**Webpack**

webpack é um empacotador de módulo JavaScript de código aberto. Ele é feito principalmente para JavaScript, mas pode transformar ativos de front-end, como HTML, CSS e imagens se os carregadores correspondentes forem incluídos. Webpack pega módulos com dependências e gera ativos estáticos que representam esses módulos.

Dentro da pasta principal do projeto, você precisa instalar o pacote “webpack” e “webpack-cli” com o npm, e o arquivo padrão do webpack é o arquivo “webpack.config,js”, dentro do arquivo “package.json”, você configura o script “start” como: “start”: “webpack”, para que a hora que você rodar o comando “npm start” no seu terminal/cmd dentro da pasta do projeto, ele execute o arquivo de configuração do webpack, e dentro do arquivo de configuração do webpack você deve exportar as configurações dentro de um objeto especificando o modo(development(desenvolvimento) ou production(produção)) e a entrada(o arquivo que será executada ao rodar o webpack), exemplo:

const webpack = require('webpack');

module.exports = {

mode: 'development',

entry: './src/principal.js'

}

A saída padrão do webpack é o arquivo “main.js” dentro da pasta “build”, caso queira mudar a saída do webpack, basta colocar nas configurações dele:

const webpack = require('webpack');

module.exports = {

mode: 'development',

entry: './src/principal.js',

output: {

filename: 'principal.js',

path: \_\_dirname + '/public'

}

}

Onde “filename” vai ser o nome do arquivo retornado, e “path” o caminho da pasta onde o arquivo será retornado. Para carregar arquivos “CSS” pelo webpack, primeiramente você precisa instalar os pacotes “style-loader” e “css-loader” com o npm, e após ter instalado os pacotes, basta colocar os modulos de CSS no arquivo de configuração do webpack:

const webpack = require('webpack');

module.exports = {

mode: 'development',

entry: './src/principal.js',

output: {

filename: 'principal.js',

path: \_\_dirname + '/public'

},

module: {

rules: [{

test: /\.css$/,

use: [

'style-loader', *// Adiciona CSS a DOM injetando a tag <style>*

'css-loader' *// Interpretar @import, url()...*

]

}]

}

}

Onde “style-loader” vai adicionar o CSS ao DOM da página, para quando ser criado um arquivo “.html” e linkar o arquivo “.js” criado pelo webpack(que no exemplo é o “principal.js”), todas as propriedades CSS já vão estar no arquivo, e serão adicionados na página, e o “css-loader” serve para interpretar @import, url e outras características do CSS. Caso queira criar um arquivo “.css” externo, basta instalar o pacote “mini-css-extract-plugin” com o npm, e dentro das configurações do webpack colocar:

const webpack = require('webpack');

const MiniCssExtractPlugin = require('mini-css-extract-plugin');

module.exports = {

mode: 'development',

entry: './src/principal.js',

output: {

filename: 'principal.js',

path: \_\_dirname + '/public'

},

plugins: [

new MiniCssExtractPlugin({

filename: 'estilo.css'

})

],

module: {

rules: [{

test: /\.css$/,

use: [

MiniCssExtractPlugin.loader,

'css-loader' *// Interpretar @import, url()...*

]

}]

}

}

Após ter ter configurado o plugin do “mini-css-extract-plugin” colocando o nome do arquivo que será gerado junto da extensão(.css), na parte de modules tem que remover o “style-loader” ou comentar ele para não ter conflito, e após isso basta rodar o “npm start” e linkar o arquivo “.css” no HTML e estará sempre sendo atualizado quando o script do webpack for executado pelo “npm start”. Para que você possa pegar um arquivo “SASS”,”SCSS” ou “CSS” e gerar um arquivo “.css”, basta instalar os pacotes “node-sass” e “sass-loader”, e depois colocar nas configurações do weback:

const webpack = require('webpack');

const MiniCssExtractPlugin = require('mini-css-extract-plugin');

module.exports = {

mode: 'development',

entry: './src/principal.js',

output: {

filename: 'principal.js',

path: \_\_dirname + '/public'

},

plugins: [

new MiniCssExtractPlugin({

filename: 'estilo.css'

})

],

module: {

rules: [{

test: /\.s?[ac]ss$/,

use: [

MiniCssExtractPlugin.loader,

'css-loader'*, // Interpretar @import, url()...*

’sass-loader'

]

}]

}

E agora ele vai procurar por todos os arquivos que terminem com “SS” como “CSS”, “SASS”, “SCSS”, etc. Para que possa configurar se vai ser ambiente de desenvolvimento ou produção, através do comando “npm start”(para desenvolvimento), e “npm run build”(para produção), primeiramente deve-se configurar nos scripts do arquivo “package.json” o build, colocando da seguinte forma:

"scripts": {

"start": "webpack",

"build": "cross-env NODE\_ENV=production webpack"

}

Após ter configurado a variável de ambiente no script do arquivo “package.json”, deve se instalar os seguintes pacotes com o npm:

cross-env,  
optimize-css-assets-webpack-plugin,  
uglifyjs-webpack-plugin

após ter instalado esses pacotes, basta configurar o arquivo de configuração do webpack:

const modoDev = process.env.NODE\_ENV !== "production";

const webpack = require('webpack');

const MiniCssExtractPlugin = require('mini-css-extract-plugin');

const UglifyJsPlugin = require('uglifyjs-webpack-plugin');

const OptimizeCSSAssetsPlugin = require('optimize-css-assets-webpack-plugin');

module.exports = {

mode: modoDev ? 'development' : 'production',

entry: './src/principal.js',

output: {

filename: 'principal.js',

path: \_\_dirname + '/public'

},

optimization: {

minimizer: [

new UglifyJsPlugin({

cache: true,

parallel: true

}),

new OptimizeCSSAssetsPlugin({})

]

},

plugins: [

new MiniCssExtractPlugin({

filename: 'estilo.css'

})

],

module: {

rules: [{

test: /\.s?[ac]ss$/,

use: [

MiniCssExtractPlugin.loader,

*// 'style-loader', // Adiciona CSS a DOM injetando a tag <style>*

'css-loader', *// Interpretar @import, url()...*

'sass-loader',

]

}]

}

}

Fazendo assim com que quando rode o comando “npm start”, ele vai para o modo de desenvolvimento, e quando rodar o “npm run build” vai para o modo de produção.

**jQuery**

jQuery é uma biblioteca de funções JavaScript que interage com o HTML, desenvolvida para simplificar os scripts interpretados no navegador do cliente. Usada por cerca de 74.4% dos 10 mil sites mais visitados do mundo, jQuery é a mais popular das bibliotecas JavaScript.

O seletor padrão do jQuery é o símbolo de dólar($), mas também pode ser invocado com a palavra “jQuery”. O jQuery usa seletor CSS, e uma das formas de chamar o jQuery é como seletor, como função, ou como objeto usando a notação ponto.

Exemplo:

$(‘div’).hide(5000);

No exemplo anterior, você estará selecionando todas as “divs” da página, e a função “.hide(5000)” vai fazer com que o elemento selecionado se esconda em 5 segundos, diminuindo seu tamanho até desaparecer.

$(‘div’).show(5000);

O “.show(5000)” vai fazer o oposto do “.hide(5000)”, vai fazer o elemento crescer até chegar ao seu tamanho original fazendo ele aparecer na página em 5 segundos também.

$(‘div’).fadeOut (5000);

O “.fadeOut(5000)” vai fazer praticamente a mesma coisa que o “.hide()”, porém em vez de fazer com que o elemento diminua de tamanho para ir desaparecendo, ele vai sumindo gradualmente da página. E você também pode encadear um após o outro, exemplo:

$(‘div’).hide(5000).show(5000).fadeOut (5000);

Fazendo assim, com que quando um acabe o outro já se inicia em seguida. Quando você seleciona vários elementos com o jQuery, como no caso selecionando uma “div” no exemplo, para você pegar em uma posição especifica, você usa a função “.get()”, exemplo:

$(‘div’).get(0);

O jQuery vai selecionar o primeiro elemento da lista que é retornada para ele pela função seletora. Você pode verificar se um objeto é vazio usando a função “.isEmptyObject()”, exemplo:

$.isEmptyObject({});

Aqui ele vai retornar true, pois o objeto está vazio.

.isEmptyObject({nome: null});

Aqui ele vai retornar false , pois o objeto não está vazio. Você pode também verificar se um elemento está contido dentro de um Array utilizando o método “.inArray()”, exemplo:

jQuery.inArray(3, [1, 2, 3]);

Aqui ele vai retornar -1 caso o elemento não esteja contido dentro do Array, ou caso o elemento esteja contido dentro do Array, ele vai retornar o próprio índice do elemento.

Com o jQuery, tem como você criar o seu próprio plugin, exemplo:

$.fn.fundoVerde = function () {

this.css(‘background-color’, ‘green’);

return this;

}

$(‘body’).fundoVerde().fadeOut(1000);

Nesse exemplo, criamos uma função “fundoVerde”, que com o seletor “.css” usa a propriedade “background-color” e altera o fundo para verde, e no fim atribui a função para o “body” da página, e depois usa o encadeamento com o “.fadeOut” para esconder a página, e para fazer o encadeamento de funções do jQuery é necessário retornar o “this(o próprio objeto)”.