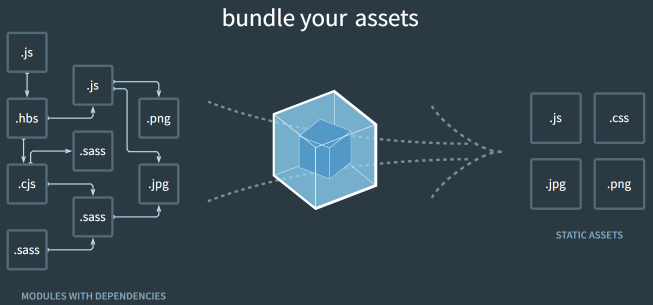
**Webpack [**[**https://webpack.js.org/**](https://webpack.js.org/)**]**

(Traversy Media Webpack tutorial: <https://www.youtube.com/watch?v=lziuNMk_8eQ>)

(Net Ninja Webpack & Typescript: <https://www.youtube.com/playlist?list=PL4cUxeGkcC9hOkGbwzgYFmaxB0WiduYJC>)

When we implemented our projects using vanilla JS, we had to put all the code inside a single JS file and include that JS file from within our HTML file. Ideally, you would like to separate out different functionality into different modules and put the code related to that functionality into a separate JS file. This is what code modularity is all about. In fact, we have seen that JS frameworks such as React and all the others (Angular, Vue, Svelte, etc.) work exactly this way: You decompose your project into different modules and components and combine them all together into a single JS bundle and use that bundle in your HTML file. Webpack is that tool that makes all this bundling together.



The figure shown above taken from Webpack Web site illustrates what Webpack does: It bundles your assets, which may consist of JS files, CSS files, images etc., into a single JS file that you can then include in your HTML file.

**Example**

Let’s assume that we have a project consisting of 3 JS files: bar.js, foo.js and index.js, where index.js is the root of the hierarchy. Here are the files:

|  |
| --- |
| export default function bar() {      const node = document.createElement("h1")      node.innerText = 'Inside bar from bar.js'      document.body.appendChild(node)      console.log(node)  } |

|  |
| --- |
| export default function foo() {      const node = document.createElement("h1")      node.innerText = 'Inside foo from foo.js'      document.body.appendChild(node)      console.log(node)  } |

|  |
| --- |
| import bar from './bar.js';  import foo from './foo.js';  bar();  foo(); |

In order to bundle these 3 JS files into a single JS file, we first need to install webpack and webpack-cli globally as follows:

|  |
| --- |
| % npm –g install webpack webpack-cli |

Then run the webpack command to create the bundle:

|  |
| --- |
| % webpack --entry ./index.js -o ./ --output-filename bundle.js |

We now have a bundle named “bundle.js”. Include this in your HTML file. Look at 13-Webpack/01-demo/

**Using a config file**

Rather than running webpack with command line arguments as we did in the previous section, the usual practice is to create a “webpack.config.js” file and put all these configuration parameters in that file. We will also organize our project a little so that all source files are stored in “src” folder and the bundled files are stored in “dist” folder. Here is our config file. Look at 13-Webpack/02-demo/

|  |
| --- |
| const path = require('path');  module.exports = {    mode: 'development',    entry: './src/index.js',    output: {      path: path.resolve(\_\_dirname, 'dist'),      filename: 'bundle.js',    },  }; |

We can now create our bundle by simply typing the “webpack” command, which uses the configuration parameters in this file to create “bundle.js” into “dist” folder.

|  |
| --- |
| % webpack |

Since we already have a package.json file, where we can specify scripts, people usually create a build target with webpack as follows:

|  |
| --- |
| "scripts": {      "build": "webpack"    }, |

You can then run the following command to build “bundle.js”:

|  |
| --- |
| % npm run build |

**Adding CSS assets**

Let’s now add a simple CSS file to our project that styles the body and the h1 tags. Call this styles.css with the following content:

|  |
| --- |
| body {      background-color: cyan;  }  h1 {      color: red;  } |

We now need to import these styles into our index.js as follows:

|  |
| --- |
| import './styles.css' |

We now need to install styles-loader and css-loader as dev dependencies as follows:

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

Finally, add the following lines to the configuration file, which will first invoke the css-loader and then style-loader for all files with the extension .css.

|  |
| --- |
| module: {      rules: [        {          test: /\.css$/,          use: [            // [style-loader](/loaders/style-loader)            { loader: 'style-loader' },            // [css-loader](/loaders/css-loader)            {              loader: 'css-loader',              options: {                modules: true              }            }          ]        }      ]    } |

We can now create the bundle by typing:

|  |
| --- |
| % npm run build |

Now, our bundle contains the CSS rules, which changes the background color of the body and the text colors of h1.

Look at 13-Webpack/03-demo/

**BlogListApp**

Recall that when we covered JS, we have implemented a SPA BlogListApp, where we put all the JS code inside a single JS file. In this section we will re-rewrite that application using different modules for each component. We will also separate out the data store (the global state) into a separate module. You can now see that our application is now very similar to ReactJS applications. In fact, this is pretty much how ReactJS applications work.