# Webpack & Babel Introduction

Vous pouvez consulter les documentations officielles des outils suivants :

- npm
- webpack
- babel

Attention lisez la remarque qui suit pour la compatiblité des dépendances :

The minimum supported Node.js version to run webpack 5 is 10.13.0 (LTS)

## Introduction

Webpack permet de packager ou regrouper des modules Javascripts. Il transforme également les assets front-end SASS ou images par exemple, à l'aide de loaders en assets statiques.

Webpack permet de construire une application moderne. La logique interne de Webpack est basée sur ES5, mais vos scripts ES6 ne seront pas transpilés en ES5.

Babel est pour sa part un transpiler (traducteur de code). Nous pouvons l'utiliser avec Webpack pour transpiler notre code ES6 en ES5 par exemple.

## Exemple d'introduction

1. Créez un dossier tests et les deux fichiers suivants dans le dossier assets : logs.js et app.js. Créez également le fichier index.html et importez le fichier app.js, vous lancerez Go live pour votre fichier index.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Je suis un test</title>
</head>
<body>
    <div id="main"></div>
    <script src="./assets/app.js"></script>
</body>
</html>
```

2. Dans le fichier logs.js exportez la fonction log suivante :

```
module.exports = function (message){
    console.log(message);
}
```

Et finalement importez la à l'aide d'un require dans le fichier app.js

```
const log = require('./logs.js');

log("message");
```

Que constatez-vous sur la page index.html, il y a effectivement une erreur. Le navigateur ne connait pas la fonction require. Webpack, nous allons le voir, va préparer un fichier bundle.js pour fusionner les fichiers app.je et logs.js afin qu'ils puissent être correctement interprétés par le navigateur.

Ins[illegible]n de Webpack. Créez tout d'abord le fichier package.json dans le dossier tests :

```
npm init -y

npm install --save-dev webpack
```

# Installation/configuration de l'exercice

Webpack possède un fichier de configuration : **webpack.config.js** que nous allons utiliser dans nos cours/exercices :

- Créez la structure d'un projet simple :

```
dragons/
    dist/
        index.html
    src/
        core/
        app.js
    webpack.config.js
```

Lancez la configuration d'un projet géré avec **npm**. A la racine de dragon, tapez la ligne de commande ci-dessous. Elle crée un fichier de gestion et configuration nommé package.json :

```
npm init -y
```

- Puis installez **webpack** & **webpack-cli** en tant que dépendance (flag -D) de votre projet :

```
npm install --save-dev webpack webpack-cli
```

Notez que si vous voulez désinstaller un module en mettant à jour le fichier package.json, vous taperez la ligne de commande suivante à la racine de votre projet :

```
npm uninstall --save-dev webpack-cli
```

Après cette modification, ajoutez la clé/valeur build : "webapck" dans la partie scripts du fichier package.json; ceci nous permettra d'appeler simplement webpack en ligne de commande à l'aide de npm, voyez l'exemple ci-après :

```
{
  "name": "dragons",
  "version": "1.0.0",
  "description": "",
  "main": "webpack.config.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^5.2.0",
    "webpack-cli": "^4.1.0"
  }
}
```

Dans la console tapez maintenant :

```
npm run build
```

# Configuration simple du fichier webpack

Exemple de configuration simple du fichier webpack.config.js. Le fichier app.js est le point d'entrée de notre application front-end. Nous importerons toutes les dépendances dans ce fichier. Le fichier **bundle.js** est le build de webpack, c'est ce dernier que l'on importera dans l'index.html :

```
module.exports = {
    entry: './src/app.js', // Point d'entrée
    // Sortie (fichier de build)
    output: {
        filename: "bundle.js"
    }
}
```

Pour construire le build de l'application, tapez la commande suivante :

```
npm run build
```

Vous devriez avoir maintenant un dossier dist dans l'application dans lequel webpack écrit son build.

## Meilleure configuration du fichier Webpack

On utilise le module path pour gérer les chemins de build. Ce module possède une méthode resolve, on lui passe deux paramètres : __dirname et le nom du dossier de build. Cette configuration est plus rapide et vous permettra de modifier éventuellement le dossier de build.

Voyez le code du fichier index.html ci-après, nous y importons le fichier de build de webpack.

La clé watch demande à Webpack de refaire le build dès qu'un fichier du projet est modifié.

```
const path = require('path');

module.exports = {
    mode : "development",
    watch : true,
    entry: './src/app.js', // Point d'entrée
    // Sortie
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: "bundle.js"
    }
}
```

Ficher index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dragons</title>
</head>
<body>
    <script src="bundle.js"></script>
</body>
</html>
```

# Installation & configuration d'un serveur local

Nous allons utiliser le server de développement de webpack. Il nous permettra de mettre à jour automatiquement la page Web dès qu'un fichier sera modifié. Notons que sa configuration est relativement simple.

## Exemple simple de configuration

Créez un dossier webpackExample.

1. Lancez à la racine du dossier (initialisation du projet) :

```
npm init -y
```

2. Créez les dossiers et fichiers suivants :

    x.html mettez du html:5 et insérez une balise div#main dans le body

- src/utils.js

- app.js
- webpack.config.js

```
module.exports = {
    watch : true,
    entry: 'app.js', // Point d'entrée
    // Sortie
    output: {
        filename: "bundle.js"
    }
}
```

```
npm install --save-dev webpack webpack-cli
npm install --save-dev webpack-dev-server
```

Ajoutez maintenant le code suivant dans le fichier package.json et vérifiez que vous avez bien la version v4 au moins de webpack-cli.

```
"scripts": {
    "start": "webpack serve"
  },
```

3. Exportez maintenant une constante message, elle contiendra la chaîne de caractères "Hello Webpack", depuis le fichier utils.js dans le fichier app.js.

Affichez ce message dans la page index.html dans un titre de niveau 1, pensez à lancer votre serveur :

```
npm run start
```

# Configuration plus technique du fichier webpack

Dans la configuration suivante vous trouverez :

- watch : pour lancer l'observation automatique des changements dans vos fichiers du projet. Ici vous n'en avez plus besoin en utilisant le server de webpack.
- mode development : aucun fichier de build ne sera créé (optimisation). Si vous avez besoin d'un build nous vous indiquons la commande ci-après à ajouter dans le fichier package.json
- output : nous désignons ici un autre dossier pour créer le fichier de build, nous sommes obligés d'utiliser le module path pour définir un chemin absolu.
- devServer : on définit le point d'entrée pour le serveur, le port, le lancement automatique du navigateur et le reload automatique.

```
// pour créer un chemin absolu pour webpack
const path = require('path');

module.exports = {
    // watch: true, // inutile avec le server webpack
    // précise que l'on est en mode développement
    mode : "development",
    entry: './src/app.js', // Point d'entrée
    // Sortie
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: "bundle.js"
    },
    // Configuration de webpack-dev-server minimale
    devServer: {
        contentBase: path.join(__dirname, 'dist'),
        port : 9000,
        open: true,
        hot : true // reload automatique
    }
}
```

Note : ue dans le script build nous précisons : le mode de développement et le fichier de configuration en argument de cette commande.

```
"scripts": {
    "start": "webpack serve",
    "build": "webpack --env NODE_ENV=production --config webpack.config.js"
},
```

Avec l'option de variable d'environement pensez à ré-écrire votre fichier webpack comme suit :

```
// pour créer un chemin absolu pour webpack
const path = require('path');

module.exports = env => {

    console.log(env.NOD_ENV)

    return {
        mode : "development",
        entry: './src/app.js', // Point d'entrée
        // Sortie
        output: {
            path: path.resolve(__dirname, 'dist'),
            filename: "bundle.js"
        },
        // Configuration de webpack-dev-server minimale
        devServer: {
            contentBase: path.join(__dirname, 'dist'),
            port : 9000,
            open: true,
            hot : true // reload automatique
        }
    }
}
```

Modifié le: vendredi 22 septembre 2023, 13:06