Docker with WebPack to build a simple project.

With a Babel, ESLint, PostCSS and StyleLint configuration guide

Recently, I reminded myself to learn some new technologies apart from my daily work and a friend suggested me to try from the ground up, mostly to teach on how to create a functional site with many different technologies and to make it easy for newcomers. The first part of this work starts with this tutorial, but roughly speaking, it is not solely my work, because tutorials sharing by some of you let me start and share my little part.

When I begin the process of making this tutorial I found some interesting others which helping me a lot to make this one.

Setup an environment for the project

Using docker to isolate things

To start the work we need a fresh environment, and in my case, I need to avoid impact over others projects and the environment I have set up for my work so I download a docker Linux image, but Windows/Mac/Others are valid options, to install it.

When **docker** is installed the next step is search an image and install it to start our develop. I use a simple **Ubuntu** image and not a **docker file** or a **NodeJs docker image** to construct it because I want to show you how to do it, but if you prefer (and you know how to do) you can use it as well and go directly to the following step. The step to download image, create a container and use it are:

```
docker pull ubuntu
mkdir -p -v ~/projects/demo
docker run -t -d -p 3000:3000 --name ubuntu \
           -v ~/projects/demo:/opt/apps ubuntu:latest
docker start ubuntu
docker exec -ti ubuntu bin/bash
```

> *Note that we are connecting the host ports 3000 (left part) with the container ports (right part), because later we need them to test our app in browser. Be aware of the sharing folder route (-v parameter) between host and docker container folder because the project files will be there.*

Installing NodeJS

```
apt-get update
apt-get install curl
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
```

Then close and reopen your terminal (or stop and start your docker image, use ctrl+d or type logout) to start using nvm

```
nvm --version
0.35.3
nvm install node
Downloading and installing node v14.9.0...
Downloading https://nodejs.org/dist/v14.9.0/node-v14.9.0-linux-x64.tar.xz...
################################################################################
################################################################################
## 100.0%
Computing checksum with sha256sum
Checksums matched!
Now using node v14.9.0 (npm v6.14.8)
Creating default alias: default -> node (-> v14.9.0)
node --version
v14.9.0
npm --version
6.14.8
```

Create and initial project

Now you have a real environment to work is time to put framework on site and start working to get an initial result.

Create the initial project folders inside the folder **/opt/apps** and run "**npm init -y**" to create the project and the **package.json** file, the you get the following structure as the starting point

```
demo/
    src/
    package.json
```

```
package.json
{
  "name": "demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "MIT"
}
```

Configuring WebPack, Babel and ESLint

Webpack is a *static module bundler* for modern JavaScript applications, but its not limited to Javascript only it can be extended to process CSS, HTML, any other resources if you have the right plugin.

Babel is a transpiler, a tool that translate our actual javascript to support the target browser/node server javascript version.

ESLint check inconsistencies in code helping us with rules and to enforce the same code style across all code base.

```
npm i webpack webpack-cli webpack-dev-server html-webpack-plugin \
   @babel/core babel-loader @babel/preset-env @babel/preset-react \
   --save-dev
```

package.json
```
{
  "name": "demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@babel/core": "^7.11.4",
    "@babel/preset-env": "^7.11.0",
    "@babel/preset-react": "^7.10.4",
    "babel-loader": "^8.1.0",
    "html-webpack-plugin": "^4.3.0",
    "webpack": "^4.44.1",
    "webpack-cli": "^3.3.12",
    "webpack-dev-server": "^3.11.0"
  }
}
```

Now create a folder named “**webpack**” with the all configurations files needed for the project.

webpack.config.dev.js

```
const path = require('path');
const loaders = require('./loaders');
// Generate an HTML5 file for you that includes all your
// webpack bundles in the body using script tags
const HtmlWebPackPlugin = require("html-webpack-plugin");
module.exports = {
  mode: "development",
  target: "web",
  devtool: "cheap-module-source-map",
  entry: ["./src/app.js"],  // the application entry point
  output: { // where to put bundles are and how to name these file
    path: path.resolve(__dirname, "dist"),
    publicPath: "/",
    filename: "bundle.js"
  },
  devServer: {
    stats: "minimal",
    overlay: true,
    historyApiFallback: true,
    disableHostCheck: true,
    headers: { "Access-Control-Allow-Origin": "*" },
    https: false
  },
  plugins: [
    new HtmlWebPackPlugin({
      template: "src/index.html"
    })
  ],
  module: {
    rules: [
      loaders.JSLoader
    ]
  },
  output: {
    path: path.resolve(__dirname, "dist"),
    filename: "js/[name].bundle.js"
  },
};
```

`loaders.js`

```
const JSLoader = {
  test: /\.js$/,
  exclude: /node_modules/,
  use: {
      loader: 'babel-loader',
      options: {
      presets: ['@babel/preset-env']
    }
  }
};
module.exports = {
  JSLoader: JSLoader
};
```

As yo see we introduced a webpack plugin named “**HtmlWebPackPlugin**” to create **index.html** automatically for us with an included **<*script*>** tag with its ***src*** pointing to the generated javascript **app.bundle.js** in this case.

After that we need to update package.json to load webpack files in the script attribute as show. For now, this is the **dev** option, the most important. Note that we are using port 3000, the same we put when we create the container. The other important thing here is the “**host 0.0.0.0”** to make our project accessible from outside.

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "webpack-dev-server --config ./webpack/webpack.config.dev.js --hot --
host 0.0.0.0 --port 3000"
}
```

Finally to test this configuration we need to add some folders and files from the project root folder to get the following structure:

```
node_modules/
        ...
src/
    js/
        util.js

    app.js
    index.html

webpack/
    loaders.js
    webpack.config.dev.js

package.json
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Welcome</title>
</head>
<body>
  <div id="welcome"></div>
</body>
</html>
```

app.js

```
import welcome from './js/util';
welcome();
```

js/util.js

```
function welcome() {
  var el = document.getElementById("welcome")
  var text = document.createTextNode("Hello from Demo")
  el.appendChild(text)
}
export default welcome;
```

Now the project is ready to make the first test running WebPack in dev mode.|

```
npm run dev
> demo@1.0.0 dev /opt/apps/demo
> webpack-dev-server --config ./webpack/webpack.config.dev.js --hot --host
0.0.0.0 --port 3000
i ｢wds｣: Project is running at http://0.0.0.0:3000/
i ｢wds｣: webpack output is served from /
i ｢wds｣: Content not from webpack is served from /opt/apps/demo
i ｢wds｣: 404s will fallback to /index.html
i ｢wdm｣:    37 modules
i ｢wdm｣: Compiled successfully.
```

It show us that we have a server waiting for request at http://localhost:3000/, then if you access it in the browser you will get the following:

Hello from Demo

and the generated index.html at *http://localhost:3000/* is:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Welcome Step</title>
</head>
<body>
  <div id="welcome">Hello from Demo</div>
  <script src="js/main.bundle.js"></script>
</body>
</html>
```

> *Remeber that the* **HtmlWebPackPlugin** *is helping us updating the template file adding a script link to the generated javascript bundle file located in the dist folder.*

ESLint

It is a tool that analyze the javascript code in the project trying to find some common problems in it using a set of rules.

```
npm i eslint eslint-loader --save-dev
```

Then we need to update *webpack/loaders.js* to add and export the new loader

loaders.js

```
const ESLintLoader = {
  test: /\.js$/,
  enforce: 'pre',
  exclude: /node_modules/,
  use: {
    loader: 'eslint-loader',
    options: {
      configFile: __dirname + '/.eslintrc'
    },
  }
};
module.exports = {
  JSLoader: JSLoader,
  ESLintLoader: ESLintLoader
};
```

Here ESLint has its own configuration file named **.eslintrc** with the suggested guidelines to avoid code violations.

.eslintrc

```
{
  "extends": "eslint:recommended",
  "env": {
    "browser": true,
    "node": true,
    "es6": true
  },
  "parserOptions": {
    "ecmaVersion": 6,
    "sourceType": "module"
  },
  "rules": {
    "no-console": "off",
    "strict": ["error", "global"],
    "curly": "warn"
  }
}
```

The last step is to put ESLint as dependency in **webpack.config.dev.js**:

```
module: {
    rules: [
      loaders.JSLoader,
      loaders.ESLintLoader
    ]
},
```

Now is time to make an ESLint test to see an error detection in practique. Add to the file *app.js* the next function then run application again and you will see the error.

```
function doNothing() {}
```

```
npm run dev
> demo@1.0.0 dev /opt/apps/demo
> webpack-dev-server --config ./webpack/webpack.config.dev.js --hot --host
0.0.0.0 --port 3000
i ｢wds｣: Project is running at http://0.0.0.0:3000/
i ｢wds｣: webpack output is served from /
i ｢wds｣: Content not from webpack is served from /opt/apps/demo
i ｢wds｣: 404s will fallback to /index.html
✖ ｢wdm｣:    37 modules
ERROR in ./src/js/util.js
Module Error (from ./node_modules/eslint-loader/dist/cjs.js):
/opt/apps/demo/src/js/util.js
  7:10  error  'doNothing' is defined but never used  no-unused-vars
✖ 1 problem (1 error, 0 warnings)
i ｢wdm｣: Failed to compile.
```

You can also use the '**friendly-errors-webpack-plugin**' to helping you showing more firendly error messages, to do that just add it to the project and as a plugin dependency in webpack.config.dev.js

```
npm install friendly-errors-webpack-plugin --save-dev
```

webpack.config.dev.js

```
const FriendlyErrorsWebpackPlugin = require('friendly-errors-webpack-plugin');

var webpackConfig = {
  // ...
  plugins: [
    new FriendlyErrorsWebpackPlugin(),
    ...
  ],
  // ...
}
```

Working with styles: PostCSS and Stylelint

Now we have our project working and configured, is time to process css files and **PostCSS** is the right tool to check css rules, verify browser compatibility, autoprefixing, minification and others.

Install it as a project dependency and update loaders.js file adding a css loader entry to process CSS files in the project building fase.

```
npm i postcss postcss-import postcss-loader style-loader css-loader postcss-
preset-env --save-dev
```

loaders.js

```
const CSSLoader = {
  test: /\.css$/,
  exclude: /node_modules/,
  use: [
    {
      loader: 'style-loader',
    },
    {
      loader: 'css-loader',
      options: {
        importLoaders: 1
      },
    },
    {
      loader: 'postcss-loader',
      options: {
        postcssOptions: {
          config: __dirname + '/postcss.config.js',
          sourceMap: true
        }
    },
  ],
};
module.exports = {
  JSLoader: JSLoader,
  ESLintLoader: ESLintLoader,
  CSSLoader: CSSLoader
};
```

At the same time we need to create **postcss.config.js** in the webpack folder to manage **postcss** configuration and plugins.

postcss.config.js

```
module.exports = {
    plugins: [
        'postcss-import',
        {
            'postcss-preset-env': {
                'browsers': 'last 2 versions',
            }
        }
    ]
}
```

Then you need to add the loader to the **webpack.config.dev.js** file.

webpack.config.dev.js

```
module: {
    rules: [
      loaders.JSLoader,
      loaders.ESLintLoader,
      loaders.CSSLoader
    ]
  },
```

To test the new configuration create a styles folder inside src, then add the app.css file in it

app.css

```
#welcome {
    font-size: 2.5em;
    color: royalblue;
}
.normal-text {
    font-size: 1.5em;
}

.highlight-text {
    background-color: #317399;
    padding: 0 5px;
    color: #fff;
}
.blue-text {
    color: blue
}
```

Update **index.html** and **app.js** files to link both with the style file.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Welcome Step</title>
</head>
<body>
  <script src="dist/bundle.js"></script>
  <div id="welcome"></div>
  <h2>PostCSS test</h2>
  <p class="normal-text;">Its <strong class="normal-text highlight-text">an
example text</strong> to show how to use PostCss</p>
  <p class="normal-text;">Here we use the <strong>some css rules</strong> from
the app.css style file</p>
  <table>
    <tbody>
      <tr>
        <td><strong>Country</strong></td>
        <td><strong>Capital</strong></td>
      </tr>
      <tr>
        <td>USA</td>
        <td>Washington</td>
      </tr>
      <tr>
        <td>England</td>
        <td>London</td>
      </tr>
      <tr>
        <td>Japan</td>
        <td>Tokio</td>
      </tr>
    </tbody>
  </table>
  <p class="normal-text blue-text">This is a blue text parragraph.</p>
</body>
</html>
```

app.js

```
import welcome from './js/util';
import './styles/app.css';
welcome();
```

Now run app again and refresh the browser to see the index page updated, but in the process of writing css rules we need also to check for errors, compatibility, formating, etc., and the **Stylelint** plugin will make it reading and processing every styles to detect any malformation in css rules.

To add it to the project as dependency, update the webpack configuration file and create the ***stylelint.config.js*** and **css.plugins.config.js** were we add the css checking rules needed by the project.

```
npm i stylelint stylelint-webpack-plugin --save-dev
```

stylelint.config.js

```
// Some simple css validation rules
module.exports = {
  rules: {
    // check the style block should not be empty
    "block-no-empty": true,
    // check the style property should be valid
    "property-no-unknown": true,
  }
};
Lets create another file configuration in the same folder with the following
name and content:
```

css.plugins.config.js

```
const _StyleLintPlugin = require('stylelint-webpack-plugin');
const StyleLintPlugin = new _StyleLintPlugin({
    configFile: path.resolve(__dirname, 'stylelint.config.js'),
    context: path.resolve(__dirname, '../src/styles'),
    files: '**/*.css',
    failOnError: false,
    quiet: false,
});

module.exports = {
    StyleLintPlugin: StyleLintPlugin
};
```

webpack.config.dev.js

```
const path = require('path');
const loaders = require('./loaders');
const cssplugins = require('./css.plugins.config');
...
  plugins: [
    ...
    cssplugins.StyleLintPlugin
  ],
```

We are ready now to test our css files, to do that add in app.css a malformed css rule:

```
.green-text { green-color: green }
```

to see the following error:

```
ERROR in
src/styles/app.css
 20:5  ✖  Unexpected unknown property "green-color"   property-no-unknown
i ｢wdm｣: Failed to compile.
```

In real project I suggest to use the '***stylelint-config-standard***' plugin to check the most common stylistic conventions rules.

```
npm install stylelint-config-standard --save-dev
```

Then the updated config will be the following:

stylelint.config.js

```
module.exports = {
  "extends": "stylelint-config-standard",
  rules: {
    "indentation": 4,
    "number-leading-zero": null,
    "unit-allowed-list": ["em", "rem", "px", "s"]
  },
};
```

Workin with multiple environment

Until here we get a developer environment but if you want a production code you need to make some changes and additions to webpack configurations to get it.

The first things to do is to split the ***webpack.config.dev.js*** into several files: **webpack.common.js, webpack.dev.js** and **webpack.prod.js**, and put things in there, but before that download and install the following plugins required in netx steps:

```
npm i webpack-merge clean-webpack-plugin mini-css-extract-plugin cssnano --save-dev
```

webpack.common.js

```
const path = require('path');
const loaders = require('./loaders');
const cssplugins = require('./css.plugins.config');
const FriendlyErrorsWebpackPlugin = require('friendly-errors-webpack-plugin');

// Generate an HTML5 file for you that includes all your
// webpack bundles in the body using script tags
const HtmlWebPackPlugin = require("html-webpack-plugin");
// remove all files inside webpack's output.path directory,
// as well as all unused webpack assets after every successful rebuild
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
```

```
module.exports = {
  entry: ["./src/app.js"],  // the application entry point
  output: { // where to put bundles are and how to name these file
    path: path.resolve(__dirname, "dist"),
    publicPath: "/",
    filename: "bundle.js"
  },
  plugins: [
    new FriendlyErrorsWebpackPlugin(),
    new CleanWebpackPlugin(),
    new HtmlWebPackPlugin({
      template: "src/index.html"
    }),
    cssplugins.StyleLintPlugin
  ],
  module: {
    rules: [
      loaders.JSLoader,
      loaders.ESLintLoader,
      loaders.CSSLoader
    ]
  },
  output: {
    path: path.resolve(__dirname, "dist"),
    filename: "js/[name].bundle.js"
  },
};
```

webpack.dev.js

```
// For development phase
const { merge } = require('webpack-merge');
const common = require('./webpack.common.js');

module.exports = merge(common, {
  mode: "development",
  target: "web",
  devtool: "cheap-module-source-map",
  devServer: {
    stats: "minimal",
    overlay: true,
    historyApiFallback: true,
    disableHostCheck: true,
    headers: { "Access-Control-Allow-Origin": "*" },
    https: false
  },
});
```

webpack.prod.js

```
// For production builds
const { merge } = require('webpack-merge');
const common = require('./webpack.common.js');

module.exports = merge(common, {
  mode: 'production',
  devtool: 'source-map',
});
```

Next update the **package.json** file to include a run develop and production scripts

package.json

```
...

  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "webpack-dev-server --config ./webpack/webpack.dev.js --hot --host 
0.0.0.0 --port 3000",
    "build": "webpack -p --config ./webpack/webpack.prod.js --display-error-
details"
  },

...
}
```

Run project again and you see some explicit css errors in the app.css file as a consequence of the "stylelint-config-standard" use:

```
ERROR in 
src/styles/app.css
  5:1    ✖  Expected empty line before rule            rule-empty-line-before
 14:1    ✖  Expected empty line before rule            rule-empty-line-before
 15:15   ✖  Expected a trailing semicolon              declaration-block-
trailing-semicolon
 19:1    ✖  Expected empty line before comment         comment-empty-line-before
 19:40   ✖  Unexpected missing end-of-source newline   no-missing-end-of-source-
newline
```

After fixing de css and rerunning the project you will get this in the dist folder

```
dist/
    js/
        main.bundle.js
        main.bundle.js.map
    index.html
```

As you have been noted the css files not exist there, and if you remember we include some plugins in the last install and now we need its, so edit and update the next files:

postcss.config.js

```
module.exports = {
    plugins: [
        'postcss-import',
        {
            'postcss-preset-env': {
                'browsers': 'last 10 versions',
            }
        },
        require('cssnano')({
            preset: 'default',
        })
    ]
}
```

loaders.js

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

...

const CSSLoader = {
  test: /\.css$/,
  exclude: /node_modules/,
  use: [
    {
      loader: 'style-loader',
    },
    {
      loader: MiniCssExtractPlugin.loader,
    },
...
```

webpack.common.js

```
const path = require('path');
const loaders = require('./loaders');
const cssplugins = require('./css.plugins.config');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
...

  plugins: [
...
    new MiniCssExtractPlugin({
        filename: '[name].css',
        chunkFilename: '[id].css',
      }),
...
```

package.json

```
{
  ...

  "devDependencies": {

  ...

  },
  "browserslist": [
    "> 5%",
    "last 4 version"
  ]
}
```

app.css

```
table {
    display: grid;
    user-select: none;
    background: linear-gradient(to bottom, white, black);
}
...
```

Running project again you will get the source map and the minified version of the css files, but if you lok closer you will see another usefull transformation to the table element rule: autoprefixing, helping us adding vendor prefixes to CSS rules.

```
## Original:
table {
    display: grid;
    user-select: none;
    background: linear-gradient(to bottom, white, black);
}

## Tranformed
table {
    display: grid;
    -webkit-user-select: none;
       -moz-user-select: none;
        -ms-user-select: none;
            user-select: none;
    background: -webkit-gradient(linear, left top, left bottom, from(white),
to(black));
    background: -o-linear-gradient(top, white, black);
    background: linear-gradient(to bottom, white, black);
}
```

Finally project will have the following structure and files:

```
src/
    js/
        util.js
    styles/
        app.css

    app.js
    index.html

webpack/
    .eslintrc
    css.plugins.js
    loaders.js
    postcss.config.js
    stylelint.config.js
    webpack.config.dev.js

package.json
```

Project resources

You can download from https://github.com/manuel-molina/webpack-demo this pdf file and all project files, also you can review the following useful links I found over the Internet:

- Install Docker Engine
- How to Install and Use Docker on Linux
- Installing Docker on Linux
- How to Install Node.js and npm on Ubuntu 20.04

- JavaScript’s strict mode
- A tale of Webpack 4 and how to finally configure it in the right way.
- *Working with Webpack 4, ES6, PostCSS with preset-env*
- *Using html-webpack-plugin to generate index.html*
- *How to set up React with Webpack and Babel*
- *Tutorial: How to set up React, webpack, and Babel from scratch (2020)*
- Getting Started with ESLint
- How to use ESLint in Webpack
- *Configuring ESLint*
- Eslint recommended style guidelines
- *Linting React using ESLint and Babel*.
- ESLint configuration and best practices
- Additional Configuration for webpack
- Introduction to PostCSS With cssnext and cssnano
- How to configure CSS and CSS modules in webpack
- Web Design Tutorials: PostCSS
- Npm PostCSS site
- The stylelint-config-standard NPM plugin site
- The complete best practices for minifying CSS
- Webpack By Example: Part 1
- Webpack By Example: Part 2
- The webpack production guide