**Webpack Youtube Series**

<https://www.youtube.com/watch?v=GU-2T7k9NfI&list=PL55RiY5tL51rcCnrOrZixuOsZhAHHy6os>

1. **Lesson 1 How does it work?**

Webpack will do the following:

* Minify the working files, css, js, html
* Bundle all of the necessary files
* Preprocess the files (Babel, SCSS)

1. **Start package.json**

Npm init

1. **Install webpack**

Npm install webpack –save-dev

1. **Write a script in the package.json file which will run webpack**

{

"name": "wepback\_01",

"version": "1.0.0",

"description": "This repository holds the source code of my Webpack 2 Basics Series on YouTube. Make sure to watch the Videos on YouTube to understand how to use that code.",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

**"build": "webpack --entry ./src/js/app.js --output-filename ./bundle.js -p",**

},

"author": "",

"license": "ISC",

"devDependencies": {

"webpack": "^4.0.1"

}

}

**" "build": "webpack --entry ./src/js/app.js --output-filename ./bundle.js -p",**

**Webpack:** **will run wepback**

**--entry src/js/app.js: This is where webpack will start the work**

**--output-filename dist/bundle.js: This is where webpack will bundle all of the files**

1. **Import other JS files**

**App.js**

import {secretParagraph, secretButton} from "./dom-loader";

This is **ES6 syntax**, which will be **recognized by wepback**, (**Object destruction**)

In order to make it work, **you need to export** the necessary elements from **dom-loader**

export var secretButton = document.querySelector('#secret-button');

export var secretParagraph = document.querySelector('#secret-paragraph');

1. **Minify the bundle for prod**

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"build": "webpack --entry ./src/js/app.js --output-filename ./bundle.js -p",

"build:prod": "webpack --entry ./src/js/app.js --output-filename ./bundle.js -p"

},

Create a script tag with **build:prod** and run wepback + minify it for production with **–p.**

* 1. **Modify the HTML <script> tag**

Now you do not have to include all of the other js files separately, just 1 bundle!

<script src="**dist/bundle.js**"></script> -->

<!-- <script src="src/js/dom-loader.js"></script>

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

1. Before you run it, you should install **webpack-cli**

npm install webpack-cli -D

**Lesson 2 Wepback DevServer**

In order to recreate the real life scenario, instead of using the chrome file protocol, we should create a simple http server and serve the static files through that at a certain port.

1. **Install built in webpack-dev-server**  so our files will be served with http protocol

npm install webpack-dev-server

1. **Add**  the **dev-server** to the package.json

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"build": "webpack-dev-server --entry ./src/js/app.js --output-filename ./dist/bundle.js",

"build:prod": "webpack --entry ./src/js/app.js --output-filename ./bundle.js -p"

},

1. **Install wepback-cli**

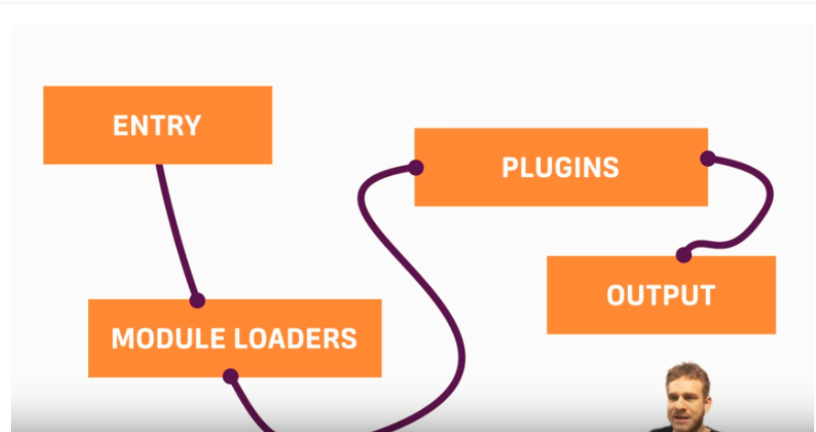
Npm install webpack-cli –D

1. **Run the following**

**npm run build** -> it will start a http-server for you at localhost, serving all of the static data

**npm run build:prod** -> it will make the bundle + minify the files

**Lesson 3 Webpack Core Concepts**



**Entry:** Where webpack starts it’s journey

**Module loaders:** What loader should be used with what file extensions

**Plugin:**  loaders on a completely different level

**Output:**  Where the bundle(s) will be created.

In order to import css files into js files, you need a module for it (**css-loader**)

In order to apply the imported css code from the javascript, you have to use (**style-loader**)

1. **Create a config wile for webpack as -> webpack.config.js**

Output:

**path:** -> absolute path where the files should be bundled

**filename:** the name of the bundle

p**ublicPath:** Tell webpack-dev-server, where the assets will be stored

Module:

**Rules:**  array of test, which will determine which loader(s) should be used

**Test:** regular expression, which will check the filename

**Use:** array of loaders to use, usage is in reversed order

**Loader:** 1 loader to use

**Plugins:**  extra loaders

//Built in nodejs package, which will difen the path for the current file

var path = require('path'); //to determine the absolute path

module.exports = {

// Where should be start the journey

// relative path!!

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

output:{

//Where to store the bundle

//Absolute path, wepback needs to create files

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

//outputted filename

filename:"bundle.js",

//Tell webpack-dev-server, where the assets will be stored

publicPath:"/dist"

},

//How wepback should treat the different modules/imported files

module:{

rules:[

{

//checking the filenames with regular expression

test:/\.css/,

// loader:"css-loader" 1 loader

//you can determine multiple loader, order is imporant!

//Wepback will execute the loaders in reversed order

use:[

'style-loader', //2nd

"css-loader" //1st

]

}

]

},

//This is where you could put your plugins

// plugins:[

// new webpack.optimize.UglifyJsPlugin({

// //... options

// })

// ]

};

1. **Modify the package.json, to implement wepback remove any extra configuration**

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

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

"build:prod": "webpack -p"

},

1. **Bundle css Files**
   1. **import them into the app.js (**this will be done by **css-loader** module**)**

import "../css/main.css";

import "../css/input-elements.css";

**3.2 create style from the imported css in js (style-loader)**

* 1. **remove css links from index.html**

<!-- <link rel="stylesheet" href="src/css/main.css">

<link rel="stylesheet" href="src/css/input-elements.css"> -->

**Lesson 4 Babel + SCSS**

1. **Importing all of the necessary modules**

**Npm init –y** -> will answer all of the questions with “yes”

**npm install webpack webpack-dev-server –save-dev**

**sass-loader:** will understand the scss code

**node-sass:**  will do the translation from scss to css

**css-loader:** to handle the compiled css in javascript

**(style-loader**: to inject the css into the header of the html )

**extract-text-webpack-plugin:** create separate css files to be imported

**babel-core**: transpile the es6 to es5

**babel-loader**: will connect the transpiled code with webpack

**babel-preset-es2015:** will transpile to es2015 version all of the code

**webpack-cli:**  to execute webpack

1. Import the scss files and other js files into the **app.js**

import "../css/main.scss"; //importing the scss file into the js file

import {RandomGenerator} from "./random-generator";

1. Import other scss files into the **main.scss**

/\*this is scss code, it will import \_colors.scss\*/

@import"colors";

3. Write the **webpack.config.json**

**ExtractTextPlugin:**  it is the plugin, which will make seperate css files from the compiled scss

We need to instantiate this object and set the **filename** of the bundled css in the “**dist**“folder

**Entry:** where wepback should start it’s journey

**Output** is where the bundle.js will be generated, where can the webpack-dev-server can serve them

**Configure a loader:** you can configure a loader, by passing an object with the specified configuration

use:[ //configure the options of the loader

{

loader:"babel-loader",

**options:{**

**presets:["es2015"]**

**}**

}

]

**Configure plugin:** we use the instatiated plugin, pass the loaders to be used into on of it’s function, so this is the congifuration

use: extractPlugin.extract({

use:['css-loader',"sass-loader"]

})

Then we are executing the plugin, so it will generate separate css files.

var path = require('path');//Import our plugin, which will make seperate css files from the transpiled code

var ExtractTextPlugin = require("extract-text-webpack-plugin");

//We need to instantiate the object and determine it's css name

var extractPlugin = new ExtractTextPlugin({

filename:"main.css" //will be placed in the /dist

});

module.exports = {

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

output:{

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

filename:"bundle.js",

publicPath:"/dist"

},

module:{

rules:[

{

test:/\.js$/,

use:[ //configure the options of the loader

{

loader:"babel-loader",

options:{

presets:["es2015"]

}

}

]

},

{

test:/\.scss/,

use: extractPlugin.extract({

use:['css-loader',"sass-loader"]

})

}

]

},

//Will execute our plugin

plugins:[

extractPlugin

]

};

1. **Insert link + script tag to the main HTML**

<link rel="stylesheet" type="text/css" href="dist/main.css">

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

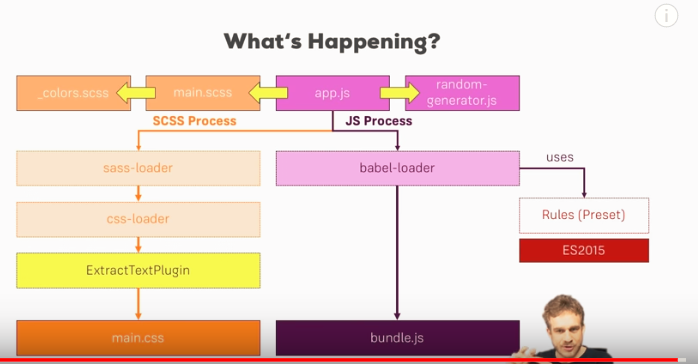
**This is how the wepback workflow is executed and the main.css + bundle.js are generated.**

Please note that a newer version of the “**extract-text-webpack-plugin**” has to be used!!

<https://github.com/webpack-contrib/extract-text-webpack-plugin/releases/tag/v4.0.0-alpha.0>

run in the node

**npm i -D extract-text-webpack-plugin@next**



**Lesson 5 HTML + IMAGE loaders**

**(webpack.config)**

**npm install –save-dev**

**html-loader:** to load all of the html to webpack

**html-webpack-plugin**: will recreate the html files in a separate directory

**file-loader:**  to copy images, other files

**clean-webpack-plugin:** will clear the dist folder upon a new wepback build:prod command

Installing

<https://github.com/mschwarzmueller/yt-webpack2-basics/tree/06-webpack-babel-scss-img-html>

Since due to newer version, a lot of code have been changed this is the working one!

Wepback.config.json is commented, main catch:

file-loader is really tricky and you have to use the exact same method as here!

**Lesson 6 Multiple HTML Files**

1. **Use file-loader to copy all of the html files to the source**

**Webpack.config.json**

{

test: /\.html$/,

use: [

{

loader: 'file-loader',

options: {

name: '[name].[ext]',

}

}

],

//We do not want to copy over base index.html

exclude: path.resolve(\_\_dirname,"src/index.html")

}

1. Just include the html files in the **app.js**

import '../users.html';

**Lesson 7 3rd Party Packages**

1. Npm install jquery –save
2. Include the installed jquery in the **app.js**

import'jquery';

1. Use Wepback Built in object to declare at which sign, what package should be included
2. var webpack = require('webpack');
3. plugins: [
4. //We map the imported js files together
5. new webpack.ProvidePlugin({
6. $:'jquery',
7. jQuery:'jquery'
8. }),
9. ]