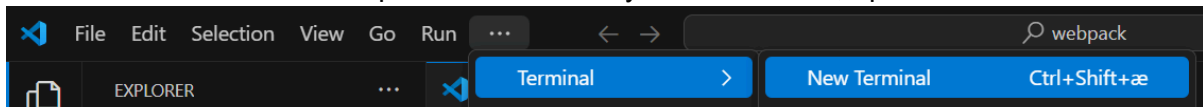


Installation of webpack

Basic Setup

To install webpack, you want to open a project in your preferred coding program. For this tutorial we'll be using Visual Studio Code.

When you've opened a project, make sure to open a folder for your project files to lay in. In our demo we've called it webpack-demo. When you've done this, open a new terminal.



In the terminal you write:

"npm install --save-dev webpack" or *"npm install --save-dev webpack@<version>"* if you wish to download a specific version. For this to work, you need to have a new version of node downloaded.

```
PS C:\VSC\Hovedforløb\Automatisering\webpack> npm install --save-dev webpack
```

Then follow that up with writing:

"npm install webpack webpack-cli --save-dev"

```
PS C:\VSC\Hovedforløb\Automatisering\webpack\webpack-demo> npm install webpack webpack-cli --save-dev
```

Then

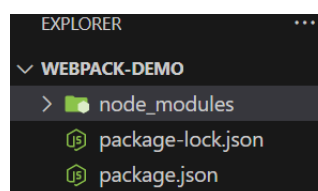
"npm init -y"

This just says yes to anything it may ask.

```
PS C:\VSC\Hovedforløb\Automatisering\webpack\webpack-demo> npm init -y
```

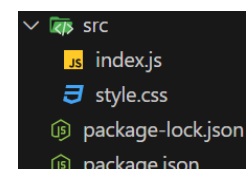
Explorer

Your explorer should look like this:

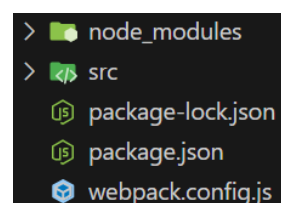


Add a folder and name it: src

In this folder, create a index.js file and a style.css file



Add a new file and name it: webpack.config.js
Make sure it's not in a folder.



Create javascript

In your terminal, write:

`"npx webpack"`

```
PS C:\VSC\Hovedforløb\Automatisering\webpack\webpack-demo> npx webpack
assets by status 0 bytes [cached] 1 asset

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value.
Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/
```

You will get a warning saying the mode option has not been set, that is intended.

Your program should have created a folder called dist, with a main.js inside it. This file should be empty.

As an example we can write this inside index.js

```
1  const headline = 'Welcome to the webpage';
2  document.querySelector('h1').innerText = headline;
```

Create an index.html file inside your dist folder.

In your index.html set up a basic page. Create a div with the class container. Inside the container create the h1. It can say whatever, since we will be replacing it with our javascript. Make sure to use script:src and link it to main.js

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  |   <title>Webpack CSS example</title>
7  |   <script src="./main.js" defer></script>
8  </head>
9  <body>
10 |   <div class="container">
11 |     <h1>placeholder</h1>
12 |   </div>
13 </body>
14 </html>
```

Add css

For the purpose of this demo, in your "style.css" in the src folder, we will create some basic styling as an example.

```
1  body {
2  |   background-color: #2e8b57;
3  |   color: white;
4  | }
5
6  h1 {
7  |   font-size: 3rem;
8  |   text-align: center;
9  | }
```

Go back to your index.js and create an import your css like shown.

```
1 import './style.scss'
2
3 const headline = 'Welcome to the internet';
4 document.querySelector('h1').innerText = headline
```

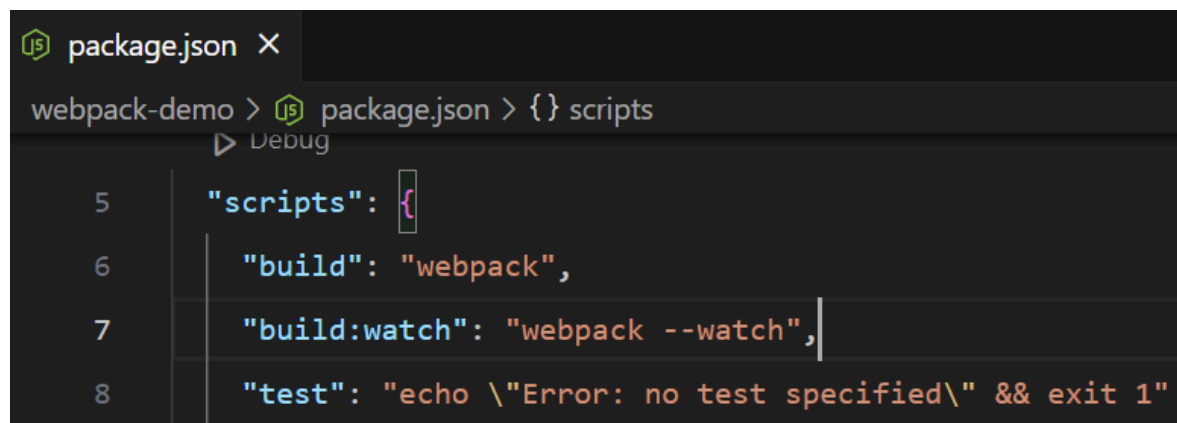
If you run “*npm webpack*” in the terminal, this should give you an error. This is because you are trying to import css into javascript. Js obviously doesn’t understand css coding, and therefore you get an error. We’ll need to install some loaders. Enter your terminal and type out “*npm i -D css-loader style-loader*”

```
PS C:\VSC\Hovedforløb\Automatisering\webpack-demo> npm i -D css-loader style-loader
```

css-loader will convert it to be understandable, but it won’t really know what to do with it. That’s what style-loader does.

Support for css, scss and sass

To be able to support both css, scss and sass, go to package.json and add a “build” script and give it the value of “webpack”, add also a “build:watch” script and give it the value of “webpack --watch”



```
package.json X
webpack-demo > package.json > {} scripts
Debug
5 "scripts": {
6   "build": "webpack",
7   "build:watch": "webpack --watch",
8   "test": "echo \"Error: no test specified\" && exit 1"
```

Rename your style.css file to style.scss so we can use some sass.

In your style.scss file, add the variable `$light-text: white;` and use the variable in body colour. You can also now scope the h1 under the rule for body.

In your terminal, write

“*npm i -D sass sass-loader*”

To get it to convert into readable css.



```
$light-text: white;

body{
  background-color: seagreen;
  color: $light-text;

  h1{
    font-size: 3rem;
    text-align: center;
  }
}
```

webpack.config.js

css, scss and sass

In your webpack.config.js, start by writing out module.exports.

In test, so that it's able to support css, scss and sass, write:

`/\.(s[ac]|c)ss$/i`

This is a regex that will search if you have a file ending in scss, sass, or css.

Under use, write out style-loader, sass-loader and css-loader to use this.

```
module.exports = {  
  ...  
  module: {  
    rules: [  
      {  
        test: /\.(s[ac]|c)ss$/i,  
        use: ['style-loader', 'css-loader', 'sass-loader'],  
      },  
    ],  
  },  
}
```

You can run `"npm run build:watch"` or just `"npm run build"` to check your site.

Type `"npx webpack"` into your terminal again. If you look in your main.js, it should now have a long single line of code.

Plugin

Mini-css-extract-plugin

In the terminal add `"npm i -D mini-css-extract-plugin"`.

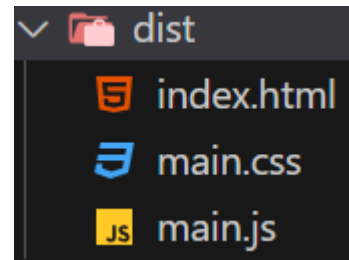
```
PS C:\VSC\Hovedforløb\Automatisering\webpack-demo> npm i -D mini-css-extract-plugin
```

Go to webpack.config.js and add the plugin via a const, and add an extra property named plugins. Then, under "rules", swap out the `style-loader` with `MiniCssExtractPlugin.loader`.

```
2  const MiniCssExtractPlugin = require("mini-css-extract-plugin")  
3  
4  module.exports = {  
5    plugins:[new MiniCssExtractPlugin()],  
6  
7    module: {  
8      rules: [  
9        {  
10         test: /\.(s[ac]|c)ss$/i,  
11         use:[MiniCssExtractPlugin.loader, 'css-loader', 'sass-loader']  
12       },  
13     ],  
14   },  
15 }
```

Run *"npm run build"* in the terminal. There should be a new file generated in the dist folder, named main.css.

Remember to link the main.css file in your html file just like any regular css file, or else the css you previously wrote, will not load onto the site.



If you inspect your page, the styling should no longer be in the head of the html content.

Mode

To change the mode of the generated code in main.js and main.css so that it's not one line of code, add a mode: "development" in the module.exports.

```
module.exports = {  
  mode: "development",  
  
  plugins:[new MiniCssExtractPlugin()],  
  
  module: {
```

Remember to run *"npm run build"* to update the coding.

Sourcemap

In webpack.config.js you need to add *"devtool: 'source-map'"* after module.

This will generate source maps. If you look in your inspect tool, it can now guide you towards your files where the styling comes from.

Make sure to run *"npm run build"* in the terminal, and reload your site

In your dist folder, two new files will have generated, main.css.map and main.js.map.

```
const MiniCssExtractPlugin = require("mini-css-extract-plugin")  
...  
  
module.exports = {  
  plugins:[new MiniCssExtractPlugin()],  
  
  module: {  
    rules: [  
      {  
        test: /\.([ac]|c)ss$/i,  
        use:[MiniCssExtractPlugin.loader, 'css-loader', 'sass-loader',]  
      }  
    ]  
  },  
  
  devtool:"source-map"  
}
```

Postcss

in the terminal, add `npm i -D postcss postcss-loader postcss-preset-env`. This will allow you to do many different cool stuff such as, much more advanced css.

```
PS C:\VSC\Hovedforløb\Automatisering\webpack-demo> npm i -D postcss postcss-loader postcss-preset-env
```

In webpack.config.js add another line to `“use”` called `“postcss-loader”`

```
use:[MiniCssExtractPlugin.loader, 'css-loader', 'sass-loader', 'postcss-loader']
```

To keep it clean, create a new file called `postcss.config.js`. Make sure it isn't inside any folders, just like your `webpack.config.js`

Create a `module.exports` just like in `webpack.config.js`

Make a plugin and write:

`“[require(“postcss-preset-env”)]”`

```
postcss.config.js > <unknown>
1  module.exports = {
  ...
2    plugins: [require("postcss-preset-env")]
3  }
```