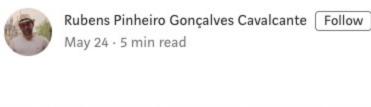
# Webpack From Zero to Hero

Chapter 3: Everything is a Module



Foam Blocks - Picture under the Creative Commons License (Source: Flickr)

Introduction

ight now we achieved a really simple application using Webpack and

ES2015+ transpilation, but real apps are more than that, they have CSS, Images, custom Fonts and all sorts of different assets. The good thing

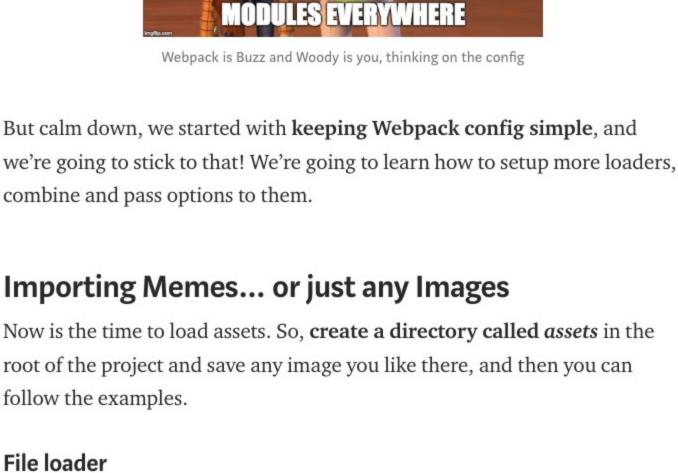
is, in your code Webpack will treat then equally as ES Modules, as long you

### have the proper loader to them.

importing it:

complain:

MODULES



The **file-loader** treats your image as any asset and gives it a url path when

### 2 document.body.innerHTML = '<div id="myMemes"></div>'; document.getElementById("myMemes").innerHTML = ` <h1>And his name is...</h1> <img src="\${johnCena}" />

view raw index.js hosted with \ by GitHub If you run this without installing the file loader first though, Webpack will

```
ERROR in ./src/assets/unexpected.jpg 1:0
Module parse failed: Unexpected character '* (1:0)
You may need an appropriate loader to handle this file type.
(Source code omitted for this binary file)
@ ./src/index.js 1:0-45 3:104-114
@ multi (webpack)-dev-server/client?http://localhost:8080 ./src
            Webpack even gives a tip about using an appropriate loader 🔍
```

yarn add file-loader --dev Append this snippet to your webpack.config.js in the module.rules section:

```
test: /\.(gif|png|jpe?g|svg)$/i,
          use: 'file-loader'
      ]
 9 },
                                                                              view raw
 webpack-rules.js hosted with \ by GitHub
Translating the regex: we are searching for any file ending with .gif, .png,
.jpg/.jpeg or .svg, all being case insensitive (the `/i` flag in the end).
```

Now if you run it again, webpack will successfully execute the build.

Using *image-webpack-loader* + *file-loader* will provide us with some nice

optimization to our image files. It's also easy to add:

yarn add image-webpack-loader --dev

And add the rule after the file-loader one:

test: /\.(gif|png|jpe?g|svg)\$/i,

mozipeg — Compress JPEG images

optipng — Compress PNG images

<u>pngquant</u> — Compress PNG images

<u>svgo</u> — Compress SVG images

gifsicle — Compress GIF images

'file-loader',

Image Webpack Loader

loader: 'image-webpack-loader', options: { disable: true // Disables on development mode

} } 10 11 ] 12 } view raw webpack-file-loader.js hosted with \ by GitHub

```
Media (Audio and Video)
Similar to images, we can set a rule for video and audio to be processed by
the file-loader.
Just add the rule to the webpack.config.js in the config.rules section:
    test: /\.(ogg|mp3|wav|mpe?g)$/i,
     use: 'file-loader'
                                                                         view raw
 media-config.js hosted with \ by GitHub
```

## import andHisNameIs from "./assets/and-his-name-is.mp3";

const audio = new Audio(andHisNameIs);

Adding Some Style with Sass

audio-snippet.js hosted with \ by GitHub

loaders and the sass parser:

<style></style> tag

Adding this to the config:

test: /\.scss\$/,

sass-rule.js hosted with \ by GitHub

text-align: center;

module: { rules: [ //...

];

#myMemes {

9 }

const img = document.querySelector("#myMemes img");

img.addEventListener("click", event => audio.play());

will behave the same, providing the source url for you):

loader and it will probably break when reading an audio/video file.

Now you can import audio files, same as you do with images (file-loader

Let's play with that and add the following to the end of the index.js file:

You can use any audio file, I'm using this one, but beware, the audio is quite loud 4. Don't play it at work!

view raw

"Haha, so funny... now let's go back please? 🤡 "

Let's add Sass support to our app! Same as before, we need the proper

margin: auto auto; 3 4 5 h1 { font-family: Arial, Helvetica, sans-serif; 6 7 color: #333; 8 } 9 10 img { 11 border-radius: 16px; 12 padding: 8px; 13 border: 1px solid #aaa; 14 box-shadow: 2px 2px 1px #eee; cursor: pointer; 15 16 } 17 } view raw style.scss hosted with | by GitHub Now add an import in index.js file: import johnCena from "./assets/unexpected.jpg"; import "./style.scss"; 4 //.... view raw index.js hosted with \ by GitHub

```
loader: "image-webpack-loader",
13
            options: {
              disable: true // webpack@2.x and newer
15
17
          }
18
        ]
      },
19
21
      test: /\.(ogg|mp3|wav|mpe?g)$/i,
        use: "file-loader"
23
24
25
       test: /\.scss$/,
          argv.mode === "production" ? MiniCssExtractPlugin.loader : "style-loader",
27
          "css-loader",
```

view raw

view raw

This article is part of the Webpack from Zero to Hero series, for more background or for the index you can check the Chapter 0: History. Previous - Chapter 2: Tidying Up Webpack Next - Chapter 4: Dynamic Imports and Code Splitting (coming soon)

Let's install it then and add to our configuration file:

import johnCena from "./assets/unexpected.jpg";

000

module: { rules: [ //... {

It has some image compressors already enabled by default:

```
"Wait there's already a rule for file-loader, how this is going to work?"
That's true, but we can add many config rules for different kinds of files.
We're just not adding the media files together with the rule that matches
images because, on the last section, we chained it with the webpack-image-
```

yarn add node-sass sass-loader css-loader style-loader --dev 🙀 — "Whawhawhaaaat? Why so many dependencies?" Each one of this dependencies has a role to play: sass-loader + node-sass: receive your Sass files and output CSS • css-loader: turns it into a JS module

style-loader: gets the CSS module and inserts it into the page inside a

use: ["style-loader", "css-loader", "sass-loader"] // It's like a pipeline

Now let's play around. Create a style.scss file and paste this code into it:

view raw

```
Production Styles
```

Each import you do will create a new <style></style> entry in the header

of the file. This can quickly go out of control, so you may want just to join all

the files together, minify and provide as a single optimized file. For this we

Let's add it to our production builds, in the webpack.config.js, on top of it

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

Then add the loader to the *scss* rule, but only for mode **production**,

otherwise let's stay with the style-loader (and let's take this opportunity to

have a solution, which is the mini-css-extract-plugin.

yarn add mini-css-extract-plugin --dev

To install is simple:

add the require call:

add **source-map** support too):

argv.mode === "production"

sourceMap: true

sourceMap: true

: "style-loader",

? MiniCssExtractPlugin.loader

loader: "css-loader", options: {

loader: "sass-loader", options: {

test: /\.scss\$/,

use: [

}

}

Hash: 9de6ba97ca87a9372ad6 Version: webpack 4.29.0

+ 1 hidden module

Done in 4.10s.

4 5

7 8

29

31

]

} 1;

},

Built at: 2019-01-23 15:25:04

[2] ./src/index.js 485 bytes (0) [built [3] ./src/style.scss 39 bytes (0) [built]

Entrypoint mini-css-extract-plugin = \*

loader/lib/loader.js!src/style.scss:

+ 1 hidden module

Asset Size Chunks Chunk

77b9f570b83e36d270c0819a173b99e2.jpg 16.6 KiB [emitted]
a93ba35eae6a143ccce10a72c71tb67d.mp3 112 KiB [emitted]
main.css.map 513 bytes 0 [emitted] main
main.js map 1.44 KiB 0 [emitted] main
main.js.map 5.92 KiB 0 [emitted] main

Child mini-css-extract-plugin .,/../node\_modules/css-loader/dist/cjs.js!../../node\_modules/sass-

[0] /webpack-workshop/node\_modules/css-loader/dist/cjs.js!/webpack-workshop/node\_modules/sass-

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

You can see it generates with the same name as the entry/chunk (main.css)

As you see, the config already started to grow. But the advantage is, it's a

Entrypoint main = main.css main.js main.css.map main.js.map [0] ./src/assets/and-his-name-is.mp3 82 bytes (0) [built] [1] ./src/assets/unknown.jpg 82 bytes {0} [built]

loader/lib/loader.js!./src/style scss 440 bytes {0} [built]

Keeping the Config Clean

Node.js file and we can split it into functions!

},

3

7

9

10

11

13

15

```
}
        ]
 17
                                                                                    view raw
 mincss-rule.js hosted with \ by GitHub
And finally, we do the same for the plugins section:
      plugins: [
      // Any option given to Webpack client can be captured on the "argv"
      argv.mode === "development" ? new HtmlWebpackPlugin() : null,
       argv.mode === "production"
       ? new MiniCssExtractPlugin({
            filename: "[name].css",
              chunkFilename: "[id].css"
          : null
 10 l.filter(
       // To remove any possibility of "null" values inside the plugins array, we filter it
 12
       plugin => !!plugin
 13 );
                                                                                    view raw
 mincss-plugins.js hosted with \( \psi \) by GitHub
Running the build, we'll see the style bundle being output:
```

9 test: /\.(gif|png|jpe?g|svg)\$/i, 10 "file-loader", 11

"sass-loader"

module.rules.js hosted with \ by GitHub

module.exports = (env, argv) => [

test: /\.js\$/,

use: "babel-loader"

Webpack Technology Nodejs Frontend 401 claps

And require it in webpack.config.js: module: { rules: require("./webpack/module.rules")(env, argv) 3 }, require-module.rules.js hosted with \ by GitHub Now you can see that is way cleaner, and you know where the module rules are because the filename follows the same name of the config schema. Remember what we learned in Chapter 2, a clean config sparks joy \*\*! asically Webpack loaders and plugins mostly follow the format we saw in this chapter. There are many loaders out there, some from the Webpack core team, some created by the open source community , and even one that's developed by me, webpack-chrome-extension-reloader (check it out 6).

Let's end here and go deeper in the next chapter. We will learn not only how to setup dynamic imports on our application, but play around with all Webpack code splitting strategies. See you in the next chapter! JavaScript