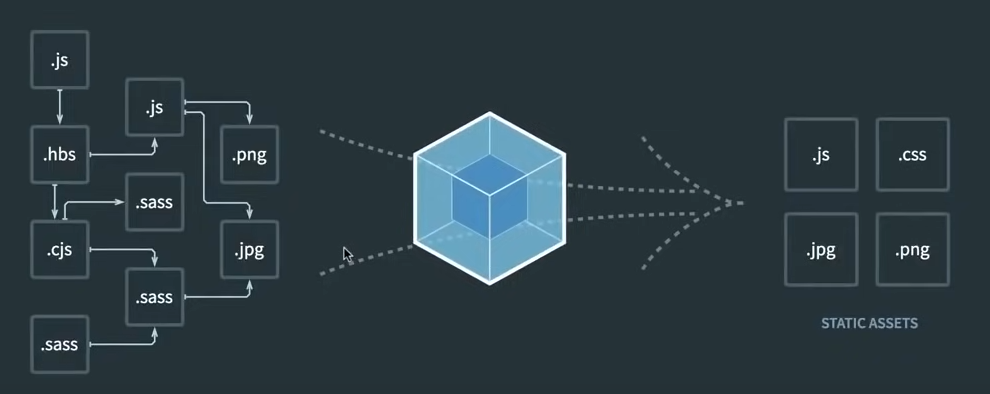
# What is Webpack

## Video Link: https://www.youtube.com/watch?v=MpGLUVbqoYQ

## What webpack does

It bundle our code/assets together

It also manages dependencies. If you read a file that depends on three other those three need to be included first.

This line of code:

\_\_webpack\_require\_\_(\*/! react 8/ ……

it is the way how webpack keep track of what is imported where, what is exported where…. basically manages the dependencies. It replace import and export with **webpack\_require** .

If we don’t use webpack we have to add dependencies on the required order like:

<script src = “./src/app/alert.service.js”></script>

<script src = “./src/app/component.service.js”></script>

<script src = “./src/app/utils/inputs-are- valid.js”></script>

<script src = “./src/app/app.js”></script>

</html>

## Install webpack

$ npm install –save-dev webpack webpack-cli

Then in the **package.json** in the **“script”**  we shoul add this line for the webpack to run on the **npm start** :

“script” : { “start” : “webpack”}

***Note*** : There should be an **index.js** in the **./src** file for webpack to start with.

After webpack done his work the **main.js** file will be produce in the **dist** folder.

***Note:*** We should add **dist** folder to **.gitignore**, because it will be created by using **npm start**.

## Depending files together

For depending files together then have a overall **main.js** file we should use **ES6** methods, **Export** and **Import**, after that we have to import main function in the **index.js** and run it.

By using webpack we can only use the main script in the **html** file :

<script src = “./dist/main.js”></script>

</html>

# Webpack config

For confing Webpack we should create a file that is named: **webpack.config.js**  and in the file:

const path = require(“path”);

module.exports = {

mode: “development”, //The main.js file will be not //minified anymore;

devtool: “none”,

entry: “./src/index.js”, //The file whole project start

output:{

filename: “main.js”,

path: path.resolve(\_\_dirname, “dist”)//It will create a //dist filt in the root

},

}

**“devtool: none”:** It will change the **eval(…)** function in the **main.js** file for more readability;

In the **package.json** file we should mention the config file:

{

.

.

.

“script”: {

“start” : “webpack –config webpack.config.js”

}

}

This line will show the config js in the npm start.

# Webpack loaders

## CSS Loader

These packages are used for loading css in webpack: **css loader** & **style loader**.

**css-loader** will turn a css file to the javasript and **style-loader**  will take this jacascript code and inject it into the DOM by a **<style>** tag.

At first we should install packages:

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

For adding css in our app, we should confing a css loader in our **config.js** file.

module.exports = {

module: {

rules: [

{

test: /\.css$/, //It’s regex and **$** means that it //has to end with css

use: [“style-loader”, “css-loader”] //The //arrangemant should be like the above

}

]

}

}

After these configs we should create a css file then import it in the file.

## Sass loader

For using sass in our document we need to install a sass loader:

npm install –-save-dev sass-loader node-sass

Then in the config :

module.exports = {

module: {

rules: [{

test: /\.scss$/,

use: [

“style-loader”,//3. Inject styles into DOM

“css-loader”, //2. Turns css into commonjs

“sass-loader” //1. Turns sass into css

]}]

}

}

# Cash busting & plugins

## Cash busting

It’s how we prevent certain assets like our main.js or css bundle from cashing by browser. For example when we change our code if it is cashed by browser, the changes will not be shown, because the cashed file will be used in the browser.

To prevent browser from cashing the old main.js file we should use **content-hash**. This variable is produced base on the content of the file.

For using content-hash and tell the browser not to use the cashed main.js, it should change this way: **main.ldfkjsflksjfklsjflkasjdlfk;js.js** the letters between main and js are content-hash

module.exports = {  
 mode: "development",  
 devtool: "none",  
 entry: "./src/index.js",  
 output: {  
 filename: "main.**[contentHash]**.js",  
 path: path.resolve(\_\_dirname, "dist"),  
 },  
 module: {  
 rules: [  
 {  
 test: /\.scss$/,  
 use: [  
 "style-loader", // 3. Inject styles into DOM  
 "css-loader", // 2. Turns css into commonjs  
 "sass-loader", // 1. Turns sass into css  
 ],  
 },  
 ],  
 },  
};

After run **npm start** the **main.js** will change to **main.sdl;sf;lsjf;safjs[gsgsgsgs.js**, and if we don’t change anything, the file name will be the same.

After using hashmethod the script in the html file:

<script src="./dist/main.js"></script>

The src above should add dynamiclly new named **main.js** file, to fullfil this we don’t include this script by ourself , we are gonna have built in webpack html file, for this method we should use plugins.

## Plugins

Inside **webpack.js.org** plugins meaning is:

“The plugins option is used to customize the webpack build process in a variety of ways.”

The plugin that we need is **HtmlWebpackPlugin**.

### HtmlWebpackPlugin

The [HtmlWebpackPlugin](https://github.com/jantimon/html-webpack-plugin) simplifies creation of HTML files to serve your webpack bundles. This is especially useful for webpack bundles that include a hash in the filename which changes every compilation. You can either let the plugin generate an HTML file for you, supply your own template using [lodash templates](https://lodash.com/docs#template), or use your own [loader](https://webpack.js.org/loaders).

1. To install:

npm install –save-dev html-webpack-plugin

1. In the **webpack.config.js**:

const path = require("path");

const HtmlWebpackPlugin = require("html-webpack-plugin");  
  
module.exports = {  
 mode: "development",  
 devtool: "none",  
 entry: "./src/index.js",  
 output: {  
 filename: "main.[contentHash].js",  
 path: path.resolve(\_\_dirname, "dist"),  
 },  
 plugins: [new HtmlWebpackPlugin()],  
 module: {  
 rules: [  
 {  
 test: /\.scss$/,  
 use: [  
 "style-loader", // 3. Inject styles into DOM  
 "css-loader", // 2. Turns css into commonjs  
 "sass-loader", // 1. Turns sass into css  
 ],  
 },  
 ],  
 },  
};

1. If we want our personal HTML file add to the html file that is produced by this package, we should create a **template.html** file and put our code inside of it then add it’s address into the **HtmlWebpackPlugin()** :

const path = require("path");  
const HtmlWebpackPlugin = require("html-webpack-plugin");  
  
module.exports = {  
 mode: "development",  
 devtool: "none",  
 entry: "./src/index.js",  
 output: {  
 filename: "main.[contentHash].js",  
 path: path.resolve(\_\_dirname, "dist"),  
 },  
 plugins: [  
 new HtmlWebpackPlugin({  
 template: "./src/template.html",  
 }),  
 ],  
 module: {  
 rules: [  
 {  
 test: /\.scss$/,  
 use: [  
 "style-loader", // 3. Inject styles into DOM  
 "css-loader", // 2. Turns css into commonjs  
 "sass-loader", // 1. Turns sass into css  
 ],  
 },  
 ],  
 },  
};

1. In our html file we don’t need these lines anymore, because webpack will take care of that:
2. The link to bootstrap (if we use bootstrap);
3. The <script> import of the main.js;
4. The svg import.
5. The output file will be in our **output.path** directory which in here is **/dist** and the name of file is **index.html**.

# Splitting Dev and Production

## Split files

For split development confing and production config, we should create a file for each of them and a common file for common configs of development mode and production mode.

To add common file to both of configs we need a package called **webpack-merge**

npm install –-save-dev webpack-merge

### Development config

for development create a file with a name that you want, here **webpack.dev.js**

const path = require("path");

const common = require("./webpack.common");

const merge = require("./webpack-merge");

module.exports = merge(common,{  
 mode: "development",  
 output: {  
 filename: "main.js",  
 path: path.resolve(\_\_dirname, "dist"),  
 },  
})

### Production mode

for production create a file with a name that you want, here **webpack.prod.js**

const path = require("path");

const common = require("./webpack.common");

const merge = require("./webpack-merge");

module.exports = merge(common,{  
 mode: "production",  
 output: {  
 filename: "main.[contentHash].js",  
 path: path.resolve(\_\_dirname, "dist"),  
 },

})

### Common mode

const path = require("path");

const HtmlWebpackPlugin = require("html-webpack-plugin");  
  
module.exports = {  
 entry: "./src/index.js",  
 plugins: [  
 new HtmlWebpackPlugin({  
 template: "./src/template.html",  
 }),  
 ],  
 module: {  
 rules: [  
 {  
 test: /\.scss$/,  
 use: [  
 "style-loader", // 3. Inject styles into DOM  
 "css-loader", // 2. Turns css into commonjs  
 "sass-loader", // 1. Turns sass into css  
 ],  
 },  
 ],  
 },

## Inside package.json

1. We can setup a dev server so we don’t need to **npm start** every time that we make a change

to do that we should install :

npm install –save-dev webpack-dev-server

1. Inside **package.json** file :

{  
{  
 “scripts”: {  
 “start”:  
 "webpack-dev-server --config webpack.dev.js --open", //--open will open the browser after npm start  
 “build”: "webpack --config webpack.prod.js",  
 },  
};