Лабораторная работа №6

**Тема**: «Введение в React»

**Цель**: изучить основные принципы работы с React, научится создавать приложение на базе React.

**Теоретический материал**:

Create React App — удобная среда для изучения React и лучший способ начать создание нового приложения на React.

Инструмент настраивает среду для использования новейших возможностей JavaScript, оптимизирует приложение для продакшена и обеспечивает комфорт во время разработки. Вам понадобятся Node.js не ниже версии 14.0.0 и npm не ниже версии 5.6 на вашем компьютере. Для создания проекта выполните команды:

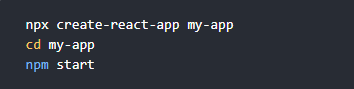


Рисунок 1 – Создание приложение командой create-react-app

Npx – это инструмент запуска пакетов, доступный в версиях npm 5.2 и выше.

Create React App не обрабатывает бэкенд логику или базы данных, он только предоставляет команды для сборки фронтенда, поэтому вы можете использовать его с любым бэкэндом. «Под капотом» используются Babel и webpack, но вам не нужно ничего знать о них.

Когда ваше приложение готово к развёртыванию в продакшене, запуск команды npm run build создаст оптимизированную сборку вашего приложения в папке build.

**Знакомство с JSX:**

Рассмотрим объявление переменной:

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

Этот странный тег — ни строка, ни фрагмент HTML.

Это JSX — расширение языка JavaScript. Мы рекомендуем использовать его, когда требуется объяснить React, как должен выглядеть UI. JSX напоминает язык шаблонов, наделённый силой JavaScript. JSX производит «элементы» React. То, как элементы рендерятся в DOM.

**Что такое JSX?**

React исходит из принципа, что логика рендеринга неразрывно связана с прочей логикой UI: с тем, как обрабатываются события, как состояние изменяется во времени и как данные готовятся к отображению.

Вместо того, чтобы искусственно разделить технологии, помещая разметку и логику в разные файлы, React разделяет ответственность с помощью слабо связанных единиц, называемых «компоненты», которые содержат и разметку, и логику.

React можно использовать и без JSX, но большинство людей ценит его за наглядность при работе с UI, живущем в JavaScript-коде. Помимо этого, JSX помогает React делать сообщения об ошибках и предупреждениях понятнее.

**Встраивание выражений в JSX:**

В следующем примере объявляем переменную name и затем используем её внутри JSX, обрамляя фигурными скобками:

const name = 'Иван-Царевич';

const element = <h1>Здравствуй, {name}!</h1>;

ReactDOM.render(

element,

document.getElementById('root')

);

**Рендеринг элемента в DOM**

Допустим, в вашем HTML-файле есть <div>:

<div id="root"></div>

Мы назовём его «корневым» узлом DOM, так как React DOM будет управлять его содержимым

Обычно в приложениях, написанных полностью на React, есть только один корневой элемент. При встраивании React в существующее приложение вы можете рендерить во столько независимых корневых элементов, во сколько посчитаете нужным.

Для рендеринга React-элемента в корневой узел DOM вызовите ReactDOM.render() с React-элементом и корневым DOM-узлом в качестве аргументов:

const element = <h1>Hello, world</h1>;

ReactDOM.render(element, document.getElementById('root'));

**Обновление элементов на странице**

Элементы React иммутабельны. После создания элемента нельзя изменить его потомков или атрибуты. Элемент похож на кадр в фильме: он отражает состояние интерфейса в конкретный момент времени.

Пока что, мы знаем только один способ обновить интерфейс — это создать новый элемент и передать его в ReactDOM.render().

Рассмотрим пример с часами:

function tick() {

const element = (

<div>

<h1>Hello, world!</h1>

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

</div>

);

ReactDOM.render(element, document.getElementById('root'));

}

setInterval(tick, 1000);

**Функциональные и классовые компоненты**

Проще всего объявить React-компонент как функцию:

function Welcome(props) {

return <h1>Привет, {props.name}</h1>;

}

Эта функция — компонент, потому что она получает данные в одном объекте («пропсы») в качестве параметра и возвращает React-элемент. Мы будем называть такие компоненты «функциональными», так как они буквально являются функциями.

Ещё компоненты можно определять как классы ES6:

class Welcome extends React.Component {

render() {

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

}

}

С точки зрения React, эти два компонента эквивалентны.

**Как отрендерить компонент**

Пока что мы только встречали React-элементы, представляющие собой DOM-теги:

const element = <div />;

Но элементы могут описывать и наши собственные компоненты:

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

Когда React встречает подобный элемент, он собирает все JSX-атрибуты и дочерние элементы в один объект и передаёт их нашему компоненту. Этот объект называется «пропсы» (props).

Например, этот компонент выведет «Привет, Алиса» на страницу:

function Welcome(props) {

return <h1>Привет, {props.name}</h1>;

}

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

ReactDOM.render(

element,

document.getElementById('root')

);

**Краткий обзор хуков**

Хуки — это функции, с помощью которых вы можете «подцепиться» к состоянию и методам жизненного цикла React из функциональных компонентов. Хуки не работают внутри классов — они дают вам возможность использовать React без классов. (Мы не рекомендуем сразу же переписывать существующие компоненты, но при желании, вы можете начать использовать хуки в своих новых компонентах.)

React содержит несколько встроенных хуков, таких как useState. Вы также можете создавать собственные хуки, чтобы повторно использовать их в других своих компонентах.

**Хук состояния:**

import React, { useState } from 'react';

function Example() {

// Объявляем новую переменную состояния "count"

const [count, setCount] = useState(0);

return (

<div>

<p>Вы нажали {count} раз</p>

<button onClick={() => setCount(count + 1)}>

Нажми на меня

</button>

</div>

);

}

В этом примере, useState — это хук (определение хука дано ниже). Мы вызываем его, чтобы наделить наш функциональный компонент внутренним состоянием. React будет хранить это состояние между рендерами. Вызов useState возвращает массив с двумя элементами, который содержит: текущее значение состояния и функцию для его обновления. Эту функцию можно использовать где угодно, например, в обработчике событий. Она схожа с this.setState в классах, но не сливает новое и старое состояние вместе. Сравнение хука useState и this.state приводится на странице Использование хука состояния.

Единственный аргумент useState — это начальное состояние. В примере выше — это 0, так как наш счётчик начинается с нуля. Заметьте, что в отличие от this.state, в нашем случае состояние может, но не обязано, быть объектом. Исходное значение аргумента используется только при первом рендере.

**Хук эффекта**

Вам скорее всего доводилось ранее запрашивать данные, делать подписки или вручную менять DOM из React-компонента. Мы расцениваем эти операции как «побочные эффекты» (или сокращённо «эффекты»), так как они могут влиять на работу других компонентов и их нельзя выполнить во время рендера.

С помощью хука эффекта useEffect вы можете выполнять побочные эффекты из функционального компонента. Он выполняет ту же роль, что и componentDidMount, componentDidUpdate и componentWillUnmount в React-классах, объединив их в единый API. Вы можете найти сравнение useEffect и этих методов на странице использование хука эффекта.

К примеру, этот компонент устанавливает заголовок документа после того, как React обновляет DOM:

import React, { useState, useEffect } from 'react';

function Example() {

const [count, setCount] = useState(0);

// По принципу componentDidMount и componentDidUpdate:

useEffect(() => {

// Обновляем заголовок документа, используя API браузера

document.title = `Вы нажали ${count} раз`;

});

return (

<div>

<p>Вы нажали {count} раз</p>

<button onClick={() => setCount(count + 1)}>

Нажми на меня

</button>

</div>

);

}

Когда вы вызываете useEffect, React получает указание запустить вашу функцию с «эффектом» после того, как он отправил изменения в DOM. Поскольку эффекты объявляются внутри компонента, у них есть доступ к его пропсам и состоянию. По умолчанию, React запускает эффекты после каждого рендера, включая первый рендер. Мы рассмотрим более подробно, как это отличается от классовых методов жизненного цикла на странице использование хука эффекта.

При необходимости вы можете вернуть из эффекта функцию, которая указывает эффекту, как выполнить за собой «сброс». Например, этот компонент использует эффект, чтобы подписаться на статус друга в сети, и выполняет сброс, отписываясь от него.

import React, { useState, useEffect } from 'react';

function FriendStatus(props) {

const [isOnline, setIsOnline] = useState(null);

function handleStatusChange(status) {

setIsOnline(status.isOnline);

}

useEffect(() => {

ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);

return () => {

ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);

};

});

if (isOnline === null) {

return 'Загрузка...';

}

return isOnline ? 'В сети' : 'Не в сети';

}

В этом примере, React будет отписываться от нашего ChatAPI перед тем, как компонент размонтируется и перед тем, как перезапустить эффект при повторном рендере. Вы можете сделать так, чтобы React пропускал повторные подписки если props.friend.id, который мы передали в ChatAPI, остался без изменений.

Так же, как и useState, вы можете использовать более одного эффекта в компоненте:

function FriendStatusWithCounter(props) {

const [count, setCount] = useState(0);

useEffect(() => {

document.title = `Вы нажали ${count} раз`;

});

const [isOnline, setIsOnline] = useState(null);

useEffect(() => {

ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);

return () => {

ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);

};

});

function handleStatusChange(status) {

setIsOnline(status.isOnline);

}

Хуки дают вам возможность организовать побочные эффекты в компоненте по связанным частям (например, добавление или отмена подписки) вместо того, чтобы принуждать вас делить всё согласно методам жизненного цикла.

**Основы тестирования**

Тестирование React-компонентов аналогично тестированию любого другого JavaScript-кода.

Есть несколько способов тестирования React-компонентов. В целом, можно выделить два основных:

Рендеринг деревьев компонентов в упрощенной тестовой среде и проверка их предполагаемого вывода.

Запуск всего приложения в реалистичной среде браузера (так называемые «сквозные» тесты).

Jest — исполнитель тестов на JavaScript, который позволяет взаимодействовать с DOM через jsdom. Несмотря на то, что jsdom только приблизительно реализует работу браузера, в большинстве случаев этого достаточно для тестирования React-компонентов. Jest предлагает отличную скорость итераций вместе с мощными возможностями, например фиктивные модули и таймеры, которые дают больше контроля над исполнением кода.

**Практическая часть:**

**Задание 1** – Создать приложение реакта через create react app. Создать в нем три основных компонента header – шапка, пропсами принимает логотип (ссылка на картинку) и название сайта, nav – панель навигации (в дальнейшем будет ссылаться на следующие ваше задания), footer - указание на автора задания.

**Задание 2** – Написать компонент по рейтингу, он принимает два пропса, 1 - максимальное количество звезд, 2 - сколько выбрано звезд. При выборе рейтинга сопровождать выбор алертом. если нажали на 3 звезду, то 3 и закрашиваются, если на 5 - то 5.

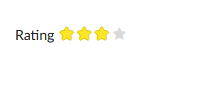


Рисунок 2 – Пример задания 2

**Задание 3** – Написать компонент для тегов, теги мы отправляем в таком формате:

tags = [

{ title: 'Тег №1', href: 'http://link1.ru'},

{ title: 'Тег №2', href: 'http://link2.ru'},

{ title: 'Тег №3', href: 'http://link3.ru'},

]



Рисунок 3 – Пример задания 3

**Задание 4** – Написать компонент, который принимает два компонента 1 – количество постов и 2 – лимит (Сколько будет выводится на страницу). Например если мы передали counts=47, limits=10, то у нас будут кнопки = Previous 1 2 3 4 5 и Next (Переход на каждую страницу по 10 постов, поэтому страниц 5 всего, так как мы указали, что выводить по 10 на страницу).

При загрузки компонента (использовать useEffect) отправляется запрос на <https://jsonplaceholder.typicode.com/posts>

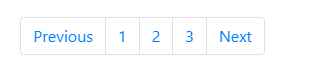


Рисунок 4 – Пример задания 4

**Ссылки для детального просмотра**:

Документация реакта - <https://ru.reactjs.org/docs/getting-started.html>