользовательского ввода и MouseEvent для обработки событий мыши.

target - ссылка на DOM-элемент на котором сработал обработчик

preventDefault() - функция предотвращающая дефолтное поведение для некоторого браузерного элемента

stopPropogation() - функция, которая прекращает дальнейшее распространение события по событийному циклу

React Context



React Context

Проблема
На верхнем уровне приложения хранится url аватарки пользователя, который необходимо передать нескольким дочерним компонентам в различных частях дерева. Для решения этой задачи можно использовать props. Однако такой подход нарушает зоны ответственности каждой компоненты.

Такое явление называется сквозной передачей данных дочерним компонентам или prop drilling.

Для решения проблемы сквозной передачи используется React-Контекст.

React Context

Создание контекста

```
4   const ThemeContext = createContext({
5       theme: 'day',
6       themeChangeHandler: undefined
7   });
```

Прокидывание данных через контекст

```
9   export class Example19 extends Component {
10      constructor(props) {
11          super(props);
12          this.state = {
13              theme: 'day'
14          };
15      }
16
17      render() {
18          return (
19              <ThemeContext.Provider
20                  value={{
21                      theme: this.state.theme,
22                      themeChangeHandler: this.#themeChangeHandler.bind(this)
23                  }}>
24                  <div
25                      style={{
26                          minHeight: '100vh',
27                          backgroundColor: this.state.theme === 'day' ? 'white' : 'black'
28                      }}>
29                      <ThemeSwitcher />
30                      <Content />
31                  </div>
32              </ThemeContext.Provider>
33          );
34      }
35
36      #themeChangeHandler() {
37          this.setState({
38              theme: this.state.theme === 'day' ? 'night' : 'day'
39          });
40      }
41  }
```

React Context

Подключение контекста через статическую переменную

```
43  class ThemeSwitcher extends Component {
44      static contextType = ThemeContext;
45
46      render() {
47          return (
48              <label>
49                  <input
50                      type='checkbox'
51                      onChange={this.context.themeChangeHandler}
52                  />
53                  <span style={{ color: this.context.theme === 'day' ? 'black' : 'white' }}>
54                      Включить темную тему
55                  </span>
56              </label>
57          );
58      }
59  }
```

React Context

Подключение контекста через Context.Consumer и renderProp

```
61  class Content extends Component {
62      render() {
63          return (
64              <ThemeContext.Consumer>
65                  {(themeContext) => {
66                      return (
67                          <div style={{ color: themeContext.theme === 'day' ? 'black' : 'white' }}>
68                              Lorem ipsum dolor sit amet consectetur adipisicing elit. Nesciunt voluptatum nihil nobis
69                              adipisci a, saepe reiciendis quod repellendus temporibus voluptates rerum voluptas perferendis
70                              minima. Repudiandae distinctio id cum obcaecati blanditiis.
71                          </div>
72                      );
73                  }}
74              </ThemeContext.Consumer>
75          );
76      }
77  }
78
```

React Context. Подытожим

Если из какого-то родительского компонента необходимо передать пропсы нескольким глубоко вложенным дочерним компонентам, то для этой задачи можно использовать контекст. Чтобы контекст работал, необходимое дочернее поддерево обернуть в Context.Provider и передать в prop value объект контекста (это могут быть данные и функции обратного вызова (callback)).

Чтобы воспользоваться данными из контекста внутри классового компонента можно проинициализировать статическую переменную contextType.

Для функциональных и классовых компонент можно воспользоваться компонентом Context.Consumer, который в качестве children prop принимает функцию, которая принимает контекст и возвращает JSX.

AJAX и Promise

AJAX (Asynchronous Javascript and XML) - это подход к построению пользовательских приложений, основанный на асинхронной отправке сообщений серверу.

Для отправки асинхронных запросов в современных версиях языка JavaScript используется функция fetch, которая первым параметром принимает некоторый url, а вторым - объект, содержащий различные HTTP-заголовки запроса.

AJAX и Promise

Список задач todos

```
this.state = {
    todos: [],
    isLoading: true,
    hasError: false
}
```

```
componentDidMount() {
    fetch('https://jsonplaceholder.typicode.com/todos')
        .then(response => response.json())
        .then(todos => {
            this.setState({
                todos: todos,
                isLoading: false
            });
        })
        .catch(() => {
            this.setState({
                isLoading: false,
                hasError: true
            });
        });
}
```

```
render() {
    let content = <div className='loader'>Загрузка...</div>;
    if (!this.state.isLoading) {
        if (this.state.hasError) {
            content = <div style={{ color: 'red' }}>Ошибка загрузки</div>;
        } else {
            content = (
                <div className='todo-list'>
                    {this.state.todos.map((todo) => {
                        return (
                            <Todo
                                id={todo.id}
                                completed={todo.completed}
                                onCompleteChange={this.#handleCompleteChange.bind(this)}>
                                {todo.title}
                            </Todo>
                        );
                    })}
                </div>
            );
        }
    }

    return (
        <>
            <h2>Список задач</h2>
            {content}
        </>
    )
}
```

AJAX и Promise

Список задач todos

```
#handleCompleteChange(id) {
    const updatedTodos = this.state.todos.map((todo) => {
        if (todo.id === id) {
            return {
                ...todo,
                completed: !todo.completed
            };
        }

        return todo;
    });

    this.setState({ todos: updatedTodos });
}
```

AJAX и Promise

Список задач todos

```
78  const Todo = ({ id, completed, onCompleteChange, children, ...props }) => {
79      return (
80          <div className='todo'>
81              <label>
82                  <input
83                      style={{ marginRight: '10px' }}
84                      type='checkbox'
85                      checked={completed}
86                      onChange={() => { onCompleteChange(id) }}
87                  />
88                  <span style={{ textDecoration: completed ? 'line-through' : 'none' }}>
89                      {children}
90                  </span>
91              </label>
92          </div>
93      );
94  }
```

React порталы

Порталы в React — это функция, которая позволяет разработчикам отображать компоненты вне текущей иерархии дерева React.

Эта возможность особенно полезна для создания компонентов, которым необходимо вырваться из своего контейнера, таких как модалы, всплывающие подсказки или любой другой элемент пользовательского интерфейса, который требует иного расположения или многоуровневости, чем его родительский компонент.

Примеры использования порталов:

— диалоги;
— модальности;
— всплывающие подсказки.

React порталы

```
class ButtonWithModal extends Component {
    #modalRoot = document.getElementById('modal')

    constructor(props) {
        super(props);
        this.state = {
            isModalWindowOpen: false
        };
    }

    render() {
        return (
            <>
                <button
                    onClick={() => {
                        this.setState({
                            isModalWindowOpen: true
                        });
                    }}>
                    Открыть модальное окно
                </button>
                {this.state.isModalWindowOpen && createPortal(
                    <ModalWindow onClose={this.#closeWindowHandler.bind(this)} />,
                    this.#modalRoot
                )}
            </>
        )
    }

    #closeWindowHandler() {
        this.setState({
            isModalWindowOpen: false
        });
    }
}
```

```
class ModalWindow extends Component {
    render() {
        return (
            <div style={{
                position: 'absolute',
                width: '300px',
                minHeight: '200px',
                padding: '10px',
                top: '100px',
                left: '50%',
                transform: 'translateX(-50%)',
                border: '1px solid black' }}>
                <h2>Модальное окно</h2>
                <div>
                    Lorem ipsum dolor sit, amet consectetur adipisicing elit.
                    Labore laboriosam nobis fugiat, voluptas aspernatur corporis et
                    adipisci facere soluta molestias dicta culpa numquam provident a natus recusandae
                    explicabo aperiam quasi!
                </div>
                <button onClick={this.props.onClose}>Закрыть</button>
            </div>
        )
    }
}
```

React-router

Настройка роутера

```
83  const router = createBrowserRouter([
84      {
85          path: '/',
86          element: <Layout />,
87          children: [
88              {
89                  path: '/',
90                  element: <HomePage />
91              },
92              {
93                  path: '/news',
94                  element: <NewsPage />
95              },
96              {
97                  path: '/blog',
98                  element: <BlogPage />
99              }
100         ]
101     }
102 ]);
```

Странички

```
59  class HomePage extends Component {
60      render() {
61          return (
62              <h1>Home page</h1>
63          )
64      }
65  }
66
67  class NewsPage extends Component {
68      render() {
69          return (
70              <h1>News page</h1>
71          )
72      }
73  }
74
75  class BlogPage extends Component {
76      render() {
77          return (
78              <h1>Blog page</h1>
79          )
80      }
81  }
```

React-router. Outlet

```
class Layout extends Component {
    render() {
        return (
            <div style={{ minHeight: '100%', display: 'flex', flexDirection: 'column' }}>
                <header
                    className='navigator'
                    style={{
                        display: 'flex',
                        alignItems: 'center',
                        justifyContent: 'center',
                        gap: '20px',
                        padding: '20px 0'
                    }}>
                    <div className='navigation__item'>
                        <Link to='/'>Home</Link>
                    </div>
                    <div className='navigation__item'>
                        <Link to='/news'>News</Link>
                    </div>
                    <div className='navigation__item'>
                        <Link to='/blog'>Blog</Link>
                    </div>
                </header>

                <main style={{ flex: '1 1 auto' }}>
                    <Outlet />
                </main>

                <footer style={{
                    backgroundColor: 'black',
                    color: 'white',
                    textAlign: 'center',
                    padding: '20px 0px'
                }}>
                    2024
                </footer>
            </div>
        );
    }
}
```

Композиция компонентов

Компоненты более высокого порядка

Функциональные компоненты

Классовые и функциональные компоненты. Резюме

Функциональные компоненты

- представляют собой функцию, которая возвращает JSX;
- работа с состоянием и жизненным циклом компонента представлена в виде функций - хуков (useState, useEffect);
- как правило используется для описания компонент, не содержащих сложную бизнес-логику.

Классовые компоненты

- представляют собой класс-наследник React.Component и переопределяет методы жизненного цикла (render, componentDidMount, componentWillUnmount, shouldComponentUpdate и другие);
- позволяют разбить сложную бизнес-логику приложения на отдельные методы;
- оверкод для маленьких компонент

Примеры из презентации

Репозиторий с примерами: https://github.com/Nodthar1107/React-Get-Started

Что дальше?

Typescript - транспилируемый в JS язык со статической типизацией

Redux - один из самых популярных state-менеджеров, который позволяет выстраивать flux-архитектуру.

mobx - более легковесный стейт-менеджер с уменьшенным количеством оверкода

Next.js - фреймворк для разработки веб-приложение с технологией SSG (server static generation). Позволяет решить проблемы с SEO-оптимизацией

MUI - библиотека с готовыми React-компонентами

Справочные материалы и видеокурсы

Для начинающих:
WebDev - https://www.youtube.com/playlist?list=PLNkWIWHIRwME_Gv2vlWAR6TfeSXylYfw4
Школа web-программирования Constcode - https://www.youtube.com/playlist?list=PL9mlH9etz6DyxCwks1tdVzlYhSxrsrjJ4
Курс по Typescrip - https://www.youtube.com/playlist?list=PLNkWIWHIRwMEm1FgiLjHqSky27x5rXvQa
React + Typescript - https://www.youtube.com/playlist?list=PLNkWIWHIRwMFQBDhZ6HfwO9NL09X3N3Gq
React Router - https://www.youtube.com/playlist?list=PLiZoB8JBsdznY1XwBcBhHL9L7S_shPGVE

Более продвинутый уровень
АйТи Синяк - https://www.youtube.com/@it-sin9k
Ulbi TV - https://www.youtube.com/@UlbiTV

Документация
Документация по React - https://react.dev/learn
Документация по Typescript - https://www.typescriptlang.org/docs/
React-router - https://reactrouter.com/en/main/start/tutorial

Спасибо за внимание!