## Практика 02

## Создание набора инструментальных средств для сборки приложений React с нуля

В этом упражнении мы вручную настроим набор инструментов для сборки приложений с нуля, чтобы посмотреть, как менеджер пакетов, сборщик пакетов и компилятор используются вместе. Обратите внимание, что следующей практикой будет использование официально рекомендованного пакета **create-react-app** для инициализации приложения.

### Запустите новый проект

1. Откройте командную строку, перейдите в выбранную вами папку и запустите:

**mkdir mytoolchain && cd mytoolchain**

1. Запустите **npm init -y**
2. Перейдите на терминал и установите react и react-dom:

**npm i react react-dom**

### Hello, World! в React

1. Создайте страницу **index.html** с:

<!doctype html>  
<html>  
<head>  
<meta charset="utf-8">  
 <title>Awesome React Project</title>  
</head>  
<body>  
  
</body>  
</html>

1. Добавьте скрипты react и react-dom и создайте элемент dom, в который будет помещен результат:

<!doctype html>  
<html>  
<head>  
 <meta charset="utf-8">  
 <title>Awesome React Project</title>  
 <script src="node\_modules/react/umd/react.development.js"></script>  
 <script src="node\_modules/react-dom/umd/react-dom.development.js"></script>  
</head>  
<body>  
<app id="app">  
  
</app>  
<script src="js/app.js"></script>  
</body>  
</html>

1. Создайте папку js и поместите в нее файл app.js (mkdir js && cdnjs).
2. Добавьте следующее содержимое в app.js:

import ReactDOM from 'react-dom/client';

const root = ReactDOM.*createRoot*(document.getElementById('root'));  
root.render(  
 <React.StrictMode>  
 <App />  
 </React.StrictMode>  
);

1. Откройте index.html в браузере. Отлично! Мы только что создали компонент React, используя чистый JavaScript: 
2. Хорошо, но давайте сделаем что-то немного более сложное и создадим простую разметку, например div > h1 ul > li li > h2:

let app = React.*createElement*('div', {},  
 React.*createElement*('h1', {}, "Hi, I'm header inside div"),  
 React.*createElement*('ul', {},  
 React.*createElement*('li', {},  
 React.*createElement*('h2', {}, "Hi, I'm list item inside list inside div")  
 ),  
 React.*createElement*('li', {},  
 React.*createElement*('h2', {}, "Hi, I'm list item inside list inside div")  
 )  
 )  
);  
  
const root = ReactDOM.*createRoot*(document.getElementById('root'));  
root.render(app);

1. Код выглядит нормально? Как вы думаете, легко ли читать такой код? – Ну уж нет!
2. Давайте начнем использовать jsx. Для этого нам понадобится WebPack и конвертер JSX:

npm i webpack -g  
npm i webpack-cli -g  
npm i -D webpack

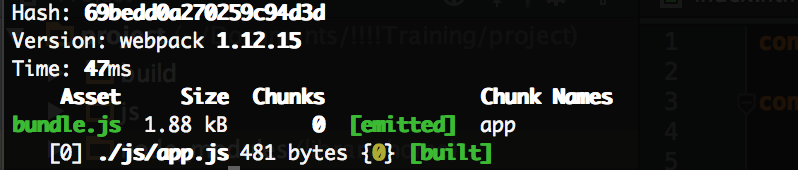
1. Позвольте объяснить, что мы сделали. Мы установили webpack глобально (флаг -g), чтобы быть уверенными, что он доступен везде (вместо того, чтобы каждый раз обращаться к node\_modules/webpack/.bin/xxx). Из коробки webpack может объединять ваши файлы javascript.
2. Прямо сейчас давайте используем webpack для сборки нашего чистого JS-решения. Создайте фай **webpack.config.js** в корне вашего проекта и поместите туда этот код:

const path = require('path');  
  
const PATHS = {  
 app: "./js/app",  
 build: path.join(\_\_dirname, 'build')  
};  
  
module.exports = {  
 mode: 'development',  
 entry: {  
 app: PATHS.app  
 },  
 output: {  
 path: PATHS.build,  
 filename: 'bundle.js'  
 }  
};

1. Мы указываем webpack выполнить упаковку из нашей точки входа и дать имя входному файлу bundle.js. Хорошо. У нас пока нет файла bundle.js, но предположим, что он у нас уже есть. Обновите index.html.

<!doctype html>  
<html>  
<head>  
 <meta charset="utf-8">  
 <title>Awesome React Project</title>  
 <script src="node\_modules/react/umd/react.development.js"></script>  
 <script src="node\_modules/react-dom/umd/react-dom.development.js"></script>  
 *<!--<link rel="stylesheet" href="css/main.css">-->*</head>  
<body>  
<app id="app">  
  
</app>  
  
<script src="build/bundle.js"></script>  
</body>  
</html>

1. Хорошо, похоже, мы готовы к созданию файла. Введите ‘webpack’ в корень вашего проекта. Вы увидите что-то вроде этого:



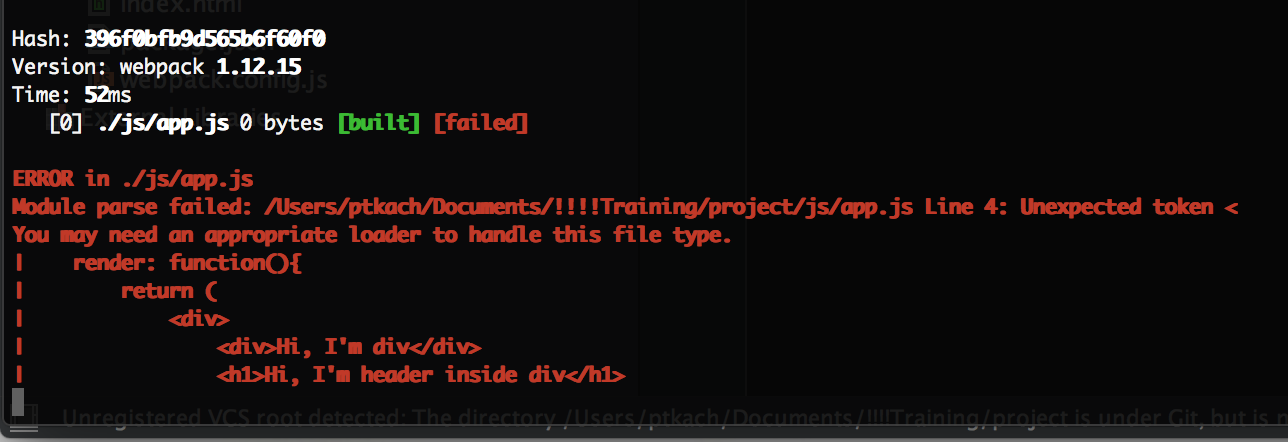
1. Откройте index.html в браузере. Ничего не изменилось? Потрясающе! Это как раз то, что мы искали.
2. Думаю, что после того, как вы 10 раз ввели webpack, чтобы перестроить пакет, вам надоело это делать. Можно упростить процесс, запустив webpack в режиме “следить за изменениями” следующим образом:

webpack -w

Насколько вам комфортно с этой разметкой React.js ? Выглядит странно, не так ли? Да, пожалуй. Давайте заменим ее на JSX, чтобы было понятнее:

function JsxApp () {  
 return (  
 <div>  
 <div>Hi, I'm div</div>  
 <h1>Hi, I'm header inside div</h1>  
 <ul>  
 <li>Hi, I'm list item inside list inside div</li>  
 <li>Hi, I'm list item inside list inside div</li>  
 </ul>  
 </div>  
 )  
}  
  
const root = ReactDOM.*createRoot*(document.getElementById("app"));  
root.render(<JsxApp/>);

1. Запустите webpack и загляните в консоль. Теперь вы увидите там следующее:



1. Похоже, нам нужен загрузчик, чтобы webpack мог работать с JSX. В конце концов, мы должны получить вот такую настройку:
   1. **Webpack**: наш сборщик модулей;
   2. **babel**-**core:** пакет для перевода JSX в JS, и ES6 в ES5;
   3. **babel presets:** различные настройки для разных типов компиляций, которые могут нам понадобиться. Мы будем использовать пресеты для приложений React (JSX -> JS) и для ES2015 (ES6-> ES5);
   4. **Webpack babel loader:** Загрузчик, который позволяет нам использовать транслятор babel из webpack.
2. Сначала установите babel core, babel loader и необходимые пресеты:

npm i @babel/core babel-loader @babel/preset-env @babel/preset-react --save-dev

1. Перейдите в webpack.config.js и разместите там следующий код:

const path = require('path');  
  
const PATHS = {  
 app: "./js/app",  
 build: path.join(\_\_dirname, 'build')  
};  
  
module.exports = {  
 devtool: 'source-map',  
 mode: 'development',  
 entry: {  
 app: PATHS.app  
 },  
 module: {  
 rules: [  
 {  
 test: /\.(js|jsx)$/,  
 exclude: /nodeModules/,  
 use: {  
 loader: 'babel-loader',  
 options: {  
 presets: [  
 "@babel/preset-react"  
 ]  
 }  
 }  
 }  
 ]  
 },  
 output: {  
 path: PATHS.build,  
 filename: 'bundle.js'  
 }  
};

1. Позвольте объяснить, что мы сделали. Иы добавили 2 вещи:
   1. Devtool: Мы заставили webpack создать для нас карту с полным исходным кодом (на всякий случай, если нам понадобится использовать отладчик);
   2. Модуль: Это самое интересное. Мы попросили webpack использовать загрузчик babel (установленный ранее) с конфигурациями react (также установленными ранее) для любых файлов \*.js или \*.jsx, исключая папку node\_modules. Почему мы должны исключить node\_modules? Поскольку в этой папке хранятся файлы библиотеки, которые уже были обработаны, поэтому нет необходимости компилировать и создавать их заново.
2. Теперь перезапустите webpack и откройте страницу index.html в браузере:



1. Отлично, работает. Итак, мы заменили JS на JSX (и это выглядит намного лучше), настроили webpack на использование JSX с загрузчиком Babel и преобразовали JSX в JS с помощью webpack.
2. Если мы используем webpack, который предоставляет нам все возможности для работы с модулями, то думаю, нет необходимости загружать React и ReactDOM в качестве глобальных скриптов:

<head>  
 <meta charset="utf-8">  
 <title>Awesome React Project</title>  
 <script src="node\_modules/react/umd/react.development.js "></script>  
 <script src="node\_modules/react-dom/umd/react-dom.development.js"></script>  
</head>

1. Теперь давайте удалим скрипты и загрузим их через webpack. Добавьте следующие строки в верхней части app.js:

import React from 'react';  
import ReactDOM from 'react-dom/client';

1. Если вы посмотрите на сгенерированный код, то увидите, что webpack сгенерировал код ES2015 (поддерживается любым современным браузером).
2. Если вы хотите, чтобы ваш код поддерживался более старым браузером, нужно будет добавить ‘preset-env’ в presets:

presets: ['@babel/preset-env', '@babel/preset-react']

1. Мы объединили webpack, babel + babel es2015, пресеты react, модули react и reactDOM – и вся инфраструктура готова!
2. Давайте займемся еще чем-нибудь полезным. Создадим таблицу, используя стили Bootstrap. Для этого нам нужно установить bootstrap и загрузчик css webpack (да, webpack тоже может обрабатывать css).

npm install bootstrap

npm install file-loader css-loader style-loader url-loader --save-dev

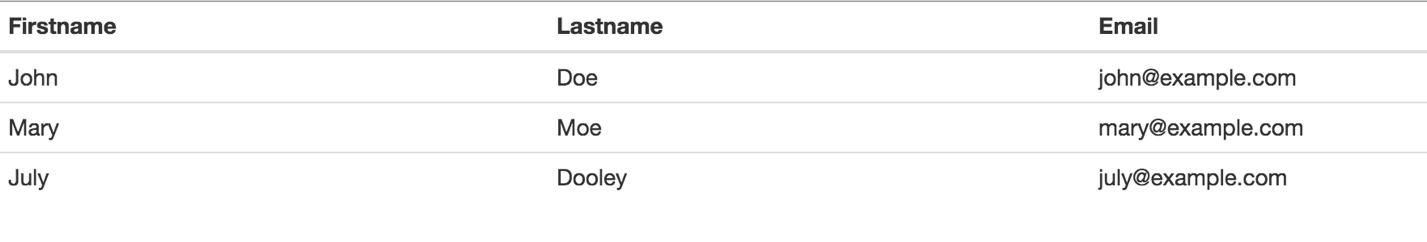
1. Обновите **webpack.config.js**, добавив к нему новые загрузчики:

rules: [  
 ...  
 { test: /\.css$/, use: ['style-loader', 'css-loader'] },  
 { test: /\.(png|woff|woff2|eot|ttf|svg)$/, use: ['url-loader?limit=100000'] }  
]

1. Обновите компонент:

function GridComponent() {  
 return (  
 <table className="table table-condensed">  
 <thead>  
 <tr>  
 <th>Firstname</th>  
 <th>Lastname</th>  
 <th>Email</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr>  
 <td>John</td>  
 <td>Doe</td>  
 <td>john@example.com</td>  
 </tr>  
 <tr>  
 <td>Mary</td>  
 <td>Moe</td>  
 <td>mary@example.com</td>  
 </tr>  
 <tr>  
 <td>July</td>  
 <td>Dooley</td>  
 <td>july@example.com</td>  
 </tr>  
 </tbody>  
 </table>  
 )  
}  
  
const root = ReactDOM.createRoot(document.getElementById("app"));  
  
root.render(  
 <GridComponent/>  
);

1. Откройте браузер:



Очень хорошо! Теперь у нас есть первый компонент пользовательского интерфейса, написанный на JSX, CSS–стили из bootstrap загружены через webpack — и все работает. Браво!