**WEBPACK-JackCoder-tutorial**

**урок 1**

**git init**

**yarn init**

Будем использовать препроцессор stylus к нему подключим 2 плагина:

autoprefixer

....

Одно из отличий Webpack от Gulp это наличие точки входа - index.js, в нем размещаем все подключения библиотек.

Установим:

**yarn add -D webpack webpack-cli webpack-dev-server path**

Далее в корне проекта создаем и настраиваем webpack.config.js

const path = require('path');

module.exports = {

    entry: {

        app: \_\_dirname + '/src/index.js'

    },

    output: {

        filename: '[name].js',

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

        publicPath: '/dist' // Необходимо для dev-server

    },

    devServer: {

        overlay: true //Показ ошибок не в консоли, а в окне браузера

    }

};

Затем в корне проекта создаем index.html и подключаем к нему dist/app.js

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Webpack 4</title>

    <link rel="stylesheet" href="">

</head>

<body>

    <div class="wrapper">

        <h1>Webpack</h1>

    </div>

    <script src="/dist/app.js"></script>

</body>

</html>

Далее создаем src/index.js в нем будут подключатся все наши зависимости, затем, в качестве примера, создаем src/js/common.js

src/js/common.js:

let add = (a, b) => a + b;

console.log(add(2, 6));

И не забываем его подключить в src/index.js:

import './js/common'; // .js указывать не обязательно

**yarn dev**

Важно! в данном режиме (**--mode development**) папка /dist не создается, она хранится в памяти, это одно из кардинальных отличий webpack от gulp, за счет этого достигается его быстрота, так как ему не надо создавать папку /dist на харде, все хранится в RAM.

папка /dist будет создана только в режиме --mode production

**yarn build**

Далее для того чтобы мы могли спокойно использовать ES6 синтаксис и быть уверенными, что он запучтиться в старых браузерах нам необходимо установить Babel:

**yarn add -D @babel/core @babel/preset-env babel-loader**

Затем необходимо добавить и настроить babel в разделе module в webpack.config.js:

module: {

        rules: [

            {

                test: /\.js$/,

                loader: 'babel-loader', // Указываем через что необхожимо обрабатывать js файлы

                exclude: '/node\_modules/'

            }

        ]

    },

Далее в корне проекта надо конфиг для babel - .babelrc:

{

    "presets": [

        "@babel/preset-env"

    ]

}

**yarn build**

теперь в /dist/app.js находится транспилированный js в синтаксис ES5

Далее подключим сторонние js файлы, подключим Vue.js, для начала скачаем пакет vue:

**yarn add -D vue**

Затем покдлючим его, есть 4 способа:

*1й (самый популярный)*

импортировать его в index.js

import 'vue';

поиск начинаеется из папки /src

*2й способ:*

import Vue from 'vue';

это удобно если мы хотим взаимодействовать с переменной Vue, например:

Vue.use();

*3й способ:*

подвесить объект Vue на window, касаемо Vue более предпочтительный

window.Vue = require('vue');

*4й способ, например нам надо установить Bootstrap и нам нужны только стили или наоборот только JS*

import 'bootstrap/dist/css/bootstrap.min.css'

или если хотим взаимодействовать, то

import Bootstrap from 'bootstrap/dist/css/bootstrap.min.css'

Далее добавим в разделе module обработчик для css файлов:

{

                test: /\.css$/,

                loader: 'css-loader', // Указываем через что необхожимо обрабатывать js файлы

            }

В следующем видео сделаем правильную настройку webpack.config.js, так как использовать один файл конфигурации не совсем правильно, лучше всего использовать три файла:

1й для dev

2й для production

3й для merge, чтобы не копировать стили

**урок 2**

В этом уроке будем работать с препроцессорами.

Настроим css и scss добавляем:

* ./src/scss/main.css
* ./src/css/main.scss

затем подключаем ./src/scss/main.css в index.js

import './css/main.css';

Для что бы webpack в дальнейшем поместил код css из src/main.css в отдельный файл dist/app.css, его необходимо отделить от js кода, для этого установим **плагин**:

**yarn add -D mini-css-extract-plugin**

*как альтернатива есть еще css-extract-plugin, но 1й вариант предпочтительнее, так как mini создан под 4 webpack.*

далее в webpack.config.js подключаем обработчик css

const MiniCssExtractPlugin = require('mini-css-extract-plugin');

{

                test: /\.css$/,

                use: [

                    MiniCssExtractPlugin.loader,

                    'css-loader'

                ]

            }

//регистрируем плагины

    plugins: [

        new MiniCssExtractPlugin({

            filename: '[name].css',

        })

    ]

текущий [name].js|css|т.д. ссылается на ярлык app из entry point

далее установим **пакеты**, для обработки css кода:

**yarn add -D css-loader style-loader**

Затем в корне проекта в index.html подключаем стили:

<link rel="stylesheet" href="/dist/app.css">

внесем что-нибудь в scr/main.css:

.wrapper {

    display: flex;

}

далее **yarn build**

теперь наш код разделяется на .css и .js файлы, и в папке dist/ теперь два файла app.js и app.css

*Далее займемся scss*

в массив use лучше передавать объекты, так как в объектах помимо названия loader с ним можно передать конфиг обычно в options

{

                test: /\.scss$/,

                use: [

'style-loader',

                    MiniCssExtractPlugin.loader,

                    {

                        loader: 'css-loader',

                        options: { sourceMap: true}

                    }, {

                        loader: 'sass-loader',

                        options: { sourceMap: true}

                    }

                ]

            },

**yarn add -D sass-loader node-sass**

укажем что-нибудь в src/main.scss:

.wrapper {

    background: bisque;

    h1 {

        color: blue;

    }

}

подключаем scss в index.js

import './scss/main.scss';

**yarn build**

Далее разберемся с postCss плагинами - autoprefixer: <https://github.com/postcss/autoprefixer>

и для postCss плагинов еще потребуется обработчик postcss

Для того чтобы было удобно подключать PostCss плагины созадим отдельный конфиг для postCss например в src/js/postcss.config.js

module.exports = {

    plugins: [

        require('autoprefixer'),

        require('css-mqpacker'), // сжимает все media query запросы в один файл

        require('cssnano')({ // минификация

            preset: [

                'default', {

                    discardComments: {

                        removeAll: true

                    }

                }

            ]

        })

    ]

}

далее настроим autoprefixer через package.json, если мы не знаем что именно писать, то можно взять настройки от автора:

. . . .

"browserslist": [

    "> 1%",

    "last 3 version"

  ],

  "devDependencies": {

. . . .

Теперь установим все эти плагины в package.json

**yarn add -D postcss-loader autoprefixer css-mqpacker cssnano**

затем передаем postcss-loader в scss и css обработчики:

{

                test: /\.scss$/,

                use: [

                    'style-loader',

                    MiniCssExtractPlugin.loader,

                    {

                        loader: 'css-loader',

                        options: { sourceMap: true}

                    }, {

                        loader: 'postcss-loader',

                        options: { sourceMap: true, config: {path: 'src/js/postcss.config.js'}}

                    }, {

                        loader: 'sass-loader',

                        options: { sourceMap: true}

                    }

                ]

            },

            {

                test: /\.css$/,

                use: [

                    'style-loader',

                    MiniCssExtractPlugin.loader,

                    {

                        loader: 'css-loader',

                        options: { sourceMap: true}

                    }, {

                        loader: 'postcss-loader',

                        options: { sourceMap: true, config: {path: 'src/js/postcss.config.js'}}

                    }

                ]

            }

**yarn build**

Одна из особенностей css-nano это то как он сжимает повторяющийся css код, такого рода код

h1 {

    color: blue;

}

h2 {

    color: blue;

}

h3 {

    color: blue;

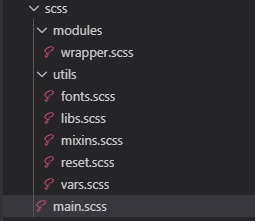
}

будет минифицирован в:

h1,h2,h3{color:#00f}

Не все минификатора умеют так.

Далее напишем общий каскад для нашего scss, для этого реструктурируем src/scss/ и добавим туда файлы:



в большинстве случаев css и sass библиотеки подключаются через src/scss/utils/libs.scss

// Sass librarys example:

@import '../../node\_modules/spinners/stylesheets/spinners';

// CSS librarys example:

@import '../../node\_modules/flickity/dist/flickity.css';

который уже в свою очередь импортируется в src/scss/main.scss

**@import "utils/libs";**

@import "utils/vars";

@import "utils/mixins";

@import "utils/fonts";

@import "utils/reset";

и далее index.js.

import './scss/main.scss';

index.js вообще не рекомендуется засорять.

Далее рассмотрим vars.scss - @import "utils/vars";

// Font

$mainFont                : 'Montserrat', Helvetica, Arial, sans-serif;

// Size

$mainFontColor           : #101010;

$mainFontSize            : 18px;

$mainFontWeight          : 400;

$mainLineHeight          : 1.4;

$desktopWidth            : 1280px;

$smDesktopWidth          : 980px;

$tableWidth              : 768px;

$phoneWidth              : 480px;

$smPhoneWidth            : 320px;

// Main color

$default-color           : #ffffff;

$primary-color           : #444ce0;

$success-color           : #26de81;

$danger-color            : #fc5c65;

$warning-color           : #fed330;

$light-color             : #999999;

$purple-color            : #8854d0;

// Neutral Color

$neutral-primary         : #303133;

$neutral-regular         : #606266;

$neutral-secondary       : #909399;

$neutral-placeholder     : #C0C4CC;

// Border Color

$border-base             : #DCDFE6;

$border-light            : #E4E7ED;

$border-lighter          : #EBEEF5;

$border-lightex          : #F2F6FC;

$success-color: #26de81 - такое название лучше чем например $green-color.

Затем идет адаптив - @import "utils/mixins"; - очень крутая штука

Особое внимание заслуживает mixin - @function rem(){}:

**Надо получше разобрать, еще не до конца вник**

магия, адаптивный размер шрифта: <https://youtu.be/qqTIqwQX8nc?list=PLkCrmfIT6LBQWN02hNj6r1daz7965GxsV&t=1387>

мы можем писать сколько угодно @madia запросов, css-mqpacker сгруппирует и оптимизирует их всех в одно место в app.css

Такой стиль написания характерен для stylus, но здесь он также применим, но не обязателен.

Содержание файлов см. в моем репозитории: здесь добьавить ссылку на ГИТ

Использование webpack для верстки. Подключение Pug

Использовать webpack для верстки многостраничных сайтов легко.  
В текущей сборке уже готово:

* Подключен babel7 - js код автоматический перегонится в старый стандарт и будет работать во всех браузерах
* Весь js код автоматический поджимается
* Подключен SASS (scss) с удобной настройкой под себя
* По желанию можно использовать обычный css
* autoprefixer - префиксы расставятся автоматический, css код будет валидным во всех браузерах
* Все стили будут максимально поджаты с cssnano и mq-packer
* Настроен livereload - авто обновление страницы при любых изменениях в js, css, html
* Авто сплиты для js и css файлов
* Подключен vue.js с удобной интеграцией шаблонов напрямубю в хтмл
* Автоматический или ручной подхват html страниц
* Константы для удобной конфигурации шаблона под себя

## Webpack для верстки

Первый вариант - без использования шаблонизатора pug:

Первый шаг скачать текущую сборку - [ссылка на прямое скачивание](https://github.com/vedees/webpack-template/archive/master.zip) или [репозиторий на GitHub](https://github.com/vedees/webpack-template)

Далее следовать простым командам:

// Установить все зависимости

npm install

// Запустить сервер для дев разработки (http://localhost:8081/)

npm run dev

// Команда для деплоя (папка на выходе будет как dist)

npm run build

Более подробную инструкцию можно найти в README md - ссылка [здесь](https://github.com/vedees/webpack-template)

Исходный код: <https://github.com/vedees/webpack-template>

## Webpack 4 Подключение и настройка pug и pug-loader:

Для устновки pug и pug-loader в терминале:

npm i pug pug-loader --save-dev

Далее нужно добавить обработчик для pug файлов:

{

test: /\.pug$/,

loader: 'pug-loader'

}

Обработка pug страниц:

Первое, что нужно сделать это указать, что будем обрабатывать и откуда брать pug файлы:

const PAGES\_DIR = `${PATHS.src}/pug/pages/`

const PAGES = fs.readdirSync(PAGES\_DIR).filter(fileName => fileName.endsWith('.pug'))

Далее есть два способа обработки. Первый способ - автоматическое добавление страниц:

...PAGES.map(page => new HtmlWebpackPlugin({

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

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

}))

Второй способ - ручной (можно легко связывать с первым способом) :

new HtmlWebpackPlugin({

template: `${PAGES\_DIR}/about/index.pug`,

filename: './about/index.html',

inject: true

}),

new HtmlWebpackPlugin({

template: `${PAGES\_DIR}/about/portfolio.pug`,

filename: './about/portfolio.html',

inject: true

})

В пример привел страницы индекс и абаут из директории about.

Теперь осталось создать страницы для пага, а также главные лаяуты с инклюдами.  
Все ваши вопросы есть в ридми, но если вы не найдете свой вопрос то можете задать его здесь или на ютубе!

Весь исходный код: - <https://github.com/vedees/webpack-template-pug>

vdfvd

csac