Webpack 5

Class	Frontend
Created	@September 11, 2021 9:52 PM
Edited	@October 16, 2021 2:57 AM
Link	
Related to Projects (Tags)	
Source	ODIN

Decided to take a read on the SURVIVEJS Webpack book online because I'm having a lot of trouble with it.

Notes:

Getting Started

- Local installation > global installation. Why? Better control over what version and what is being added, plus continuous integration environment.
- CI (Cont. Integration) is a type of production environment in which ongoing automation and monitoring are implemented to improve and expedite development processes for integration and testing, and deployment and delivery.
- webpack-cli package can be used to handle webpack through the terminal, it doesn't need a config file for basic set up. A good alternative is webpack-nano for basic usage.
- For more complicated set ups I'll require a separate, custom mode webpack.config.js
- mini-html-webpack-plugin and html-webpack-plugin can be used to generate HTML entry points automatically.

- Configuring scripts in the **package.json** file can be useful to manage webpack quickly.

Development Server

- It's possible to setup **BrowserSync** with webpack through ***browser-sync-webpack-plugin*** but there are better alternatives such as *watch mode and a development server.*
- **webpack-dev-server** is the officially mantained dev server running in-memory.
- WDS depends on webpack-cli
- WPS **webpack-plugin-serve** is a third party plugin. Iy combines the watch mode with HMR Hot Module Replacement.
- There is a **webpack-plugin-ramdisk** to write in the RAM instead, it improves performance.

```
npm add webpack-plugin-serve --develop
```

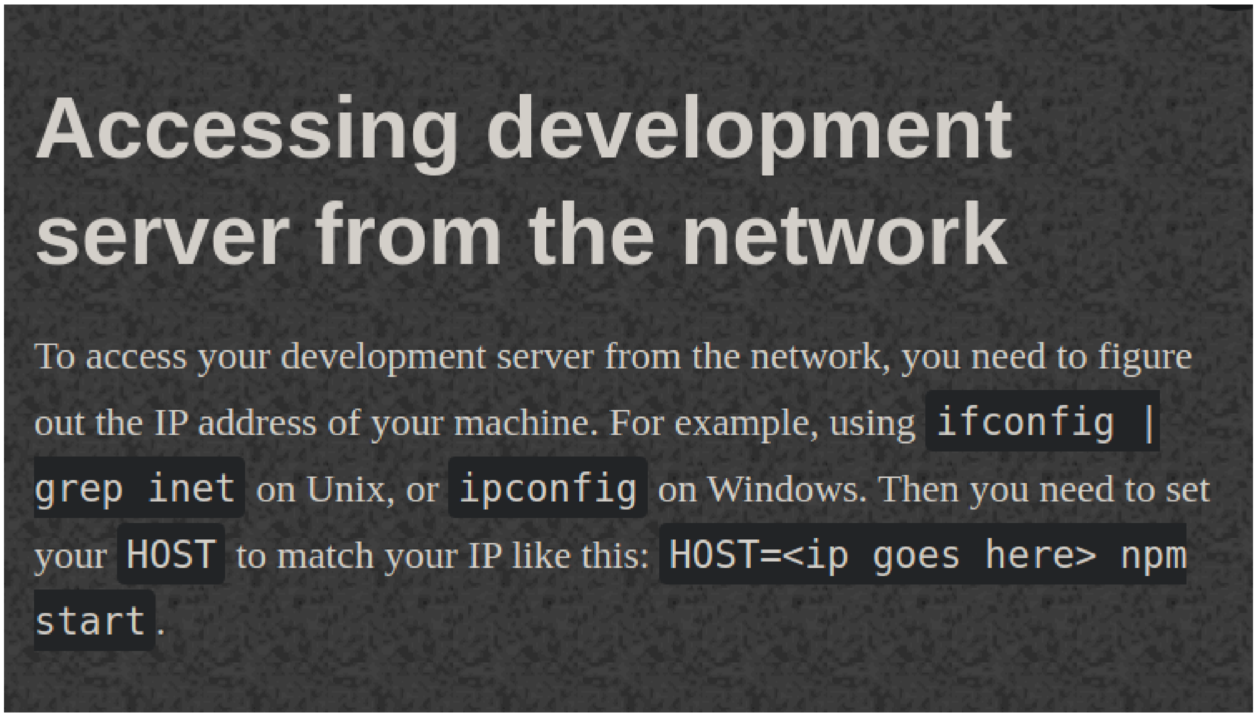
- When installing certain plugins there might be problems with the peer dependencies. This is a problem with NPM V7, to solve use:

```
npm install --legacy-peer-deps [filename] -d
```

- —force can be used too, but it is better to just use old versions.

```
const { mode } = require("webpack-nano/argv");
const {
  MiniHtmlWebpackPlugin,
} = require("mini-html-webpack-plugin");
const { WebpackPluginServe } = require("webpack-plugin-serve");

module.exports = {
  watch: mode === "development",
  entry: ["./src", "webpack-plugin-serve/client"],
  mode,
  plugins: [
    new MiniHtmlWebpackPlugin({ context: { title: "Demo" } }),
    new WebpackPluginServe({
      port: process.env.PORT || 8080,
      static: "./dist",
      liveReload: true,
      waitForBuild: true,
    }),
  ],
};
```

- webpack-plugin-serve can be used to hard-refresh the webpage each time a change is inputed, It's a better version for working.



Accessing development server from the network

To access your development server from the network, you need to figure out the IP address of your machine. For example, using `ifconfig | grep inet` on Unix, or `ipconfig` on Windows. Then you need to set your `HOST` to match your IP like this: `HOST=<ip goes here> npm start`.

Maybe useful in the future?

- 'Polling instead of watching files' covered a section based on the chance that wp —watch doesn't work on old unix and windows system. I've skipped it.

Faster to develop webpack configuration

- Webpack doesn't identify changes in the config file natively, but it can be done by using nodemon (node monitoring tool).
- Install it

```
npm add nodemon --develop
```

- Configure package.json

```
{
  "scripts": {

    "watch": "watch": "nodemon --watch \"./webpack.*\" --exec \"npm start\"",
    "start": "wp --mode development"

  }
}
```

Composing Configuration

As the needs of my projects grow, the need to optimize the webpack configuration process does too.

Ways to do so:

- Maintain configuration within multiple files for each of my environements, and point webpack to each through the —config paramter, sharing config through module imports.
- Push config to a library, which I can then use. webpack-config-plugins, Neutrino, webpack-blocks.
- Push config to a tool such as create-react-app, kyt y nwb.
- Maintain all configurations within a single file and branch there and rely on the —*mode* parameter.

Composing Configuration by Merging

There is a plugin called ***webpack-merge*** that does this. It concatenates arrays and merges objects instead of overriding them.

```
npm add webpack-merge --develop
```

To 'give a degree of abstraction' and what I understand is: to increase the complexity and precision of this aproach, use 2 config files: webpack.config.js for high level config and webpack.parts.js for config the parts to consume.

webpack.parts.js - example

```
const { WebpackPluginServe } = require("webpack-plugin-serve");
const {
  MiniHtmlWebpackPlugin,
} = require("mini-html-webpack-plugin");

exports.devServer = () => ({
  watch: true,
  plugins: [
    new WebpackPluginServe({
      port: process.env.PORT || 8080,
      static: "./dist", // Expose if output.path changes
      liveReload: true,
      waitForBuild: true,
    }),
  ],
});

exports.page = ({ title }) => ({
  plugins: [new MiniHtmlWebpackPlugin({ context: { title } })],
});
```

webpack.config.js

```
const { mode } = require("webpack-nano/argv");
const { merge } = require("webpack-merge");
const parts = require("./webpack.parts");

const commonConfig = merge([
  { entry: ["./src"] },
  parts.page({ title: "Demo" }),
]);

const productionConfig = merge([]);

const developmentConfig = merge([
  { entry: ["webpack-plugin-serve/client"] },
  parts.devServer(),
]);

const getConfig = (mode) => {
  switch (mode) {
    case "production":
      return merge(commonConfig, productionConfig, { mode });
    case "development":
      return merge(commonConfig, developmentConfig, { mode });
    default:
      throw new Error(`Trying to use an unknown mode, ${mode}`);
  }
};

module.exports = getConfig(mode)
```

Summary

@October 16, 2021 2:39 AM

Added another section as I forgot a lot of it when initiating another project.

Every time a project is initiated webpack has to be installed in the directory, this is done by

```
npm install webpack webpack-cli —save-dev
```

—save means it will store the information to be shipped on the production bundle

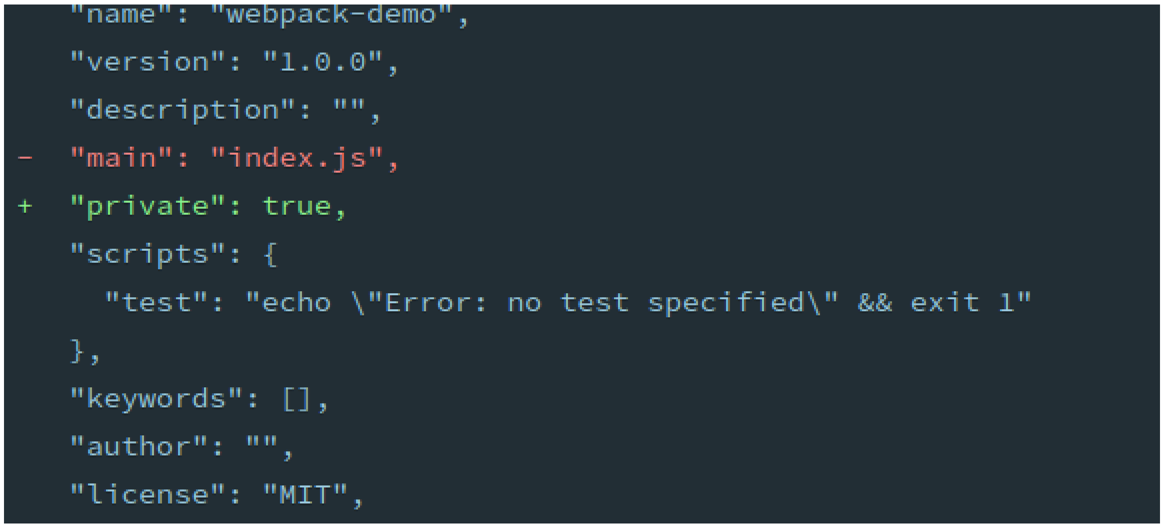
—save-dev means it will store the information to only be used in the development portion and won't be shipping on production

important ! ! !

then npm can be initiated in the locale, creating a config file with

```
npm init -y
```

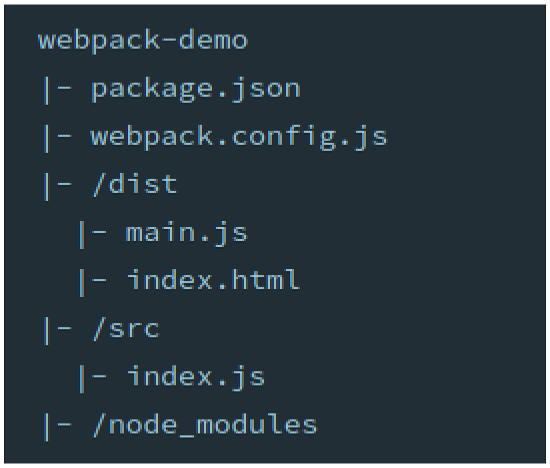
at this point, edit the package.json contents to remove the posibility of accidentally publishing the code



```diff
   "name": "webpack-demo",
   "version": "1.0.0",
   "description": "",
-  "main": "index.js",
+  "private": true,
   "scripts": {
     "test": "echo \"Error: no test specified\" && exit 1"
   },
   "keywords": [],
   "author": "",
   "license": "MIT",
```

this will prevent publishing

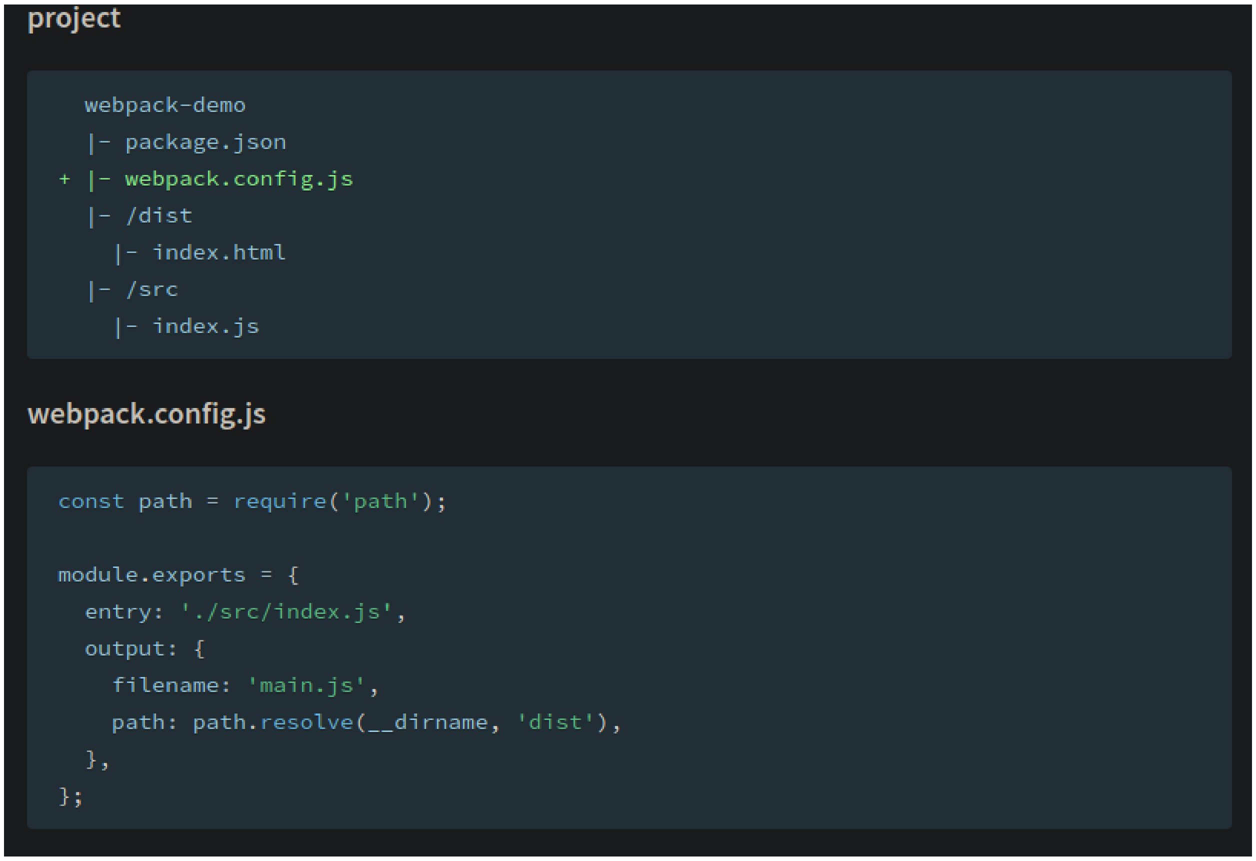
create a work tree suggested by the webpack.js.org page, like this:



```
webpack-demo
|- package.json
|- webpack.config.js
|- /dist
  |- main.js
  |- index.html
|- /src
  |- index.js
|- /node_modules
```

webpack can be initiated by using npx webpack

nowadays iteraton of webpack doesn't require a config file but it's better to have one, make a webpack.config.js file



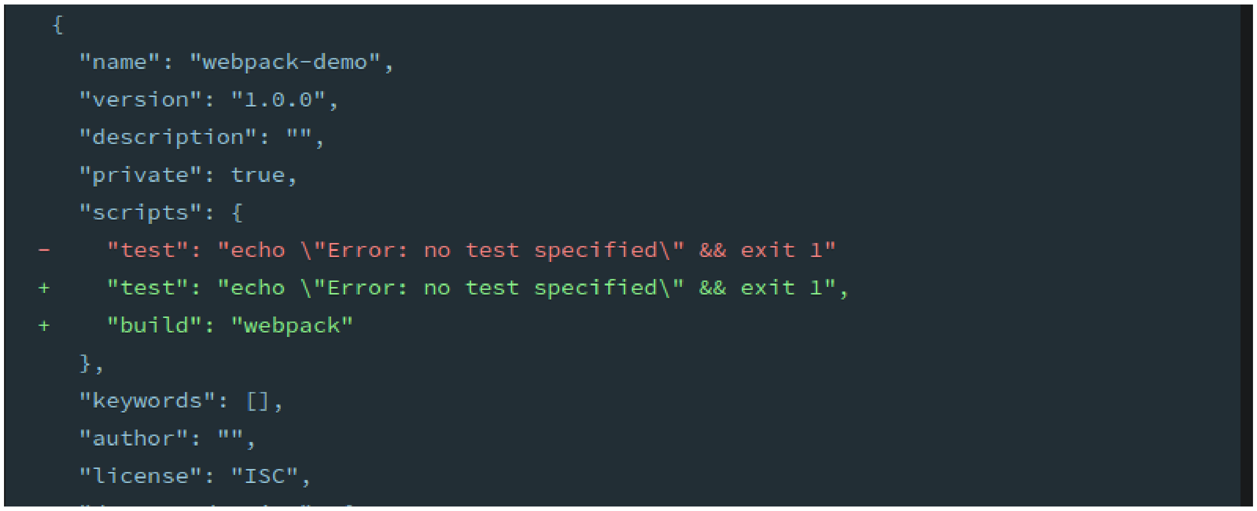
project

```
  webpack-demo
  |- package.json
+ |- webpack.config.js
  |- /dist
    |- index.html
  |- /src
    |- index.js
```

webpack.config.js

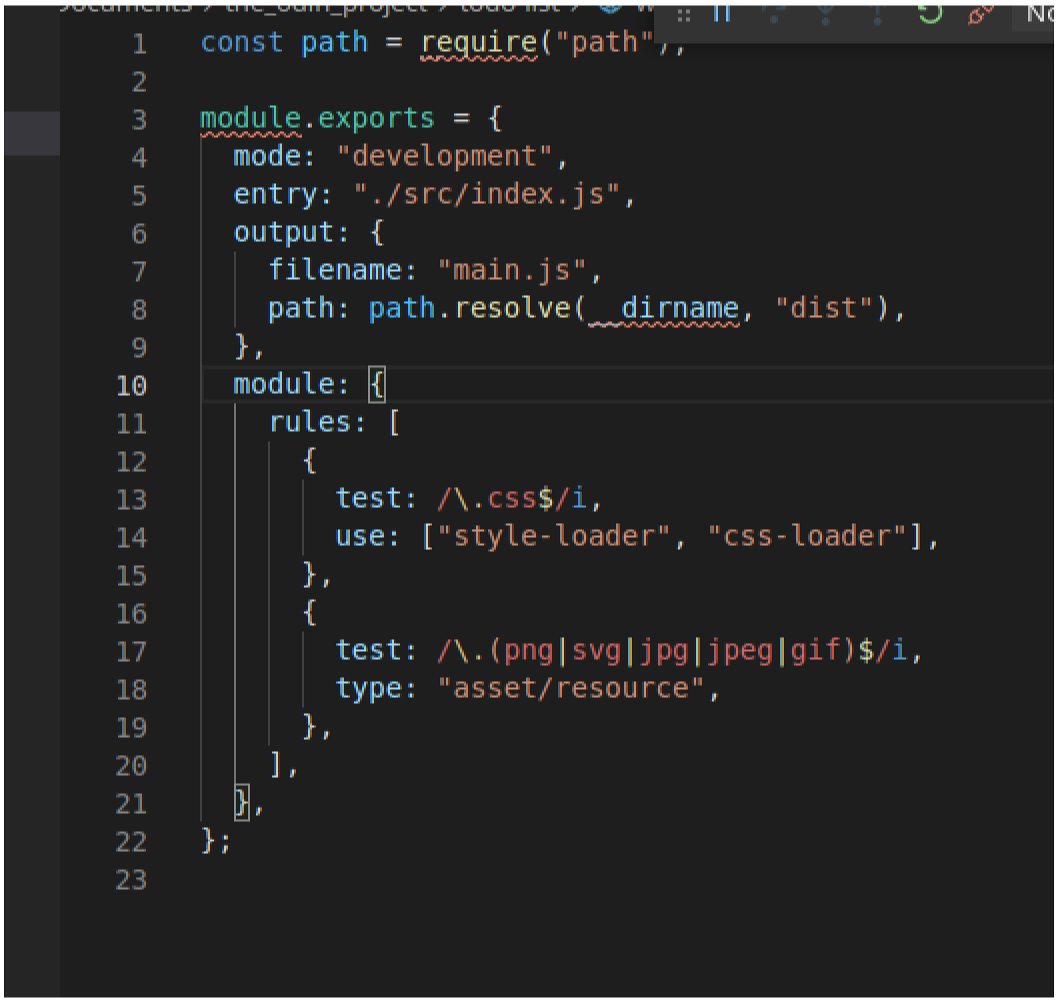
```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'main.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```

it's good to create custom scripts for running webpack easily from the CLI



```
  {
    "name": "webpack-demo",
    "version": "1.0.0",
    "description": "",
    "private": true,
    "scripts": {
-     "test": "echo \"Error: no test specified\" && exit 1"
+     "test": "echo \"Error: no test specified\" && exit 1",
+     "build": "webpack"
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
```

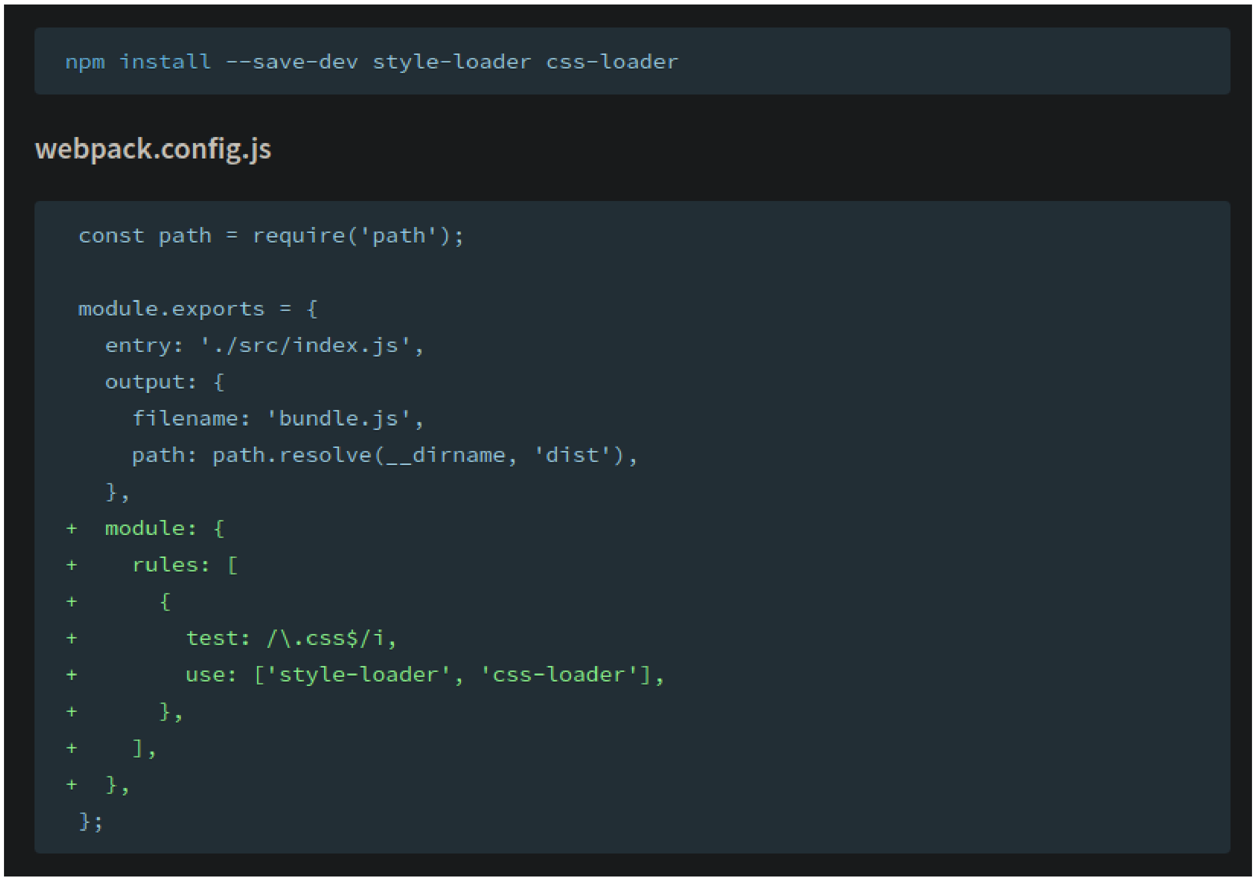
I have previously used this webpack config to incorportate images and css



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

module.exports = {
  mode: "development",
  entry: "./src/index.js",
  output: {
    filename: "main.js",
    path: path.resolve(__dirname, "dist"),
  },
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: ["style-loader", "css-loader"],
      },
      {
        test: /\.(png|svg|jpg|jpeg|gif)$/i,
        type: "asset/resource",
      },
    ],
  },
};
```

config ready to handle css files and images

to load css you need a style-loader and a css-loader, they must be added to module configuration (already in the sample above)



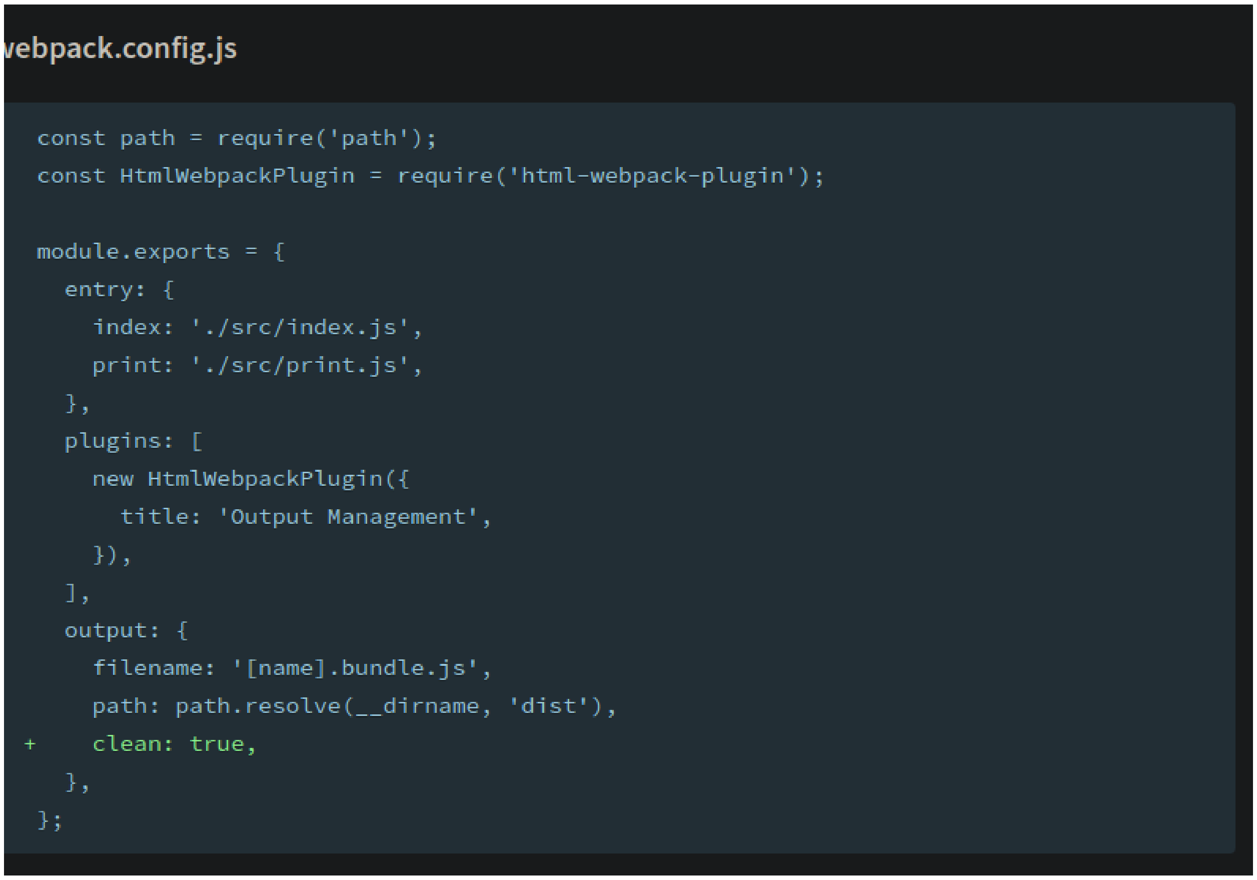
```
npm install --save-dev style-loader css-loader
```

webpack.config.js

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

 module.exports = {
   entry: './src/index.js',
   output: {
     filename: 'bundle.js',
     path: path.resolve(__dirname, 'dist'),
   },
+  module: {
+    rules: [
+      {
+        test: /\.css$/i,
+        use: ['style-loader', 'css-loader'],
+      },
+    ],
+  },
 };
```

further reading to handle assets (inputed info): https://webpack.js.org/guides/asset-management/

clean can be added to the output options in the webpack.config.js to cleanse the dist folder everytime a new compilation is completed


ebpack.config.js

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

module.exports = {
  entry: {
    index: './src/index.js',
    print: './src/print.js',
  },
  plugins: [
    new HtmlWebpackPlugin({
      title: 'Output Management',
    }),
  ],
  output: {
    filename: '[name].bundle.js',
    path: path.resolve(__dirname, 'dist'),
+    clean: true,
  },
};
```

further reading on output management: https://webpack.js.org/guides/output-management/