Лабораторная работа №1

Тема: «Доклад по библиотеке «React» для JS».

Выполнила Ермина С.Д. 4ИСИП-519

JavaScript-библиотека «React»

React – это JavaScript-библиотека для создания пользовательских интерфейсов. Обратите внимание, что это именно библиотека, а не фреймворк. React часто называют фреймворком, но это ошибка. Во-первых, его использование ни к чему вас не обязывает, не формирует «фрейм» проекта. Во-вторых, React выполняет единственную задачу: показывает на странице компонент интерфейса, синхронизируя его с данными приложения, и только этой библиотеки в общем случае недостаточно для того, чтобы полностью реализовать проект.

Вскоре после появления React и подобные ему решения (Vue.js, Svelte) практически захватили мир фронтенда: потому что они помогают решать проблемы, основываясь на идее декларативного программирования, а не на императивном подходе.

* Декларативный

Создавать интерактивные пользовательские интерфейсы на React — приятно и просто. Вам достаточно описать, как части интерфейса приложения выглядят в разных состояниях. React будет своевременно их обновлять, когда данные изменяются.

Декларативные представления сделают код более предсказуемым и упростят отладку.

При императивном подходе описываются конкретные шаги для достижения конечного результата (как мы хотим что-то получить).

* Основан на компонентах

Создавайте инкапсулированные компоненты с собственным состоянием, а затем объединяйте их в сложные пользовательские интерфейсы.

Поскольку логика компонента написана на JavaScript, а не содержится в шаблонах, можно с лёгкостью передавать самые разные данные по всему приложению и держать состояние вне DOM.

* Научитесь однажды — пишите где угодно

Нам не нужно ничего знать про остальную часть вашего технологического стека, поэтому вы можете разрабатывать новую функциональность на React, не изменяя существующий код. React также может работать на сервере, используя Node.js и на мобильных платформах, используя React Native.

* Простой компонент

React-компоненты реализуют метод render(), который принимает входные данные и возвращает что-то для вывода. В этом примере используется XML-подобный синтаксис под названием JSX. Входные данные, передаваемые в компонент, доступны в render() через this.props.

JSX необязателен для работы с React. Попробуйте Babel REPL, чтобы увидеть JavaScript-код, полученный на этапе компиляции JSX.

Пример:

class HelloMessage extends React.Component {

render() {

return (

<div>

Привет, {this.props.name}

</div>

);

}

}

root.render(<HelloMessage name="Стефан" />);

* Компонент с состоянием

Помимо входных данных (доступных через this.props), компонент поддерживает внутренние данные состояния (доступные через this.state). Когда данные состояния компонента изменятся, React ещё раз вызовет render() и обновит отрендеренную разметку.

Пример:

class Timer extends React.Component {

constructor(props) {

super(props);

this.state = { seconds: 0 };

}

tick() {

this.setState(state => ({

seconds: state.seconds + 1

}));

}

componentDidMount() {

this.interval = setInterval(() => this.tick(), 1000);

}

componentWillUnmount() {

clearInterval(this.interval);

}

render() {

return (

<div>

Секунды: {this.state.seconds}

</div>

);

}

}

* Приложение

Используя props и state, можно создать небольшое приложение списка дел. В этом примере используется state для отслеживания текущего списка элементов, а также текста, введённого пользователем. Хотя обработчики событий встроены в разметку, они собираются и реализуются с помощью делегирования событий.

Пример:

class TodoApp extends React.Component {

constructor(props) {

super(props);

this.state = { items: [], text: '' };

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

render() {

return (

<div>

<h3>Список дел</h3>

<TodoList items={this.state.items} />

<form onSubmit={this.handleSubmit}>

<label htmlFor="new-todo">

Что нужно сделать?

</label>

<input

id="new-todo"

onChange={this.handleChange}

value={this.state.text}

/>

<button>

Добавить #{this.state.items.length + 1}

</button>

</form>

</div>

);

}

handleChange(e) {

this.setState({ text: e.target.value });

}

handleSubmit(e) {

e.preventDefault();

if (this.state.text.length === 0) {

return;

}

const newItem = {

text: this.state.text,

id: Date.now()

};

this.setState(state => ({

items: state.items.concat(newItem),

text: ''

}));

}

}

class TodoList extends React.Component {

render() {

return (

<ul>

{this.props.items.map(item => (

<li key={item.id}>{item.text}</li>

))}

</ul>

);

}

}

* Компонент с использованием внешних плагинов

React позволяет работать с другими библиотеками и фреймворками. В этом примере используется внешняя библиотека remarkable, чтобы в режиме реального времени преобразовать Markdown-синтаксис, введённый в элемент <textarea>.

Пример:

class MarkdownEditor extends React.Component {

constructor(props) {

super(props);

this.md = new Remarkable();

this.handleChange = this.handleChange.bind(this);

this.state = { value: 'Привет, \*\*мир\*\*!' };

}

handleChange(e) {

this.setState({ value: e.target.value });

}

getRawMarkup() {

return { \_\_html: this.md.render(this.state.value) };

}

render() {

return (

<div className="MarkdownEditor">

<h3>Редактор</h3>

<label htmlFor="markdown-content">

Введите что-нибудь

</label>

<textarea

id="markdown-content"

onChange={this.handleChange}

defaultValue={this.state.value}

/>

<h3>Вывод</h3>

<div

className="content"

dangerouslySetInnerHTML={this.getRawMarkup()}

/>

</div>

);

}

}