1. Webpack : Du Typescript au navigateur

Webpack est un compilateur de modules dont la sotrie est un paquet – ou *Bundle - qui* comprendra du Javascript, du css, du html et d’autres formats comme des images ou du JSON.

Le fonctionnement de Webpack passe par le parcours des fichiers de scripts à la recherches des imports (méthode définie dans l’ECMASCRIPT 2015, voir notre article dédiée).

Cette documentation a pour intêret de se focaliser sur l’utilisation de webpack au seing d’un framework Javascript du type Angualr ou React.

Installation

Installer webpack en global car cette dépendance pourra être utilisée de l’ensemble es projets :

**npm install webpack -g**

Commencer en douceur

**Nous allons explorer les fonctionnalités de Webpack en y intégrant la notion de loader . Un loader est une librairie donnant à Webpack des foncitonnalités supplémentaire, dans l’example ci-dessous, nous allons ajouter un fichier css et, à l’aide de Webpack, l’intégere dan snoter fhcier html sans modifier ce dernier.**

Voici quelques étapes très pour une première génération avec le loader css :

1. Dans un répertoire de travail (ici, SRC), créer un fichier au format ts (ici, main.ts) avec le contenu ci-dessous.

main() {

console.info(`main executé`);

}

main();

1. Créer un fichier style.css avec le contenu ci-dessous.

.All-red {

background-color : red

color : white ;

}

1. Afin que webpack sache identifier le css et l’injecter dans le fichier html, il faudra ajouter le loader *style-loader*

*npm install style-loader --save-dev*

1. Créer un fichier **webpack.config.ts** afin d’indiquer au compilateur de récupérer le css puis de l’injecter :
2. ...
3. Lancer la commande ci-dessous afin de compiler le typescript en javascript et de l’executer

Webpack ./main.ts main.js && node ./main.js

**Aller plus loin**

1. Modifier le fchier main.ts afin d’y intégrer du typescript :

export class main {

constructor() {

const target = 'JS'

console.info(`En route vers le ${target}`);

}

}

1. Créer un fichier index.html avec le contenu suivant :

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<div class="all-red">

<span>Content</span>

</div>

<script type="text/javascript" src="bundle.js" charset="utf-8"></script>

</body>

</html>

1. Afin que webpack sache identifier le css et l’injecter dans le fichier html, il faudra ajouter le loader *style-loader*

*npm install style-loader --save-dev*

1. Faire évoluer le contenu du webpack.config.js afin que le compilateur sache identifier le ts d’entrée, le javascript sible et le loader css :

module.exports = {

devtool: 'inline-source-map',

entry: "./main.ts",

context: \_\_dirname, // identifier tsconfig.json automatiquement

output: {

filename: "bundle.js"

},

module: {

rules: [

{ test: /\.tsx?$/, // ts & tsx gérés

loader: 'ts-loader',

options: {

transpileOnly: true

}

},

]}

};

1. Enfin, ajouter le fichier config.js qui permet de spécifier des paramètres de compilation et le répertoire racine de par sa localisation :

{

"compilerOptions": {

"sourceMap": true

}

}

1. Executer *webpack*

*Il est alors intéressant d’ouvrir le fichier js et de constater la syntaxe du fichier généré en sortie.*

(function(module, exports, \_\_webpack\_require\_\_) {

"use strict";

exports.\_\_esModule = true;

var main = /\*\* @class \*/ (function () {

function main() {

var target = 'JS';

console.info("En route vers le " + target);

}

return main;

}());

exports.main = main;

/\*\*\*/ })

Nous constatons les points suivants :

* Le codeses tjoué dans une closure en execution IIFE
* la chaine "use strict"; afin de jouer le script en mode strict (variables forcément déclarées, tolérance moindre avec eval…)
* la classe devient une fonciton IIFE
* Les variables deviennent des vars