**Webpack**

Introduction

Webpack is a static module bundler for modern Javascript applications. When webpack processes your application, it recursively builds a dependency graph that includes every module your application needs, then packages all of those modules into one or more bundles. Modules can be custom files or files installed by npm.

Webpack by default can only work with native Javascript, but you can use ‘loaders’ to translate other technologies in to Javascript, e.g. CSS, JSX…

Webpack has clever parsing to include most libraries, and it also has a plugin system.

Webpack bundles the entry point and all of its dependencies into a single file.

Example loaders:

* CSS & Style
* SASS & LESS
* JSX (react)
* Babel (translates ES6 into ES5)
* Typescript
* JSON
* Etc…

Installation

-install webpack locally on your projects in dev dependencies

npm init (to create package.json file)

npm install webpack -D

npm install webpack-cli -D

npx webpack app.js –output bundle.js

//app.js being the source file, bundle.js being the compiled file

//this will create a bundle.js file now – don’t be concerned with what it all means

All our work goes in the app.js (src) file.

Example:

*People.js*

Module.exports = “Hello World”;

*App.js*

Let people = require(‘./people.js’);

Alert(people);

To confirm changes, run *npx webpack app.js –output bundle.js* in the terminal. You can add *--watch­* to the end of this command to be able to refresh browser then application saved to see changes (without putting cmd in terminal again – read on to see how this can be made simpler). This command basically runs the app through webpack, changing the contents of the bundle.js file.

More complex examples (to add on to the example above):

1. *People.js*

Function greetHello() {

Return “Hello there”;

}

Module.exports = getHello();

1. *People.js*

Let people = [

{name: ‘Ali Issaee’},

{name: ‘Lauren Foster’}

];

Module.exports = people;

*App.js*

Let people = require(‘./people’);

Console.log(people);

Console.log(people[0].name);

1. *Terminal*

Npm install jquery

*People.js*

(same as e.g. 2)

*App.js*

Let $ = require(‘jquery’);

$(‘body’).append(‘<h1>’ + people[0].name +’</h1>’);

Loader

To install:

Npm install *loadername*

Npm install css-loader style-loader -D

In app.js, you can now type – *require(‘style.css’) –* at the top of the script, with style.css being personal stylesheet. You will need to edit the webpack.config.js file to apply the loader – see below.

Structure

\*create a file called webpack.config.js\*

Module.exports = {

Entry: ‘.src/js/app.js’,

Output: {

Path: \_\_dirname + ‘/dist’,

Filename: ‘bundle.js’

},

Module: {

Rules: [

{test: /\.css$/, loader: “style-loader!css-loader”}

]

}

}

//where rules are loaders – in rules - /\.css$ means apply to all .css file formats.

//once this file is created, you can now run *npx webpack* in terminal, without specifying the src file and compiled file

File structure

Correct folder layout:

**Dist**

Bundle.js

**Src**

Js

App.js

People.js

Css

Style.css

Index.html

Package.json

Webpack.config.js

//in package.json file, under *scripts*, create property *“build”: “npx webpack”* – in terminal, you can now run *npm run build* to update bundle.js file and view up to date application.

//ensure the script src in the index.html file shows the location of the bundle.js file – dist/bundle.js

Babel

Babel translates ES6 in to ES5 so you are able to use ES6 syntax and keep up to date, whilst the babel loader converts this to ES5 so it will still work in your browser. This can be installed and added in web.config.js file under *rules*:

{test: /\.js$/, loader: ‘babel-loader’, exclude: /node\_modules/, query: {presets: [‘es2015’]}}

This is important to note as syntax such as *let* is ES6 syntax and may need to be used in application along others.