|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ\_\_\_\_\_\_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА\_\_\_\_\_\_\_\_\_\_\_\_ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***“Система контроля версий???.”***

Студент \_\_ИУ10-24\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**Л. К. Ревякин (ну и лох)**\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Е.А.Ефремов**\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2021 г.*

# **Теоретическая часть.**

## **React.**

### **Актуальность программирования на «React.js».**

React.js - JavaScript библиотека, позволяющая создавать пользовательские интерфейсы. Приложение, написанное с помощью библиотеки «React.js», собирается из составных частей – компонентов. Компоненты позволяют разбить приложение на независимые смысловые части, которые в последствии можно повторно использовать.

Такой подход упрощает разработку, так как легче сначала создать маленькие независимые части приложения, а потом собрать их в готовый проект, чем сразу же разработать приложение целиком.

К достоинствам React.js можно отнести:

* Лёгок в изучении. Считается, что научиться программировать, используя React.js, легче чем научиться программировать с использованием аналогичных фреймворков (Angular.js, Vue.js).
* Легок для восприятия. В основе React.js лежит расширение языка JavaScript – JSX. JSX позволяет комбинировать в себе HTML и JavaScript код, что улучшает читабельность кода в целом.
* Однонаправленная передача данных. Это означает, что дочерний элемент не может напрямую изменять состояние родителя. Дочерний элемент может только вызвать изменения через call-back функцию.
* Virtual DOM. То есть React.js сравнивает DOM-дерево с его виртуальной копией, определяет их отличия и перерисовывает только то, что изменилось. Такой подход позволяет React.js работать значительно быстрее.
* Сообщество поклонников React.js активно развивается, поэтому искать ответы, на возникающие при разработке вопросы, становиться всё легче и легче.

Совокупность всех этих факторов делает React.js одним из самых популярных и востребованных на рынке труда инструментов для разработки web-приложений.

Согласно статье [Библиотека React: особенности, перспективы, ситуация на рынке... (hexlet.io)](https://ru.hexlet.io/blog/posts/biblioteka-react-review-article):

«[По информации сооснователя npm Inc Лорри Восса (Laurie Voss)](https://medium.com/npm-inc/this-year-in-javascript-2018-in-review-and-npms-predictions-for-2019-3a3d7e5298ef), доля React на рынке фронтенд-библиотек и фреймворков в конце 2018 года достигла 60 %. Данные о популярности библиотек и фреймворков получены в ходе опроса пользователей системы управления пакетами npm.

По данным исследования The State of JS 2019, в котором приняло участие более 20 000 разработчиков, React [лидирует в категории «Фронтенд-фреймворки и библиотеки»](https://2019.stateofjs.com/front-end-frameworks/). 71 % респондентов сказали, что использовали эту библиотеку и будут использовать её в будущем. С 2016 года доля разработчиков, которые используют React и планируют использовать его в будущем, [выросла с 48 до 71 %](https://2019.stateofjs.com/front-end-frameworks/react/).»

Так что можно сказать, что React.js уже сейчас занимает высокие позиции по популярности среди прочих инструментов для разработки и, скорее всего, продолжит наращивать свои позиции в будущем.

### **Программирование с использованием React.js.**

Так как результатом нашей курсовой работы является приложение, написанное на React.js, и библиотека с готовыми компонентами, созданными для разработчиков, использующих React.js, нам необходимо усвоить основные принципы написания web-приложения на React.js.

Как и у любого другого инструмента для разработки web-приложений, React.js имеет свои особенности. Давайте разберёмся в основных принципах при работе с React.js.

Одним из пунктов в преимуществах библиотеки React.js мы указали использование расширения языка JavaScript – JSX. Это расширение позволяет в самом компоненте совмещать разметку HTML и JavaScript код.

Пример:

Листинг 1 – пример JSX кода

const element = <h1>Привет, мир!</h1>;

Если JSX-тег написан с большой буквы, то это означает, что он ссылается на React-элемент.

Пример:

Листинг 2 – пример вызова React-элемента.

const element = <Welcome name="Алиса" />;

Как отмечалось ранее, основная концепция при разработке приложения на React.js заключается в разделении всего проекта на маленькие независимые части – компоненты. Компоненты во многом схожи с функциями в JavaScript: они принимают от своего родителя некоторые входные данные (props) и возвращают компонент, который мы хотим увидеть в нашем приложении.

В React.js существует два способа создать компонент:

1. Функциональный – компонент является функцией, которая принимает параметры (props) и возвращает React-элемент.

Пример:

Листинг 3 – пример функционального компонента.

function Welcome(props) {

return <h1>Привет, {props.name}</h1>;

}

1. Классовый – компонент представляет из себя класс ES6, у которого есть обязательный метод render, который возвращает React-элемент.

Пример:

Листинг 4 – пример классового компонента.

class Welcome extends React.Component {

render() {

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

}

}

Примечание: в практической части мы почти всегда использовали классовые компоненты.

Часто бывает такое, что в нашем компоненте со временем обновляются данные. Для решение данных задач в React.js реализованы состояния (state). Объект state нужен, чтобы описать внутренние состояния компонента. Главное отличие state от props заключается в том, что props передаются от родителя потомку, и в самом компоненте мы можем только читать props. State управляется самим компонентом, то есть внутри компонента мы не можем изменить props, но можем изменять state. Изменять state мы можем с помощью асинхронного метода setState(). При изменении данных внутри компонента, он рендериться заново с новыми данными.

Пример:

Листинг 5 – пример вызова метода setState().

this. setState({comment: 'Привет'});

Таким образом метод setState нужен для обновления интерфейса при изменении внутренних данных самого компонента.

В ходе работы с библиотекой React.js нередко возникают случаи, когда необходимо выполнить некоторые действия в определённый момент существования компонента. Для того чтобы работать с компонентом на разных этапах его жизненного цикла, в React.js реализованы методы жизненного цикла. Весь жизненный цикл компонента можно разделить на три этапа:

1. Монтирование (первоначальный рендеринг компонента в DOM).
2. Обновление ().
3. Размонтирование (компонент удаляется).

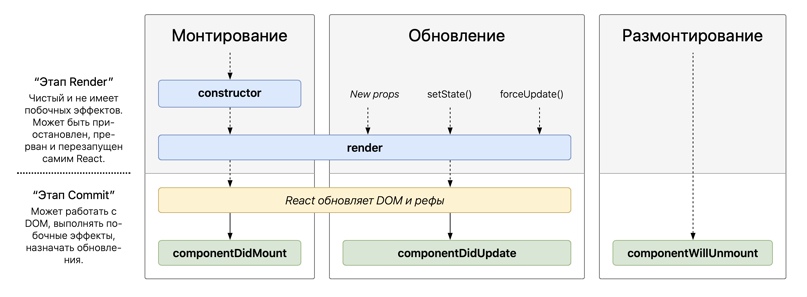
У каждого этапа есть свои методы. Мы разберём только основные методы жизненного цикла, так как их функционала хватит, чтобы решить большинство подобных задач.

Рис.1 Методы жизненного цикла

* constructor(props): конструктор компонента вызывается до того, как компонент будет смонтирован. Поэтому конструкторы в React.js обычно используют для инициализации внутреннего состояния компонента. Чтобы получить доступ к this.props внутри конструктора необходимо вызвать super(props) у класса-предка React.Component.
* render(): мы уже говорили, что этот метод является обязательным в классовом компоненте. Обычно метод render() возвращает React-элемент. Метод render() должен быть чистым. То есть он не изменяет состояние компонента, всегда возвращает один и тот же результат.
* componentDidMount(): этот метод вызывается только один раз сразу же после монтирования компонента. На этом этапе удобно создавать сетевые запросы.
* componentDidUpdate(): этот метод вызывается, если компонент обновился, при этом он не срабатывает при первом рендере. С помощью этого метода можно проверять совпали ли предыдущие props с нынешними, и в зависимости от результаты проверки совершать определённые действия (например, можно создать сетевые запросы). В этом методе можно использовать метод setState(), однако нужно его нужно обернуть в условие, иначе метод componentDidUpdate может зациклиться.
* componentWillUnmount(): этот метод вызывается перед размонтированием компонента. В этом методе обычно отменяются действия, которые были совершенны в методе componentDidMount() (сброс таймеров, отмена сетевых запросов). В этом методе уже нельзя использовать метод setState(), так как компонент больше не будет рендериться.

Существуют и другие методы жизненного цикла, но они используются крайне редко.

При создании приложения часто возникает потребность во взаимодействии с пользователем. Например, мы хотим изменить интерфейс по нажатию на кнопку или поменять текст при наведении на элемент. Для того чтобы реагировать на действия пользователя, в React существуют обработчики событий. Обработка событий в React-элементах почти ничем не отличается от обработки событий в DOM-элементах, но есть несколько нюансов:

* События в React именуются в стиле camelCase (это характерно для React в целом).
* В событие мы передаём функцию (но не вызываем её), а не строку.

Пример в HTML:

Листинг 6 – пример обработки события в HTML.

<button onclick="activateLasers()">

Активировать лазеры

</button>

Пример в React.js:

Листинг 7 – пример обработки события в React.

<button onclick={activateLasers}>

Активировать лазеры

</button>

Выше мы описали основные принципы, которые необходимо понимать при работе с React. Мы перечислили только самые фундаментальные моменты, которые мы сами использовали чаще всего при разработке нашего приложения. Конечно, возможности React гораздо шире, но здесь мы описали принципы, необходимые для создания самого простого проекта на React. При создании более сложного проекта, потребуется более глубокое изучение возможностей этой библиотеки.

### **Что необходимо знать, чтобы написать систему контроля версий компонентов на React?**

Очевидно, что для начала нам нужно знать хотя бы базовые принципы программирования на React, которые мы описали в пункте 1.1.2. Но этого мало. Во-первых, нам необходимо ознакомиться с Node.js, о котором мы поговорим в следующей главе. При разработке нашего приложения мы использовали Node.js для установки необходимых библиотек (React, Prism), также мы использовали встроенный модуль «fs». Кроме этого, при разработке нашего приложения мы активно использовали систему управления версиями GIT. Наше приложение нельзя было бы назвать законченным, если бы в нем отсутствовало оформления. Для стилизации нашего приложения мы использовали каскадные таблицы стилей CSS и препроцессор SASS. Это основные технологии, которые мы задействовали при разработке нашего проекта.

### **Node.js.**

Node.js – среда выполнения JavaScript кода, основанная на движке V8. JavaScript движок – это программа, которая исполняет JavaScript код. Движок V8 – JavaScript движок, написанный на языке программирования C++, созданный компанией «Google». Таким образом, Node.js интерпретирует ваш JavaScript код в более быстрый машинный код. Это позволяет компилировать JavaScript код отдельно от вашего браузера.

Node.js позволяет превратить JavaScript из языка программирования для разработки интерактивных web-сайтов в скриптовый язык программирования общего назначения (такой как Python).

В нашем проекте мы использовали Node.js как среду выполнения JavaScript кода, чтобы скомпилировать наш скрипт treeBuilder отдельно от браузера. Мы написали программу, которая обрабатывает определённые данные и формирует некоторый вывод во внутренний файл, то есть она никак не взаимодействует с браузером. Если бы мы не использовали Node.js, то нам пришлось бы написать этот же код на другом языке программирования (например Python).

Почти ни одна из функций в Node.js не работает напрямую с вводом/выводом, поэтому поток никогда не блокируется. Это означает, что большинство функций Node.js выполняются асинхронно, то есть основной блок программы может выполнятся, независимо от того, закончила ли своё выполнение асинхронная функция. Это называется неблокирующая модель ввода/вывода.

Например, на вашей странице вы хотите отобразить фото товара, которое берётся из базы данных, и выполнив определённую функцию получить определённый фон для текста, который тоже берётся из базы данных. Вы можете делать все действия по очереди и так формировать вашу страничку:

1. Отправить запрос к базе данных, чтобы получить фото товара.
2. Дождаться ответа от базы данных.
3. Сделать определённые действия с картинкой.
4. Выполнить функцию, определяющей фон для текста.
5. Отправить запрос к базе данных, чтобы получить фон для текста.
6. Дождаться ответа от базы данных и загрузить страничку.

Node.js позволяет выполнять эти действия параллельно:

1. Отправить запрос к базе данных, чтобы получить фото товара.
2. Выполнить функцию, определяющую фон для текста.
3. Отправить запрос к базе данных, чтобы получить фон для текста.
4. На этом этапе к нам уже придёт фото товара, и мы сможем сделать с фото необходимые действия.
5. Дождаться ответа от базы данных и загрузить страничку.

Хоть по списку действий не кажется, что мы где-то выигрываем, на самом деле, мы ускоряем формирование нашей страницы в разы. Ведь в первом случае два раза просто ждём ответа от базы данных и параллельно ничего не делаем. Во втором случае мы, не дожидаясь ответа от базы данных, выполняем необходимые действия. Таким образом, наша страница формируется динамически, на что уходит значительно меньше времени.

Ещё одним достоинством Node.js является пакетная экосистема npm (Node Package Manager). Этот менеджер пакетов является самой большой экосистемой библиотек с открытым исходным кодом. Npm содержит в себе библиотеки, составленные другими разработчиками. Используя труды других разработчиков, вы можете облегчить разработку своего проекта, а также сможете не задумываться о сложных задачах, которые уже решили другие пользователи.

Например, в нашем приложении нам нужно было сделать вывод определённого кода в качестве инструкции. Чтобы этот код лучше усваивался пользователем, мы решили сделать подсветку кода. Конечно, мы могли бы попытаться написать собственную программу, которая бы осуществляла подсветку кода, но это было бы труднее чем написать всю остальную практическую часть. Поэтому мы просто подключили готовую библиотеку «Prism», которая справляется с поставленной задачей.

Node.js устанавливается сразу же с некоторыми встроенными модулями, которые могут решить множество возникающих задач. Мы приведём только основные модули:

* fs: модуль для работы с файлами и каталогами.
* path: модуль для работы с путями к файлам и каталогам.
* http: для работы с сервером и протоколом http.
* url: утилиты для разбора URL и извлечения из него элементов.

Так как в нашем проекте мы используем встроенный модуль «fs», разберём его основные функции.

* fs.readFile(path[,option], callback) – асинхронная функция, которая позволяет прочитать содержимое файла. В параметрах передаются путь к файлу, кодировка к файлу (указывается опционально) и функция, которая вызывается после завершения чтения.
* fs.writeFile(file, data[,option], callback) – асинхронная функция, которая позволяет записывать файл. При этом функция полностью перезаписывает файл. Если файл нужно дозаписать, то мы можем использовать методы fs.appendFile() . В параметрах передаются название файла, данные, которые мы хотим записать в файл, кодировка к файлу (указывается опционально) и функция, которая вызывается после завершения записи.
* fs.unlink(path, callback) – эта асинхронная функция удаляет файл. В параметрах ей передаётся путь к файлу, который необходимо удалить, и функция, которая вызывается после удаления файла.

У всех приведённых выше функций есть свои синхронные аналоги. Их отличие заключается только в том, что дальнейший код не будет исполняться, пока не выполняться эти функции.

Здесь мы рассмотрели только некоторые функции модуля «fs». На официальном сайте Node.js можно найти полный список всех функций этого модуля с подробным описанием.