1. Открываем консоль, прописываем команду: npm init

После чего создаём файл package.json со след настройками:

package name: (wptest2) - имя проекта, главное не называть проект webpack а то будет конфликт

version: (1.0.0) - можно оставить такой же..

description: - можно что нить написать.. к примеру обучение WebPack

entry point: (index.js) - пока оставляем по умолчанию

test command: - не нужно

git repository: - не нужно

keywords: js, javaScript, webPack - можно и не писать

author: coder1 <c.o.d.e.r.1@mail.ru>

license: (ISC) - по умолчанию

OK

1. В этом файле заранее вносим правки:

"scripts": {

"dev": "cross-env NODE\_ENV=webpack --mode development", - собирать в режиме разработки

"build": "cross-env NODE\_ENV=webpack --mode production", - в режиме продакшена (минифицированные файлы)

"watch": "cross-env NODE\_ENV=webpack --mode development --watch" - автопроверка на изменения и пересборка проекта

"start": "cross-env NODE\_ENV=webpack-dev-server --mode development --open " - запуститься сервер который будет обнавлять окно в браузере, ключ опен позволит сразу открыть окно браузера

}

"main": "index.js" на строку "private": "true"

1. Устанавливаем все необходимые пакеты для node\_modules, через скрипт inst.vbs
2. Создаём два каталога dist (где будет уже готовое приложение) и src (исходный код)
3. В src создаём файл index.js а в dist index.html
4. В index.js для теста создаём вывод алерта:

window.onload = function(e){

alert("WebPack");

}

1. В файле index.html за счёт Emmit создаём стандартную структуру документа:

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

</body>

<script src="main.js" type="text/javascript"></script>

</html>

1. В консоли вводим команду npm run dev после чего должна произайти сборка и в каталоге dist появятся файлы в том числе и main.js содержащий весь необходимый код. Если в браузере запустить страничку index.html то мы получим сообщение alert которое прописали в скрипте. Начало работы с ВебПаком положено, осталось развить это дело до совершенства.

В webpack.config.js в строке filename: main.js', меняем на bundle.js, в index.html имя скрипта так же меняем на bundle.js.

Добавляем объект module:

module.exports = {

context: path.resolve(\_\_dirname, 'src'), // корень исходников

mode: 'development', // собираем проект в режиме разработки

entry: './index.js',

output: {

filename: 'bundle.js',

path: path.resolve(\_\_dirname,'dist')

},

module: {

rules: [

{

test: /\.css$/,

use: [

'style-loader',

'css-loader'

],

},

{ // работа с препроцессором SCSS

test: /\.s[ac]ss$/,

use: [

'style-loader', // Создает узлы стиля из строк JS

'css-loader', // Переводит CSS в CommonJS

'sass-loader', // Компилирует Sass в CSS

],

},

{ // работа с файлами шрифтов

test: /\.(ttf|woff|woff2|eot)$/,

use: ['file-loader']

},

],

},

};

Тут мы добавили css , scss обработчик файлов в том числе для шрифтов.

Изменяем файл index.js:

import \* as $ from 'jquery' // подключаем джеквери библиотеку из папки node\_modules

import './style.scss' // все стили препроцессора всех компонентов и страниц

import './javascript.js' // весь js код, всех компонентов и страниц

Содержимое файла style.scss:

$font-adres: 'fonts/';

$font-name:(

'OpenSans'

);

@each $current-font in $font-name {

@font-face {

font-family: $current-font; /\* Имя шрифта \*/

src: url('#{$font-adres}#{$current-font}.woff2');

font-weight: normal;

font-style: normal;

}

}

@mixin helvetica($style : map-get($font-name, 0)) {

@each $current-font in $font-name {

@if $style == $current-font

{font-family: $current-font;}

}

}

body{

font-size: 26px;

@include helvetica('OpenSans');

color: blue;

}

Содержимое файла javascript.js:

window.onload = function(e){

alert("WebPack");

}

При данной конфигурации мы получим компиляцию scss файла в котором будет подгружен шрифт и применён к тегу body .. а в отдельном файле js будет выполнен код Алерта. Такой подход даёт нам возможность делать свой проект модульным, состоящим из множество файлов и подключаемых через импорты.

Что бы заработал локальный сервер с обновлением окна браузера при изменении данных в проекте, необходимо проделать следующие:

1. Установить пакеты именно этих версий (с новыми конфликт – вопрос пока не изучен)

"webpack": "^4.43.0",

"webpack-cli": "^3.3.11",

"webpack-dev-server": "^3.11.0",

1. Прописать конфигурацию для dev-server: ( module.exports )

devServer: { // локальный сервер который будет запущен на http://localhost:8081/

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

compress: true,

port: 8081,

watchOptions: {

poll: true

},

},

Когда работает сервер, консоль будет заблокирована. Что бы разблокировать её нужно нажать Ctrl+C затем ввести Y и нажать Enter.

Для обнаружения ошибок в файлах при сборке: (прописываем в module.exports)

devtool: 'inline-source-map', // показывает ошибки допущенные в файлах.

Полезные плагины: (module.exports)

plugins: [

new CleanWebpackPlugin(), // очищаем от лишних файлов в папке дист

new HTMLWebpackPlugin({ // работаем с html файлами

template: './index.html',

}),

],

Оптимизация: (module.exports)

optimization:{

splitChunks:{

chunks: 'all' // создаёт отдельные вендор файлы в которые кидает весь лишний код, при этом наш бандел файл перестаёт весить полтора мегобайта :).

}

},

Теперь доработаем конфиг и сделаем по серьёзней.

Все подключаемые модули:

const path = require('path') // доступ к основным адресам

const HTMLWebpackPlugin = require('html-webpack-plugin') // упрощает создаение HTML файлов, добавления хеша в имена файлов

const {CleanWebpackPlugin} = require('clean-webpack-plugin') // удаляет все ненужные файлы при перестройке проекта

const webpack = require('webpack');

const CopyPlugin = require('copy-webpack-plugin'); // Копирует отдельные файлы или целые каталоги, которые уже существуют, в каталог сборки.

const MiniCssExtractPlugin = require('mini-css-extract-plugin') // Он создает файл CSS для каждого файла JS, который содержит CSS

const UglifyJsPlugin = require('uglifyjs-webpack-plugin') // минимизация js

const fs = require('fs')

var ImageminPlugin = require('imagemin-webpack-plugin').default

Булевы флаги для переключения вида сборки:

const isDev = process.env.NODE\_ENV === 'development'

const isProd = !isDev

Константы для основных адресов:

const PATHS = {

src: path.join(\_\_dirname, './src'),

dist: path.join(\_\_dirname, './dist'),

assets: 'assets/'

}

Функция которая при сборке проверяет тип файла и ложит сгенерированный файл в dist/assets/\*\*

const filename = function(ext) {

let dir='';

if(ext === 'css')

{dir= `${PATHS.assets}css/`;} // ложим в каталог css

if(ext === 'js')

{dir= `${PATHS.assets}js/`;} // ложим в каталог js

if(isDev)

{return `${dir}[name].${ext}`;}

else

{return `${dir}[name].[hash].${ext}`;}

return '';

}

Оптимизация файлов:

// в зависемости от сборки выбераем оптимизацию на выходе

const optimization = () => {

const config = { // позволяет лишний код который используеться в нескольких js файлах вынести в один отдельный файл тем самым минимизируя основные.

runtimeChunk: 'single',

splitChunks: { // создаються файлы vendors

chunks: 'all' // создаёт отдельные вендор файлы в которые кидает весь лишний код, при этом наш бандел файл перестаёт весить полтора мегобайта :).

}

}

if (isProd) { // минимизируем код если собираем в продакшен

config.minimizer = [

// new OptimizeCssAssetWebpackPlugin(), // минимизация css файлов

new UglifyJsPlugin({ // сжимаем js файлы, минифицируем код

parallel: true, // пораллельная сборка - увеличивает скорость

cache: true, // кеширует данные что бы при повторной сборке собирать только то что изменилось.

}),

]

}

return config

}

Функция обработки css scss файлов:

const cssLoaders = extra => {

const loaders = [

{

loader: MiniCssExtractPlugin.loader,

options: {

hmr: isDev, // в зависимости от сборки ужимать файлы или нет

reloadAll: true,

},

},

'css-loader',

'postcss-loader', // обработчики подключаются в разделе Плагины

]

if (extra) {

loaders.push(extra)

}

return loaders

}

Бейбл обработчик:

const babelOptions = preset => {

const opts = {

presets: [

'@babel/preset-env'

],

plugins: [

'@babel/plugin-proposal-class-properties'

],

cacheDirectory: true,

}

if (preset) {

opts.presets.push(preset)

}

return opts

}

Функция обработчики js файлов:

const jsLoaders = () => {

const loaders = [{

loader: 'babel-loader',

options: babelOptions()

}]

if (isDev) {

loaders.push('eslint-loader') // позволяет проводить анализ качества вашего кода, написанного на любом выбранном стандарте JavaScript (файл конфигурации пишется отдельно)

}

return loaders

}

Файл конфигурации для eslint-loader располагается на одном уровне с webpack.config.js и имеет имя: .eslintrc

Содержимое файла:

{

"parser": "babel-eslint",

"rules": {

"no-unused-vars": "warn", // показывать ошибки и предупреждения по коду

"no-debugger": "off",

"no-console": 0

},

"env": { // подключаем технологии которые хотим поддерживать в нашем коде.

"es6": true,

"browser": true,

"jquery": true,

"node": true

},

"extends": [

"eslint:recommended"

]

}

Заменяем все страницы HTML на PUG и меняем конфу под него:

const PAGES\_DIR = `${PATHS.src}/pages/` // каталог где располагаються PUG файлы

const PAGES = fs.readdirSync(PAGES\_DIR).filter(fileName => fileName.endsWith('.pug')) // получаем все PUG файлы в данном каталоге

Изменяем корневые адреса на наши константы:

module.exports = {

context: PATHS.src, // корень исходников

mode: 'development', // собираем проект в режиме разработки

entry: './index.js', // входной файл (их может быть несколько)

output: {

filename: filename('js'),

path: PATHS.dist // каталог в который будет выгружаться сборка.

},

Делаем так что бы в index.js можно было писать короткие относительные пути и имена файлов

resolve: {

extensions: ['.js', '.scss'], // когда мы прописываем тут расширения то при импуте в index.js их можно не прописывать

alias: {

'@plugins': `${PATHS.src}plugins`,

'@styles': `${PATHS.src}/${PATHS.assets}styles`,

'@javascript': `${PATHS.src}/${PATHS.assets}js`,

'@': PATHS.src,

}

},

Плагины:

plugins: [

new CleanWebpackPlugin(), // очищаем от лишних файлов в папке дист

...PAGES.map(page => new HTMLWebpackPlugin({ // автоматическое добавление страниц PUG

template: `${PAGES\_DIR}/${page}`,

filename: `./${page.replace(/\.pug/,'.html')}`

})),

new webpack.HashedModuleIdsPlugin(), // позволяет не менять хеши у файлов код которых не был затронут.

new CopyPlugin({

patterns: [

{ from: `${PATHS.assets}img`, to: `../dist/${PATHS.assets}img/` }, // копируем все изображения в папку dist/assets/img

],

}),

new ImageminPlugin({ test: /\.(jpe?g|png|gif|svg|webp)$/i, // сжатие изображений работает только после плагина копирования

disable: !isProd // сжимать только в продакшене.

}),

new MiniCssExtractPlugin({

filename: filename('css')

}),

// ПостCss

require('autoprefixer'), // проставляет префиксы стилям

require('css-mqpacker'), // сжимает медиа запросы

require('cssnano')({ // максимально минифицирует исходные стили

preset: [

'default', {

discardComments: {

removeAll: true,

}

}

]

}),

new webpack.ProvidePlugin({ // подключаем jquery плагином, самый нормальный способ ..

$: 'jquery',

jQuery: 'jquery',

'window.jQuery': 'jquery'

})

],

Показ ошибок:

devtool: isDev ? 'inline-source-map' : '', // показывает ошибки допущенные в файлах. (карта для браузеров)

Делаем наш ДевСервер немного интересней:

devServer: { // локальный сервер который будет запущен на http://localhost:8081/

contentBase: PATHS.dist,

compress: true, // сжатие данных

port: 8081,

overlay: {

warnings: true, // показывает предупреждения.

errors: true // если true то будет показывать ошибки если они есть. (показывает в окне браузера)

},

watchOptions: {

poll: true // отслеживание изменений

},

},

Вызываем функцию оптимизации которая была описана ранее:

optimization: optimization(), // минификация и оптимизация файлов на выходе (если это Продакшен)

Обработчики файлов:

module: {

rules: [ // описание правил как вебПаку работать с тем или иным расширением файлов.

{

test: /\.css$/,

use: cssLoaders(),

},

{

test: /\.pug$/,

loader: 'pug-loader', // обработчик PUG файлов

options: {

pretty: isDev // минифицировать или нет в зависемости от типа зборки.

}

},

{ // работа с препроцессором SCSS

test: /\.s[ac]ss$/,

use: cssLoaders('sass-loader') // преобразовать scss в css (вызываем функцию описанную выше)

},

{ // работа с файлами шрифтов

test: /\.(ttf|woff|woff2|eot)$/,

loader: 'file-loader',

options: {

outputPath: `${PATHS.assets}fonts/`, // берём файлы по этому адресу

publicPath: '../fonts/', // и ложим их в папку dist/assets/fonts

},

},

{ // работа с js файлами

test: /\.js$/,

exclude: /node\_modules/, // игнорируем эту папку. что бы не обрабатывать файлы от туда.

use: jsLoaders() // вызываем функцию описанную выше

},

{ // работа с графическими файлами (обычно они подключены либо в index.js либо в файле стилей)

test: /\.(png|jpg|svg|gif|webp)$/,

loader: 'file-loader',

options: {

outputPath: `${PATHS.assets}img/decor`, // откуда берём

publicPath: '../img/decor/', // куда ложим

},

},

{ // работа с xml структурой

test: /\.xml$/,

use: ['xml-loader']

},

{ // работа с выгрузкой базы данных

test: /\.csv$/,

use: ['csv-loader']

},

],

},

На этом основную конфигурацию ВебПака для ФронтЭнд разработки можно считать законченной. Со временем можно и нужно дорабатывать конфу, использовать новые инструменты и ускорять свою разработку а главное делать более качественной.