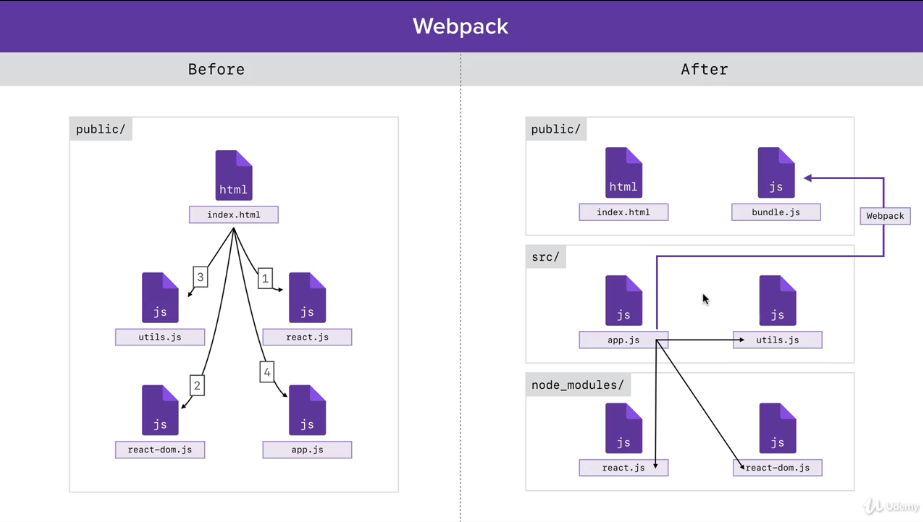
**WEBPACK**



WebPack в отличии от других пакетных менеджеров (Grunt, Gulp) может собирать файлы из разрозненных частей (через javascript module exports) в один большой файл, который при указании прогоняет через Babel, минифицирует и вставляет на фронтенд под названием bundle.js

# УСТАНОВКА WEBPCK (ЛОКАЛЬНО)

Через yarn:

yarn add webpack

yarn add live-server

yarn add webpack-dev-server

yarn add babel-core babel-loader

yarn add babel-plugin-transform-class-properties

yarn add css-loader style-loader sass-loader node-sass

yarn add normalize.css

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": {

"serve": "live-server public/",

"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/

},{

test: /\.s?css$/,

use: [

'style-loader',

'css-loader',

'sass-loader'

]

}]

},

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

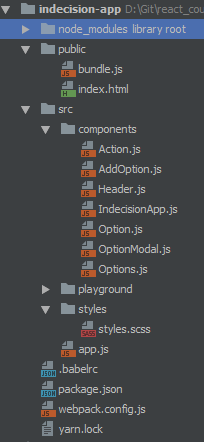
После этого можно запустить webpack, который возьмет файл src/app.js (из webpack.config.js) и выльет его в папку public/bundle.js. Это версия для продакшна

yarn run webpack

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

yarn run webpack-dev-server

# ПОДРОБНО

Структура должна выглядеть так, если рабочий основной файл - src/app.js, а итоговый файл - bundle.js

# УСТАНОВКА DEV-SERVER

webpack имеет свой дев-сервер, который работает быстро, потому что заточен именно под webpack. А еще он не требет физического расположения конечного собранного файла типа bundle.js: сервер сконфигурирован так, что все держит в памяти.

Из-под корня проекта пишем:

yarn add webpack-dev-server

# BABEL В WEBPACK

Устанавливаем babel core и babel-loader

yarn add babel-core babel-loader

После установки в webpack.config.js необходимо их добавить в rules. Дописываем:

module: {

rules: [{

loader: 'babel-loader',

test: /\.js$/,

exclude: /node-modules/

}]

},

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

**loader** - модуль, который заинтсллили

**test** - какие файлы модуль должен искать? (тут все файлы, у которых окончание .js)

**exclude** - папка для исключения поиск в

**devtool** - babel-source maps для отображения ошибок помодульно. Подробнее - <https://webpack.js.org/configuration/devtool/#devtool>

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

{

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

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

}

# SCSS В WEBPACK

Для начала необходимо заинсталлить модули:

*css-loader* (<https://www.npmjs.com/package/css-loader>) необходим для webpack, что бы взять CSS и превратить его в JS обьект для дальнейшей работы с ним

*style-loader*(<https://www.npmjs.com/package/style-loader>) - берет созданный выше обьект с CSS стилями и вставляет его в HTML DOM, в <style> тэг

*sass-loader* - делает JS образ из SCSS кода

*node-sass* - делает из JS образа SCSS кода обычный CSS код, используется самостоятельно в sass-loader

yarn add css-loader style-loader sass-loader node-sass

В **webpack.config.js** в модули добавляем новые правила. Вебпак так и продолжит работать с загрузчиком babel, но добавит к нему еще 3 из "use". **Последовательность расположения loader'ов в "use" важна!** Первые два отвечают за обработку CSS, последний - за SCSS

module: {

rules: [{

loader: 'babel-loader',

test: /\.js$/,

exclude: /node-modules/

},{

test: /\.scss$/,

use: [

'style-loader',

'css-loader',

'sass-loader'

]

}]

},

test обрабатывает все файлы, который оканчиваются либо на css, либо на scss

# PRODUCTION BUILD

Не сжатый файл весит более 6 мб - из-за того, что там много source maps. Вебпак делает все настрйоки сам, если запускать его с флагом -p. Так же нужно передать туда флаг на передачу окружения в функцию webpack.config.js Изменим скрипт в **package.json**:

"scripts": {

"serve": "live-server public/",

"build:dev": "webpack",

"build:prod": "webpack -p --env production",

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

"test": "jest --config=jest.config.json"

}

Вынесем css файлы из bundle.js с помощью плагина. Поскольку вынос css source maps до сих пор головная боль разработчиков, их настройка идет достаточно сложно. Если на продакшне все достаточно хорошо, то на development эти мапы могут работать некорректно

yarn add extract-text-webpack-plugin

Далее необходимо внести изменения в **webpack.config.js.**

Для начала его необходимо превратить в функцию, которая будет возвращать обьект с настройками. В ней нужно проверять окружение, в котором запущен вебпак. Окружение передается выше, из скрипта (--env production). Если окружение - production, source maps и CSS будут вынесены в отдельный файл, что существенно сократит размер bundle.js:

**const path = require('path');**

**const ExtractTextPlugin = require('extract-text-webpack-plugin');**

**module.exports = (environment) => {**

**const isProduction = environment === 'production';**

**const CSSExtract = new ExtractTextPlugin('styles.css');**

**return {**

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

**output: {**

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

**filename: "bundle.js"**

**},**

**module: {**

**rules: [{**

**loader: 'babel-loader',**

**test: /\.js$/,**

**exclude: /node-modules/**

**}, {**

**test: /\.s?css$/,**

**use: CSSExtract.extract({**

**use: [**

**{**

**loader: 'css-loader',**

**options: {sourceMap: true}**

**},**

**{**

**loader: 'sass-loader',**

**options: {sourceMap: true}**

**}**

**]**

**})**

**}]**

**},**

**plugins: [CSSExtract],**

**devtool: isProduction ? 'source-map' : 'inline-source-map',**

**devServer: {**

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

**historyApiFallback: true**

**}**

**}**

**};**

Теперь можно написать

yarn run build:prod

Будет созданы файлы: bundle.js, bundle.js.map, styles.css, styles.css.map. При чем bundle.js.map будет грузится на стороне клиента только тогда, когда он открывает консоль разработчика