# React

## УСТАНОВКА YARN

Инсталлим node.js с сайта, после этого через npm инсталлим yarn:

npm i yarn -g

yarn -v

## УСТАНОВКА WEBPCK (ЛОКАЛЬНО)

Через yarn:

yarn add webpack

yarn add webpack-dev-server

yarn add babel-core babel-loader

После этого нужно в **package.json** добавить скрипт,

**build** - для продакшн версси, единожды запускает webpack, прогоняет все файлы через себя и babel, выливает в bundle.js

**dev-server** - для девелопмент версии, запускает веб-сервер, все файлы держит в оперативке

"scripts": {

"build": "webpack",

"dev-server": "webpack-dev-server"

},

## WEBPACK.CONFIG.JS

Все настройки по всем установленным модулям webpack берет из файла **webpack.config.js**.

Необходимо создать файл в корне проекта, в него пишем:

const path = require('path');

module.exports = {

entry: "./src/app.js",

output: {

path: path.join(\_\_dirname, 'public'),

filename: "bundle.js"

},

module: {

rules: [{

loader: 'babel-loader',

test: /\.js$/,

exclude: /node-modules/

}]

},

devtool: 'cheap-module-eval-source-map',

devServer: {

contentBase: path.join(\_\_dirname, 'public')

}

};

## .BABELRC

После этого необходимо в корне проекта создать **.babelrc**, куда добавить пресеты babel

{

"presets": ["env", "react"],

"plugins": ["transform-class-properties"]

}

## ЗАПУСК

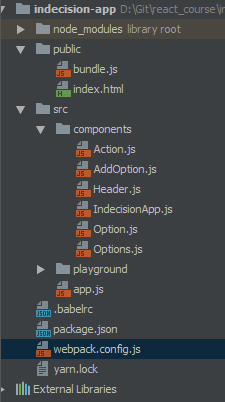
После этого можно запустить webpack, который возьмет файл src/app.js (из webpack.config.js) и выльет его в папку public/bundle.js. Это версия для продакшна

yarn run webpack

Для девелопмента, что бы запустить веб-сервер, пишем,

yarn run webpack-dev-server

## ФАЙЛОВАЯ СТРУКТУРА REACT

Каждый компонент в отдельном js-файле

**app.js** выглядит так:

import React from 'react';

import ReactDOM from 'react-dom';

import IndecisionApp from './components/IndecisionApp';

ReactDOM.render(

<IndecisionApp/>, document.getElementById('app')

);

**IndecisionApp.js** так:

import React from 'react';

import Header from './Header';

import Action from './Action';

import Options from './Options';

import AddOption from './AddOption';

export default class IndecisionApp extends React.Component {

constructor(props) {

super(props);

...

## СИНТАКСИС REACT ШАБЛОНОВ

Шаблоны React'a принимают только утверждения (statements) либо тернарные операторы, но не выражения (expressions)

true, false, null, undefined - не будут отображены

const user = {name: 'Alex', age: 17, location: 'Kiev'};

function getLocation(location) {

if (location) return <p>Location: {location}</p>;

}

// покажет имя или Анонимус, если имя undefined

// проверит, есть ли возраст и больше ли он 18, если true - отобразит строку с возрастом

// отобразит строку с локацией или вообще ничего не покажет

const template2 = (

<div>

<h1>{user.name ? user.name : 'Anonymous'}!</h1>

{(user.age && user.age >= 18) && <p>Age: {user.age}</p>}

{getLocation(user.location)}

</div>

);

const userRoot = document.getElementById('user');

ReactDOM.render(template2, userRoot);

JSX так же поддерживает шаблоны внутри шаблонов. Можно обрабатывать массивы, но не обьекты:

<ul>

{

app.options.map((option, i, arr) => {

return <li key={i}>{option}</li>

})

}

</ul>

## РЕНДЕР

Некоторые свойства HTML нельзя писать напрямую в шаблоны JSX. Например class, вместо него используется className.

<https://reactjs.org/docs/dom-elements.html> - внизу можно посмотреть список неизмененных свойств

<https://reactjs.org/docs/events.html#form-events> - HTML эвенты

ReactDOM.render() работает крайне эффективно за счет виртуального DOM-дерева, потому даже если рендерить после небольшого изменения целую страничку сразу этим методом, Реакт отследит изменения и внесет их только в те части на стороне клиента, где они нужны

## КОМПОНЕНТЫ

Реакт имеет метод React.Component что является обычным родительским классом. Синтаксис:

class Header extends React.Component{

render(){

return <p>This if Header</p>

}

}

Все компоненты создаются с Прописной буквы. Если без прописной, Реакт пытается отрендерить элемент как строчный, вместо передачи класса

Можно отрендерить компонент внутри компонента:

class IndecistionApp extends React.Component {

render(){

return {

<div>

<Header/> {}

<Action/> {}

</div>

}

}

}

ReactDOM.render(<IndecistionApp/>, document.getElementById('app'));

Цикл внутри компонента внутри компонента с привязкой this к обработчику события. Когда функция вешается на элемент html как обработчик события, this теряется

class Options extends React.Component {

constructor(props) {

super(props);

this.handleRemoveAll = this.handleRemoveAll.bind(this);

}

handleRemoveAll() {

this.props.options.length = 0;

}

render() {

return (

<div>

<button onClick={this.handleRemoveAll}>Remove all</button>

<ol>

{

this.props.options.map((option, i, arr) => {

<Option key={i} optionText={option}/>

})

}

</ol>

</div>

);

}

}

class Option extends React.Component {

render() {

return <li key={this.props.index}>{this.props.optionText}</li>

}

}

ReactDOM.render(<Options />, document.getElementById('app'));

## СОСТОЯНИЕ КОМПОНЕНТА (COMPONENT STATE)

Реакт автоматически отслеживает изменения состояния, что бы рендерить их сразу в UI. Состояния - обычные пары ключ\значение, которые пишутся как свойства обьекта в компонент. Что бы создать состояние, нужно в конструкторе создать обьект state, а в нем задать свои свойства. Доступ к свойствам осуществляется через метод наследуемый метод setState:

class Counter extends React.Component {

constructor(props){

super(props);

this.handleAddOne = this.handleAddOne.bind(this);

this.state = {

count: 0

}

}

handleAddOne(){

this.setState((prevState) => {

return {

count: prevState.count + 1

}

})

}

render(){

return (

<div>

<h1>Count: {this.state.count}</h1>

<button onClick={this.handleAddOne}>+1</button>

</div>

)

}

}

ReactDOM.render(<Counter/>, document.getElementById('app'));

**Передача функций**

Что бы вложенные компоненты могли обрабатывать и изменять состояния всего приложения или других компонентов, в родительском компоненте создаются функции, которые передаются детям как обработчики:

class IndecistionApp extends React.Component {

constructor(props) {

super(props);

this.state = {

options: ['Item 1', 'Item 2', 'Item 3', 'Item 4',]

};

this.handlePick = this.handlePick.bind(this);

}

handlePick() {

const randomNum = Math.floor(Math.random() \* this.state.options.length);

const option = this.state.options[randomNum];

alert(option);

}

render() {

return (

<div>

<h1>Super app</h1>

<Action handlePick={this.handlePick} />

</div>

)

}

}

class Action extends React.Component {

render() {

return <button onClick={this.props.handlePick}>What should I do?</button>

}

}

ReactDOM.render(<IndecistionApp/>, document.getElementById('app'));

**Обработка ошибок и передача информации в функции**

Ошибка вызывается за счет того, что при успехе изменяется this.state.options, а возвращается ничего. При ошибке возвращается строка, которая является true, соответственно элемент отображается

class IndecistionApp extends React.Component {

constructor(props) {

super(props);

this.state = {

options: []

};

this.handleAddOption = this.handleAddOption.bind(this);

}

handleAddOption(option) {

if(!option){

return 'Enter valid value';

} else if(this.state.options.indexOf(option) > -1){

return 'This option already exists'

}

this.setState((prevState) => {

return {

options: prevState.options.concat(option)

}

});

}

render() {

const title = 'Indecision';

return (

<div>

<h1>Indecision</h1>

<AddOption handleAddOption={this.handleAddOption} />

</div>

)

}

}

class AddOption extends React.Component {

constructor(props){

super(props);

this.handleAddOptionHelper = this.handleAddOptionHelper.bind(this);

this.state = {

error: undefined

}

}

handleAddOptionHelper(e) {

e.preventDefault();

const option = e.target.elements.option.value.trim();

const error = this.props.handleAddOption(option);

this.setState(() => {

return { error }

})

}

render() {

return (

<div>

{this.state.error && <p>{this.state.error}</p>}

<form onSubmit={this.handleAddOptionHelper}>

<input type="text" name="option"/>

<button>Add option</button>

</form>

</div>

)

}

}

ReactDOM.render(<IndecistionApp/>, document.getElementById('app'));

## STATELESS FUNCTIONAL COMPONENT

В качестве компонента используется обыкновенная функция. Плюсы - нет лишнего кода, он быстрее и легче тестируется. Минусы - нет объекта state, т.е. нельзя менять состояние

defaultProps - передача параметров по умолчанию, если явно не указан какой-то из них

const User = (props) => {

return (

<div>

<p>name: {props.name}</p>

<p>age: {props.age}</p>

</div>

)

};

// свойство по умолчанию, если не указано в свойствах при рендере

User.defaultProps = {

name: 'Anonymous'

};

ReactDOM.render(<User name="Alex" age={26}/>, document.getElementById('app'));

## ПРОБРОС СВОЙСТВ В CHILD COMPONENTS

const Option = (props) => {

return (

<div data-option-index={props.optionIndex}>

{props.optionText}

<button

onClick={(e) => {

props.handleDeleteOptionSingle(props.optionIndex)

}}

>

remove

</button>

</div>

)

};

Если надо передать какую-то функцию детям, которые например динамически генерируются на каком-то уровне прослойке, функцию можно передавать как обычное свойство от компонента к компоненту, а на самом нижнем уровне вызвать ее анонимной функцию. Если вызвать ее напрямую, то будет передано не то, что нам нужно ( в данном случае prop.optionIndex), а event - e

## COMPONENTS LIFECYCLE METHODS

Вызвать можно только из class-based components, из stateless не выйдет

componentDidMount() - че то типа DOMContentLoaded, вызывается, когда компонент построил дерево

componentDidUpdate(prevProps, prevState) - вызвается, когда компонент обновляет props или state

componentWillUnmount() - вызывается перед удалением компонена

<https://reactjs.org/docs/react-component.html> - остальные функции тут

## ПЕРЕДАЧА ШАБЛОНА ЧЕРЕЗ СВОЙСТВА

**Подход 1.**

const Layout = (props) => {

return (

<div>

<p>Header</p>

{props.content}

<p>Footer</p>

</div>

)

};

const template = (

<div>

<h1>Page title</h1>

<p>This is my page</p>

</div>

);

ReactDOM.render(<Layout content={template}/>, document.getElementById('app'));

**Подход 2. Через props.children**

const Layout = (props) => {

return (

<div>

<p>Header</p>

{props.children}

<p>Footer</p>

</div>

)

};

ReactDOM.render(<Layout><p>This is inline</p></Layout>, document.getElementById('app'));

## THIRD-PARTY LIB IN REACT (REACT-MODULE)

Например, ставим библиотеку для модальных окон.

yarn add react-modal

После попытки запуска у меня посыпались ошибки компиляции через babel, пришлось доставить пресет и модуль:

yarn add babel-plugin-add-module-exports

yarn add babel-preset-stage-2

В .babelrc соотвественно дописываем:

{

"presets": ["env", "react", "stage-2"],

"plugins": ["transform-class-properties", "add-module-exports"]

}

Создаем модуль. Делаем отдельный файл в Components, назовем его **OptionModule**

import React from 'react';

import Modal from 'react-modal';

const OptionModal = () => <div>some text</div>;

export default OptionModal;

Теперь его можно спокойно импортировать и добавить в родительский модуль, как любой другой