Webpack

You store your code splitting it into two different files or modules. Historically in order to include those modules into your app you need to reference them as script tags inside of your index.html file.

Now, not only this was tedious but also error prone since the order of these scripts mattered.

What if instead of manually managing these script tags in their order we had a tool that did this for us. This tool intelligently takes all those scripts and combine them into one single bundle, then we’ll only need to reference that bundle. Well this and a little bit more is exactly what Webpack does.

Here’s webpack.config.js file

module.exports = {  
entry: ["./global.js" , "./app.js"],  
output: {  
filename: "bundle.js"  
},  
**module: {  
loaders: [  
{  
test: /\.es6$/,  
exclude: /node\_modules/,  
loader: 'babel-loader',  
query: {  
presets: ['react', 'es2015']   
}  
}  
]  
},  
resolve: {  
extensions: ['', '.js', '.es6']  
},**}

When you are using modules, there’s always a module which is a kick off point in your app. In webpack.config file, we specify this module as entry.

Webpack allows you to run transformations on the files before they get added to final bundle, these are called loaders. Here, we are using babel-loader to transform our next-generation javascript to JavaScript of today.

Finally we need to tell the webpack where to put the final bundle with output property.

**Getting Started**

**Webpack Conventions**

1. Webpack works best with NPM, not Bower
2. Uses a module system (AMD, CommonJS, ES6)

**Installing Webpack globally:**

npm install webpack -g

**The most basic of builds:**

In your root directory create 2 files: index.html & app.js

**In app.js:**

document.write('welcome to my app');console.log('app loaded');

**In index.html:**

<html>  
 <body>  
 <script src="bundle.js"></script>  
 </body>  
</html>

Open your console, and run:

webpack ./app.js bundle.js

The above command uses the webpack command (webpack) to reference our app file (./app.js) and the last argument is the name of the file that we want Webpack to create for us (bundle.js).

The above command should have created a file called bundle.js that is our first Webpack bundle! Easy huh?

**Defining a config file**

A configuration file in Webpack is basically a common.js module. The config file is a place to put all of your configuration, loaders (explained later), and other specific information relating to your build.

In your root directory, create a file called webpack.config.js, and add the following code:

module.exports = {  
 entry: "./app.js",  
 output: {  
 filename: "bundle.js"  
 }  
}

**entry** — name of the top level file or set of files that we want to include in our build, can be a single file or an array of files. In our build, we only pass in our main file (app.js).

**output** — an object containing your output configuration. In our build, we only specify the filename key (bundle.js) for the name of the file we want Webpack to build.

Now, to run our app, go to your command line and run the following command:

webpack

Once a webpack.config file is present, the webpack command will build your application based on the configuration made available in the file.