МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт Информационных технологий и управления в технических системах

Кафедра Информационные системы

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дата поступления на кафедру | Подпись отв. за регистрацию | Подпись преподавателя |
|  |  |  |  |
|  |  |  |  |

**ОТЧЕТ**

о производственной (по получению профессиональных умений и опыта профессиональной деятельности) практике

в \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование организации)

Выполнил Волобуев Ю.С.

(Фамилия И.О. обучающегося)

\_\_\_\_\_\_\_\_\_\_\_\_ИС/б-17-2-о\_\_\_\_\_\_\_\_\_\_\_

(шифр группы)

Направление / специальность 09.03.02

\_\_\_\_\_Информационные системы и\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_технологии\_\_\_\_\_\_\_\_\_\_\_

(код, наименование)

Руководитель практики от Университета \_\_\_\_\_\_\_Старший преподаватель\_\_\_\_\_

(должность)

\_\_\_\_\_\_\_\_\_\_\_Заикина Е. Н.\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О. руководителя)

Севастополь

2020 г.

**ЦЕЛЬ РАБОТЫ**

Получить дополнительные навыки программирования в области WEB-технологии, изучить язык JavaScript и фреймворк React JS.

**ХОД РАБОТЫ**

**JavaScript** — [мультипарадигменный](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Поддерживает [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [императивный](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [функциональный](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) стили. Является реализацией стандарта [ECMAScript](https://ru.wikipedia.org/wiki/ECMAScript) (стандарт ECMA-262).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам [приложений](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). Наиболее широкое применение находит в [браузерах](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) как язык сценариев для придания [интерактивности](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [веб-страницам](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0).

Основные архитектурные черты: [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [слабая типизация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [прототипное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), функции как [объекты первого класса](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_%D0%BF%D0%B5%D1%80%D0%B2%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0).

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

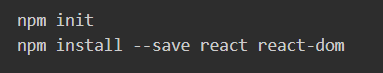
**ReactJS** — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций. React может использоваться для разработки одностраничных и мобильных приложений.

Его цель  — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, [Redux](https://ru.wikipedia.org/wiki/Redux) и GraphQL.

1. **React JS. Установка и технические требования**

### React 16 полагается на типы коллекций [Map](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Map) и [Set](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Set). Во многих старых устройствах и браузерах (например, IE < 11) эти типы коллекций отсутствуют. В других же они имеют несоответствующую реализацию (например, IE 11). Также можно включить глобальный полифил в  приложение, такой как [core-js](https://github.com/zloirock/core-js) или [babel-polyfill](https://babeljs.io/docs/usage/polyfill/).

### **Установка Laravel**

Установить React JS можно при помощи npm:  
  
 

1. **Свойства и жизненный цикл**

Рассмотрим пример тикающих часов из [одного из предыдущих разделов](https://ru.react.js.org/docs/rendering-elements.html#updating-the-rendered-element). В разделе [Отрисовка элементов](https://ru.react.js.org/docs/rendering-elements.html#rendering-an-element-into-the-dom) мы изучили только один способ обновления пользовательского интерфейса (UI). Мы вызываем ReactDOM.render() для изменения отрисованного вывода:

function tick() {

const element = (

<div>

<h1>Привет, мир!</h1>

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

</div>

);

ReactDOM.render(

element,

document.getElementById('root')

);

}

setInterval(tick, 1000);

Мы можем начать с инкапсуляции кода в функциональный компонент часов:

function Clock(props) {

return (

<div>

<h1>Привет, мир!</h1>

<h2>Сейчас {props.date.toLocaleTimeString()}.</h2>

</div>

);

}

function tick() {

ReactDOM.render(

<Clock date={new Date()} />,

document.getElementById('root')

);

}

setInterval(tick, 1000);

Тем не менее, следующий код упускает ключевое требование: то, что Clock — настраиваемый таймер, который обновляет свой интерфейс каждую секунду, должно быть деталью реализации Clock.

В идеале мы хотим написать это один раз и иметь само обновление Clock:

ReactDOM.render(

<Clock />,

document.getElementById('root')

);

Для реализации этого, нам нужно добавить «состояние» к компоненту Clock.

Состояние похоже на свойство, но оно является закрытым и полностью контролируется компонентом.

1. **Функциональные и классовые компоненты**

#### **Общедоступная директория**

Самый простой способ определить компонент — написать JavaScript-функцию:

function Welcome(props) {

return <h1>Привет, {props.name}</h1>;

}

Данная функция — корректный компонент React, потому что он принимает один аргумент-объект «props» (который обозначает свойства) с данными и возвращает элемент React. Такие компоненты мы называем «функциональными», потому что они являются буквально функциями JavaScript.

Вы также можете использовать [класс из ES6](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Classes) для определения компонента:

class Welcome extends React.Component {

render() {

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

}

}

Два вышеуказанных компонента эквивалентны с точки зрения React.

## Композиция компонентов

Компоненты могут ссылаться на другие компоненты в своём выводе. Это позволяет использовать одну и ту же абстракцию компонента для любого уровня детализации. Кнопка, форма, диалоговое окно, экран: в приложениях React все они обычно являются компонентами.

Например, мы можем создать компонент App, который многократно отрисовывает Welcome:

function Welcome(props) {

return <h1>Привет, {props.name}</h1>;

}

function App() {

return (

<div>

<Welcome name="Сара" />

<Welcome name="Кахаль" />

<Welcome name="Эдит" />

</div>

);

}

ReactDOM.render(

<App />,

document.getElementById('root')

);

Как правило, в новых приложениях React есть один компонент App, который находится в самом верху иерархии компонентов. Однако, если неоходимо интегрировать React в существующее приложение, можно начать снизу вверх с небольшого компонента, такого как Button, и постепенно идти вверх по иерархии представлений.

## Извлечение компонентов

Рассмотрим компонент Comment:

function Comment(props) {

return (

<div className="Comment">

<div className="UserInfo">

<img className="Avatar"

src={props.author.avatarUrl}

alt={props.author.name}

/>

<div className="UserInfo-name">

{props.author.name}

</div>

</div>

<div className="Comment-text">

{props.text}

</div>

<div className="Comment-date">

{formatDate(props.date)}

</div>

</div>

);

}

Он принимает author (объект), text (строка) и date (дата) в качестве свойств и описывает комментарий на сайте социальных сетей.

Этот компонент может быть сложно изменить из-за вложенности, а также трудно повторно использовать отдельные его части. Извлечём из него несколько компонентов.

Сначала мы извлечём Avatar:

function Avatar(props) {

return (

<img className="Avatar"

src={props.user.avatarUrl}

alt={props.user.name}

/>

);

}

Компонент Avatar не должен знать, что он отрисовывается внутри Comment. Вот почему мы присвоили свойству объекта props более общее имя: user, а не author.

Теперь мы можем чуть-чуть упростить Comment:

function Comment(props) {

return (

<div className="Comment">

<div className="UserInfo">

<Avatar user={props.author} />

<div className="UserInfo-name">

{props.author.name}

</div>

</div>

<div className="Comment-text">

{props.text}

</div>

<div className="Comment-date">

{formatDate(props.date)}

</div>

</div>

);

}

Затем мы извлечём компонент UserInfo, который отобразит Avatar рядом с именем пользователя:

function UserInfo(props) {

return (

<div className="UserInfo">

<Avatar user={props.user} />

<div className="UserInfo-name">

{props.user.name}

</div>

</div>

);

}

Это позволяет нам упростить Comment ещё больше:

function Comment(props) {

return (

<div className="Comment">

<UserInfo user={props.author} />

<div className="Comment-text">

{props.text}

</div>

<div className="Comment-date">

{formatDate(props.date)}

</div>

</div>

);

}

Если часть пользовательского интерфейса используется несколько раз (Button,Panel, Avatar) или достаточно сложна сама по себе (App, FeedStory,Comment), то это хороший кандидат на извлечение компонента, чтобы он стал повторно используемым компонентом.

# **Обработка событий**

Обработка событий React-элементов очень похожа на обработку событий на элементах DOM. Существуют синтаксические различия:

* События React именуются в стиле camelCase, а не в нижнем регистре.
* С JSX вы передаёте функцию как обработчик события вместо строки.

Например, HTML:

<button onclick="activateLasers()">

Активировать лазеры

</button>

немного отличается по сравнению с React:

<button onClick={activateLasers}>

Активировать лазеры

</button>

Другое отличие состоит в том, что нельзя вернуть false для предотвращения поведения по умолчанию в React. Вы должны явно вызывать preventDefault. Например, с помощью обычного HTML, чтобы предотвратить поведение ссылки по умолчанию, которое состоит в открытии новой страницы, можно написать:

<a href="#" onclick="console.log('Была нажата ссылка.'); return false">

Нажми на меня

</a>

В React это могло бы быть так:

function ActionLink() {

function handleClick(e) {

e.preventDefault();

console.log('Была нажата ссылка.');

}

return (

<a href="#" onClick={handleClick}>

Нажми на меня

</a>

);

}

Здесь e — синтетическое событие. React определяет эти синтетические события в соответствии со [спецификацией W3C](https://www.w3.org/TR/DOM-Level-3-Events/). При использовании React вам обычно не нужно вызывать addEventListener, чтобы добавить обработчиков событий в элемент DOM после его создания. Вместо этого используется обработчик, когда элемент изначально отрисовывается.

Когда определяется компонент, используя [класс ES6](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Classes), распространённой практикой является объявлять обработчик события как метод класса. Например, данный компонент Toggle отображает кнопку, которая позволяет пользователю переключаться между состояниями «ON» и «OFF»:

class Toggle extends React.Component {

constructor(props) {

super(props);

this.state = {isToggleOn: true};

// Это привязывание необходимо, чтобы работал объект `this` в колбэке

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

this.setState(state => ({

isToggleOn: !state.isToggleOn

}));

}

render() {

return (

<button onClick={this.handleClick}>

{this.state.isToggleOn ? 'ON' : 'OFF'}

</button>

);

}

}

ReactDOM.render(

<Toggle />,

document.getElementById('root')

);

Нужно быть аккуратным со значением this в колбэках JSX. В JavaScript методы класса не [привязаны](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_objects/Function/bind) по умолчанию. Можно использовать поля классов для правильной привязки колбэков:

class LoggingButton extends React.Component {

handleClick = () => {

console.log('this это:', this);

}

render() {

return (

<button onClick={this.handleClick}>

Нажми на меня

</button>

);

}

}

Этот синтаксис включён по умолчанию в [Create React App](https://github.com/facebookincubator/create-react-app).

Вместо него можно воспользоваться [стрелочной функцией](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Functions/Arrow_functions) в колбэке:

class LoggingButton extends React.Component {

handleClick() {

console.log('this это:', this);

}

render() {

// Данный синтаксис гарантирует, что `this` привязан внутри handleClick.

return (

<button onClick={(e) => this.handleClick(e)}>

Нажми на меня

</button>

);

}

}

Проблема с этим синтаксисом заключается в том, что при каждой отрисовке LoggingButton создаётся новый колбэк. В большинстве случаев это не создае проблем. Однако, если этот колбэк передаётся в качестве свойства нижестоящим компонентам, то эти компоненты могут выполнять дополнительную повторную отрисовку. Как правило, мы рекомендуем привязывать обработчики в конструкторе или использовать синтаксис полей классов, чтобы избежать такого рода проблем с производительностью.

## Передача аргументов в обработчики событий

Внутри цикла обычно требуется передать дополнительный параметр обработчику событий. Например, если id — это идентификатор строки, рабочим будет один из следующих вариантов:

<button onClick={(e) => this.deleteRow(id, e)}>Удалить строку</button>

<button onClick={this.deleteRow.bind(this, id)}>Удалить строку</button>

Вышеуказанные две строки эквикаленты и используют [стрелочные функции](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Functions/Arrow_functions) и [Function.prototype.bind](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_objects/Function/bind) соответственно.

В обоих случаях аргумент e, представляющий событие React, будет передан как второй аргумент после ID. При использовании стрелочной функции нужно передавать её явно, но с помощью bind любые дополнительные аргументы передаются в функцию автоматически.

**ЗАКЛЮЧЕНИЕ**

В ходе данного исследования были выявлены преимущества и недостатки React JS, а также его особенности. Были получены базовые навыки для использования React JS и внедрения его в проекты.