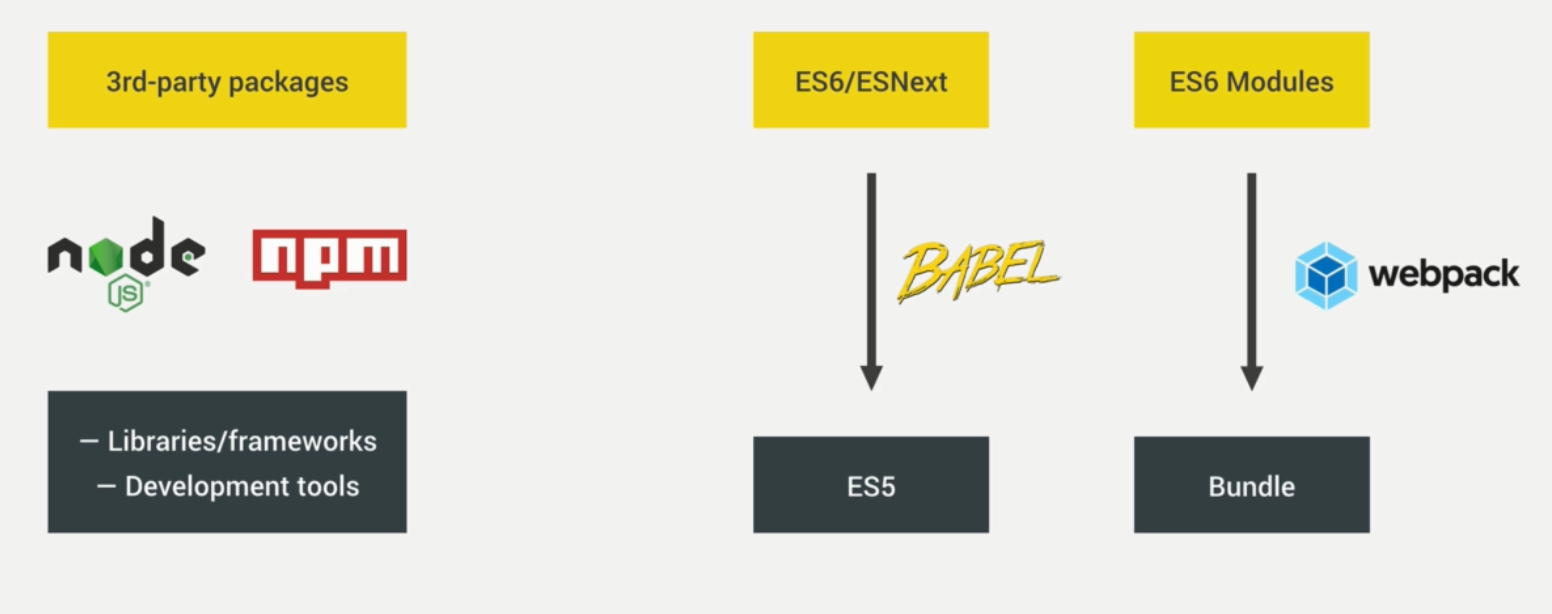
SETTING UP THE BABEL AND WEBPACK FOR THE WEBSITE



* Download Node,js
* Create package .json file
  + Npm init in cmd to create
  + This utility will walk you through creating a package.json file.
  + It only covers the most common items, and tries to guess sensible defaults.
  + This will create a package.json file
* Download Webpack thrugh npm
  + npm install webpack --save-dev
  + ‘—save-dev’ will save webapck as a developer dependency
  + we basically have two types of Node.js packages.
  + Ones are libraries and frameworks and stuff like that, and others are development tools.
  + So libraries like React or jQuery, for example,
  + they are really dependencies
  + because we use this code in our,
  + in our product, in our app.
  + But something like webpack,
  + this is really just a development tool.
  + And so, it's here in the dev dependencies field.
  + Now imagine that we wanted to use something like jQuery here
  + then we can also go ahead and install it.
  + So let's say "npm install jquery"
  + but this time we only say "--save".
* ‘npm install’ will install all the dependencies and dec dependencies in package .json file
* Npm uninstall jquery –save will uninstall jquery from the node module
* ‘npm install packagename –global’ to install a package globally

Installing and configuring Webpack

* webpack is the most commonly used asset bundler.
* So it doesn't actually not only bundle
* JavaScript files or modules, but bundles all kinds of assets
* like JavaScript, CSS, and images.
* Create webpack.config.js
* in webpack there are four core concepts:
* it's the entry point, the output, loaders, and plugins.
* the entry point is where webpack will start the bundling. So basically, this is the file where it will start looking for all the dependencies which it should then bundle together
* import node package ‘path’ to use the absolute path
* const path = require(‘path’);
* path:path.resolve(\_\_dirname, 'dist')
* resolve function in path package will join the dir to dist/js

In webpack four we now have something called the production and the development mode.

So the **development mode** simply builds our bundler without minifying our code

in order to be as fast as possible. But the **production mode** will automatically enable all kinds of optimization, like minification and tree shaking

in order to reduce the final bundle size.

Synatx :

: mode:'devolopement'

* Syntax for exporting
* export default 23;
* Syntax for importing form anther module
* import num from './test.js';

change test in the sript property of package.json file to

"dev": "webpack"

So that the package will dierectly call the webpack to automatically look for the file and bundle it.

 install the webpack command line interface. So right now we only have webpack,

but we need something to be able to access it through the command line interface.

And so there's another package for that one.

Command to download webpack command line interface

npm install webpack cli --save-dev

Command to run the npm script

npm run dev

this will tell the webpack to look the js files and bundle.js file wont be automatically updated.

Once we are ready, we need to change the mode from ‘development’ to ‘production’ mode.

For this we make a change in the scripts.

"scripts": {

"dev": "webpack --mode deveplement",

"build" : "webpack --mode production"

So the code will be compressed after the development mode.

So if we want to run the server on development mode

npm run dev

And if we want to run the server on production mode

Npm run build

Webpack dev server

a bit easier when writing JavaScript we're going to add the webpack dev server to our setup in order to automatically reload the page when we save our code.

It will automatically upadatethe page whenever we are making any change in the javascript file inturn to bundle.js file.

Command:

npm install webpack-dev-server –-save-dev

It will save as a dev dependency

Setting up Webpack-dev-server

In webpack.config file add one property

devServer: {

contentBase: './dist'

}

So content base is the place where in here we will specify the folder from which webpack should serve our files. And in this case that is the distribution folder. So dist, so this folder here, and that is because basically this is here the code that we're going to ship to the client. So our app, all of the final code of that app, is here inside of this distribution folder, so we will always have an index file with the html, then the JavaScript, image, and CSS, and so as I explained in the beginning the source folder here is only for our development purposes, so all our source code basically goes here. Which then gets compiled or bundled into this distrubution folder as bundle.jsand so what we really want to serve is only what's in this folder here.

In package.json file add ‘start’ property to script

"scripts": {

"dev": "webpack --mode development",

"build": "webpack --mode production",

"start": "webpack-dev-server --mode development --open"

},

‘--open’ is to specify that the file should be open in background and make the changes automatically.

It is now running on a real webserver, it's a local webserver. So its local host on the port but it still, it's simulating a real http server,

HTML WEB PACK PLUGIN

To inject our script to the source code and to remove the index.html file from the dist folder:

Remember a webpack we have entry point, output and plugins. We alreader specified the entry point and output in webpack.config file and now we need to add the plugins.

plug-ins allow us to do complex processing of our input files, and in this case of our index.html file. So we want to use a plug-in called html webpack plug-in, and in order to use it, once again, you can probably guess it already, we have to install it.

npm install html-webpack-plugin -–save-dev

in turn we need to import to webpack.config file from the node modules

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

plug-ins basically receives an array of all the plug-ins that we are using.

Addin plugins property to webpack.config

plugins : [

new HtmlWebpackPlugin({

filename :'index.html'

template :'./src/index.html'

})

]

Here HtmlWebpackPlugin is more like a afuntion conatructor, that the reason for new operator on calling it. ‘filename’ and ‘template’ are the arguments passed as object properties to HtmlWebpackPlugin function constructor. Filename is the file html file in dist folder and template is the source code in src folder.

Even after loading the site we wont see the index.html file generated by the html-webpack-plugin, since it doesn not save it on the disc. Is not really visible right now,because it's not saving this data on the disk.

Inorder to see the html file in the dist folder run dev or build in command, it will bundle and save the html file in the dist folder.

Setting up BABEL

Babel is a javascript compiler. Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments. Here are the main things Babel can do for you:

Packages needed to install:

npm install babel-core babel-preset-env babel-loader --save-dev

which will install all these packges together. babel preset env, and this is a babel preset that will take care, that all the modern JavaScript features are converted back to ES5.

concept of loaders in Webpack.

So, loaders in Webpack allow us to import or to load all kinds of different files. And more importantly, to also process them. Like converting SASS to CSS Code or covert ES6 code to ES5 JavaScript.

Syntax for loaders(in webpack.config):

(The most weired syntax)

module:{

rules : [

{

test: /\.js$/,

exclude: /node\_modules/,

use: {

loader:'babel-loader'

}

}

]

}

Exclude will exclude the babel from applying to all node\_module files.

Test will test whether the file is .js file.

Now we need a config file for babel.

Create a file named .babelrc in the main folder.

{

"preset" :[

"env", {

"target":{

"browsers":[

"last 5 versions",

"ie >= 8"

]

}

}

]

}

a preset here is a collection of code transform plug-ins, which are like the pieces of code that actually apply the actual transformations to our code.

And what this will do is that Babel automatically figures out which ES6 features it actually needs to convert in order to work on the last five versions of all the browsers. And so that is what it means with environment.

Now, there are some things that we cannot really convert because they simply were not present in the ES5 version of the language. And so there is no way that we can simply convert them back from ES6 to ES5. And so these need to be pollyfilled.

Command to install poltfil

npm install babel-polyfil –save

And so this is not really a development tool, but it is really code that is going to go into our final bundle And so this is not a devDependency, but it's a real dependency.

Make the correscponding change in the entry property in webpack.json file

So basically installing babel was a three step process. First, we installed all of these packages and then added this rule here with the babel loader. Second, we created then a config file here, in order to tell Babel which stuff we want to convert back to ES5. And then finally, we also included a polyfill in order to not convert, but polyfill the features that we cannot convert with the babel loaders.