

This is how angular-cli/webpack delivers your CSS styles to the client

When you generate a new Angular project and run `ng new myapp` you end up with the following project structure:

```
src
app
assets
index.html
styles.css
...
```

What interests us here is the `styles.css` file which is supposed to be used to declare global styles. You can write the styles in this file directly or import using [CSS specific imports](#). If you put the following in the `index.html`:

```
<body>
<header>
<span>I am header span</span>
</header>
<span>I am regular span outside of header</span>
</body>
```

And add the following styles into the `styles.css`

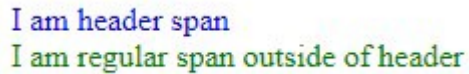
```
@import "header.css";

span {
color: green;
}
```

and to the `header.css`:

```
header span {  
  color: blue;  
}
```

You will see the following presentation when running the application:



I am header span
I am regular span outside of header

This is expected but the real question is how does Angular-CLI does it? Actually the styles are packaged and delivered to the client by webpack. Angular-CLI uses webpack under the hood and only configures it.

Setting up

First, let's create a new project with the following structure:

```
app  
header.css  
styles.css  
index.html
```

We will be working only with CSS in the project so we don't need any JavaScript. The `app` folder houses a `css` file for the header. We will import it inside the main `style.css` using CSS imports.

The `index.html` will have the following markup:

```
<head>  
<meta charset="UTF-8">  
<title>Title</title>  
</head>  
<body>  
<header>  
<span>I am header span</span>  
</header>  
<span>I am regular span outside of header</span>  
</body>
```

Here we have have one span inside the header and at the top level of the body. We want to see blue text inside the header and green text inside the regular span.

To achieve that we put the header specific styles inside `app/header.css`:

```
header span {  
color: blue;  
}
```

and the general styles into `styles.css`:

```
@import "app/header.css";
```

```
span {  
color: green;  
}
```

We've got that and now we need to deliver that into the client using webpack. Let's start by installing webpack:

```
npm i webpack --save-dev
```

Angular-CLI loads these styles as a separate bundle to the client. So we will do the same. Webpack creates bundles based on the [entry points](#). So let's add styles specific entry point to the webpack configuration.

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

```
module.exports = {  
  entry: {  
    styles: './styles.css'  
  }  
};
```

Also we need to tell where the [output](#) should go. Let's add this information:

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

```
module.exports = {  
  entry: {  
    styles: './styles.css'  
  },  
  output: {
```

```
path: path.resolve(__dirname, 'dist'),
filename: "[name].js"
}
};
```

Now we have our basic config. If you now run webpack we will get the error:

Module parse failed: D:\medium\styles.css\styles.css... You may need an appropriate loader to handle this file type.

Every file/module we want to be used in the bundle webpack expects to be a valid JavaScript module. And certainly `styles.css` is not a valid JavaScript module. So we need something to turn this CSS module into JS module. And this is where [loaders](#) come in. Here is what webpack docs say about loaders:

Loaders are transformations that are applied on the source code of a module. They allow you to pre-process files as you `import` or “load” them... Loaders can transform files **from a different language (like TypeScript) to JavaScript**, or inline images as data URLs.

CSS loader

Most loaders and plugins come as separate packages so let's install it:

```
npm install --save-dev css-loader
```

After we've got it installed we need to add it to our configuration:

```
module.exports = {
  entry: ...,
  output: ...,
  module: {
    rules: [
      {
        test: /\.css$/,
        use: ['css-loader']
      }
    ]
  }
}
```

```
]
}
```

We pass `/\.css$/` regexp to the `test` property. Webpack applies that regexp to each request/file and if there's a match the loaders from the `use` array are applied to the contents of the file. This regexp matches all files that end with `.css`.

Cool, let's run `webpack` now and see what we get:

```
$ webpack
Hash: 5e39e00b22e7d8dbb305
Version: webpack 3.4.1
Time: 698ms
Asset Size Chunks Chunk Names
styles.js 5.25 kB 0 [emitted] styles
[1] ./styles.css 272 bytes {0} [built]
[2] ./node_modules/css-loader!./app/header.css 200 bytes {0} [built]
+ 1 hidden module
```

It basically says here that it generated `styles.js` bundle in 698 ms. This bundle includes `styles.css` and `/app/header.css` modules plus one module from the `node_modules`. Webpack reports all modules that come from `["node_modules", "bower_components", "jam", "components"]` as hidden. You can actually see what this module is if you run:

```
webpack --display-modules
```

And you can see that it's a module supplied by the `css-loader`:

```
$ webpack --display-modules
Hash: 1229210b090997ed5ae2
Version: webpack 3.4.1
Time: 417ms
Asset Size Chunks Chunk Names
styles.js 5.26 kB 0 [emitted] styles
[0] ./node_modules/css-loader/lib/css-base.js 2.26 kB {0} [built]
[1] ./styles.css 274 bytes {0} [built]
[2] ./node_modules/css-loader!./app/header.css 201 bytes {0} [built]
```

So now inside our `dist` folder webpack generated the `styles.js` bundle. Let's reference this file in the `index.html` and see if we get green color for the top level span and blue color for the header span:

```
<head>
<meta charset="UTF-8">
<title>Title</title>
<script src="dist/styles.js"></script>
</head>
<body>
<header>I am header</header>
<span>I am regular span outside of header</span>
</body>
```

And...

```
I am header span
I am regular span outside of header
```

Nope, nothing. The color is still black.

Okay, let's see what's inside the generated `styles.js` bundle. Here is the gist of it:

```
/* 1 */
(function (module, exports, __webpack_require__) {
  exports = module.exports = __webpack_require__(0)(undefined);
  exports.i(__webpack_require__(2), "");
  exports.push([module.i, "span {\r\n color: green;\r\n}", ""]);
}),
/* 2 */
(function (module, exports, __webpack_require__) {
  exports = module.exports = __webpack_require__(0)(undefined);
  exports.push([module.i, "header span {\r\n color: blue;\r\n}", ""]);
})
```

Basically here we see that each module just exports styles as a string. So after running these two modules we end with the following array:

```
exportedStyles = [
  // specified in the styles.css
  "span {\r\n color: green;\r\n}",
```

```
// imported from the app/header.css
"header span {\r\n color: blue;\r\n}"
];
```

But actually they are not used anywhere. We need something that will use this array of strings and actually add to the `style` tag on the page so we end up with something like this:

```
<head>
<meta charset="UTF-8">
<title>Title</title>
<script src="dist/styles.js"></script>
<style>
span { color: green; }
header span { color: blue; }
</style>
```

And as you've probably guessed there's a loader that can do that.

Style loader

Again, this one comes a separate package and need to be installed:

```
npm install style-loader --save-dev
```

Similarly to the `css-loader` we need to add this loader to the webpack configuration:

```
module.exports = {
  entry: ...,
  output: ...,
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          "style-loader",
          "css-loader"
        ]
      }
    ]
  }
}
```

```

]
}

```

You can see we added `style-loader` before the `css-loader` in the configuration. Webpack applies all loaders in the reverse order so `css-loader` will be applied first followed by the `style-loader`. This is exactly what we need. The `css-loader` will generate a JS module that exports styles and `style-loader` will use them to add to the `<style>` tag in the html.

Cool, let's run webpack now and see what we get:

Hash: 9fe17c2f175614de78ea

Version: webpack 3.4.1

Time: 556ms

Asset Size Chunks Chunk Names

styles.js 18.1 kB 0 [emitted] styles

[1] ./styles.css 999 bytes {0} [built]

[2] ./node_modules/css-loader!./styles.css 274 bytes {0} [built]

[3] ./node_modules/css-loader!./app/header.css 201 bytes {0} [built]

+ 3 hidden modules

No errors and built is ready. Cool. If we now run the `index.html` we will get the desired presentation:

```

I am header span
I am regular span outside of header

```

HTML Webpack Plugin

Here is what the docs say about it:

...simplifies creation of HTML files to serve your webpack bundles. This is especially useful for webpack bundles that include a hash in the filename which changes every compilation. You can either let the plugin generate an HTML file for you, supply your own template using lodash templates or use your own loader.

Okay, let's see this thing in action. Install it:


```
$ npm install --save-dev html-webpack-plugin
```

By default this plugin generates its own `index.html` file. But since we already have our own `index.html` we can use `template` option to specify the path to the existing `index.html`:

```
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: {...},
  output: {...},
  module: {...},
  plugins: [
    new HtmlWebpackPlugin({
      template: './index.html'
    })
  ]
};
```

Run the build and here is contents of the resulting `index.html` this is now generated inside the `dist` folder by the plugin:

```
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<header>
<span>I am header span</span>
</header>
<span>I am regular span outside of header</span>
<script type="text/javascript" src="styles.js"></script>
</body>
```

Full configuration

Here is the complete configuration of the `webpack.config.js` used in the project:

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: {
    styles: './styles.css'
  },
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].js'
  },
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          'style-loader',
          'css-loader'
        ]
      }
    ]
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: './index.html'
    })
  ]
};
```

Angular-cli

On top of what we've configured above Angular-CLI adds loaders for the files generated by the SASS, LESS and STYLUS preprocessors. It is done

using respective loaders. The CLI also applies a number of `postcss` plugins like [postcss-url](#), [autoprefixer](#) and [cssnano](#) through the `postcss-loader`:

```
const postcssPlugins = function () { .... };  
  
...  
  
{  
  "loader": "postcss-loader",  
  "options": {  
    "plugins": postcssPlugins  
  }  
},
```

GitHub

Thanks for reading! If you liked this article, hit that clap button below 🖐️. It means a lot to me and it helps other people see the story. For more advanced articles you can follow me on [Twitter](#) or on [Medium](#).

Viewed using [Just Read](#)