TABLE DES MATIÈRES

TABLE DES MATIÈRES 1
INTRODUCTION 3
I. GÉNÉRALITÉS 4
1. Qu'est-ce que Webpack ? 4
2. Historique 4
3. Caractéristiques 5
a. Objectif 5
b. Concepts principaux autour Webpack 5
c. Points forts 7
d. Points faibles 8
4. Comment fonctionne Webpack ? 8
5. Quoi de neuf dans Webpack ? 8
6. Comparaison avec des outils similaires 10
a. Browserify 10
b. Grunt 10
c. Gulp 10
d. Rollup 10
e. Parcel 10
7. Pourquoi choisir Webpack ? 12
II. IMPLEMENTATION 12
1. Outils utilisés pour notre travail 12
2. Procédure de travail 12
3. Difficultés rencontrées 12
4. Captures d'écran et détails sur l'implémentation 12
CONCLUSION 14
RÉFÉRENCES 15
ANNEXE 17
1. Guide d'installation et de création de projets 17
2. Dictionnaire technique 17

No table of figures entries found.

INTRODUCTION

Le développement web fait référence au processus d'écriture d'un site ou d'une page web dans un langage technique. Il s'agit d'une étape incontournable pour qu'un contenu soit mis en ligne et atteigne ses lecteurs. 1 Avec le temps, bon nombre d'outils ont été mis sur pied pour faciliter ce processus, à l'instar de Webpack, qui fait l'objet de notre exposé. De ce fait, il sera question pour nous de présenter cet outil à travers quelques généralités, appuyées par une implémentation.

I. GÉNÉRALITÉS

Dans cette partie, il sera question pour nous de présenter quelques spécificités de l'outil Webpack. Il s'agira tout d'abord d'une formalisation via une définition, puis une présentation de son historique, puis nous donnerons quelques de ses caractéristiques (usage, point forts, points faibles), et enfin clôturerons cette partie avec une comparaison avec des outils similaires.

1. Qu'est-ce que Webpack ?

Webpack est un outil logiciel open-source de type « module bundler » (littéralement, « groupeur de modules »), conçu pour faciliter le développement et la gestion de sites et d'applications web modernes. 2 Dans un projet particulier, webpack traite tous les fichiers et actifs comme des modules. Sous le capot, il s'appuie sur un graphe de dépendances. Un graphique de dépendance décrit comment les modules sont liés les uns aux autres à l'aide des références (instructions require et import) entre les fichiers. De cette façon, webpack parcourt statiquement tous les modules pour construire le graphique et l'utilise pour générer un seul bundle (ou plusieurs bundles) - un fichier JavaScript contenant le code de tous les modules combinés dans le bon ordre. « Statiquement » signifie que, lorsque webpack construit son graphe de dépendances, il n'exécute pas le code source mais assemble les modules et leurs dépendances en un ensemble. Cela peut ensuite être inclus dans vos fichiers HTML. D

2. Historique

L'outil logiciel Webpack a été développé le 19 février 2014 par Tobias Koppers, Sean Larkin, Johannes Ewald, Juho Vepsäläinen, Kees Kluskens et quelques autres contributeurs. En effet, l'ensemble de la communauté des développeurs était impliqué dans une quête constante d'amélioration de l'expérience globale des utilisateurs et des développeurs autour de l'utilisation et de la création d'applications javascript/web. A Au cours de la dernière décennie, avec la popularisation rapide des téléphones mobiles, des tablettes mobiles et des appareils portables, le développement de l'interface Web est également passé du développement traditionnel de pages Web sur PC au développement multi-terminal. B Par conséquent, nous avons vu beaucoup de nouvelles bibliothèques et frameworks introduits.

Quelques modèles de conception ont également évolué au fil du temps pour offrir aux développeurs un moyen meilleur, plus puissant mais très simple d'écrire des applications JavaScript complexes. Avant, les sites Web n'étaient plus simplement un petit paquet contenant un nombre impair de fichiers. Ils ont déclaré devenir volumineux, avec l'introduction des modules JavaScript, car l'écriture de petits morceaux de code encapsulés était la nouvelle tendance. Finalement, tout cela a conduit à une situation où nous avions 4x ou 5x he de fichiers dans le package de candidature globale.

Non seulement la taille globale de l'application était un défi, mais il y avait aussi un énorme écart dans le type de code que les développeurs écrivaient et le type de code que les navigateurs pouvaient comprendre. Les développeurs devaient utiliser beaucoup de code d'aide appelé polyfills pour s'assurer que les navigateurs étaient capables d'interpréter le code dans leurs packages. Pour répondre à ces problématiques, webpack a été créé. C

3. Caractéristiques

a. Objectif

Cet outil permet de réaliser un certain nombre de tâches fastidieuses et répétitives liées au développement d'interfaces web (comme la gestion des dépendances, la compilation du code source, le déploiement d'applications sur des serveurs, etc.), de manière automatique. Il est initialement destiné au code JavaScript, mais sa forte modularité lui permet cependant de gérer beaucoup d'autres langages de programmation, notamment à l'aide de plugins tiers permettant d'étendre ses capacités. 3

b. Concepts principaux autour Webpack

Tout d'abord, il faut savoir que Webpack requiert un environnement avec Node.js version 10.13.0+ pour fonctionner. Pour comprendre cet outil, il faut comprendre les cinq notions de base qui gravitent autour de lui.

i. Point d'entrée

Un point d'entrée indique quel module webpack doit utiliser pour commencer à construire son graphe de dépendance interne. Webpack déterminera de quels autres modules et bibliothèques ce point d'entrée dépend (directement et indirectement). Par défaut, sa valeur est ./src/index.js, mais nous pouvons spécifier un point d'entrée différent (ou plusieurs) en définissant une propriété ***entry*** dans la configuration webpack. Par exemple:

webpack.config.js

```
module.exports = {
  entry: './path/to/my/entry/file.js',
};
```

ii. La propriété output

La propriété output indique à webpack où émettre les bundles qu'il crée et comment nommer ces fichiers. Il s'agit par défaut ./dist/main.js du fichier de sortie principal et du ./distdossier pour tout autre fichier généré. L'on peut configurer cette partie du processus en spécifiant un champ ***output*** dans notre configuration. Exemple :

webpack.config.js

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

module.exports = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'my-first-webpack.bundle.js',
  },
};
```

Dans l'exemple ci - dessus, nous utilisons les propriétés webpack output.filename et output.path c'est à dire le nom de notre paquet et où nous voulons qu'il soit mis.

iii. Les loaders

Prêt à l'emploi, webpack ne comprend que les fichiers JavaScript et JSON. Les loaders permettent à webpack de traiter d'autres types de fichiers et de les convertir en modules valides pouvant être consommés par une application et ajoutés au graphique de dépendance.

Note : L'une des caractéristiques spécifiques de webpack est la possibilité de faire des ***import*** n'importe quel type de module, par exemple des fichiers .css, qui peuvent ne pas être pris en charge par d'autres bundlers ou exécuteurs de tâches. Nous pensons que cette extension du langage est justifiée car elle permet aux développeurs de créer un graphe de dépendances plus précis.

À un niveau élevé, les loaders ont deux propriétés dans la configuration Webpack :

- La propriété ***test*** identifie le ou les fichiers à transformer
- La propriété ***use*** indique quel chargeur doit être utilisé pour effectuer la transformation

Nous avons par exemple :

webpack.config.js

```
-   const path = require('path');
-
-   module.exports = {
-     output: {
-       filename: 'my-first-webpack.bundle.js',
-     },
-     module: {
-       rules: [{ test: /\.txt$/, use: 'raw-loader' }],
-     },
    };
```

La configuration ci-dessus a défini une propriété ***rules*** pour un seul module avec deux propriétés requises : ***test*** et ***use***. Cela indique au compilateur de webpack ce qui suit :

"Hey compilateur webpack, lorsque vous rencontrez un chemin qui se résout en un fichier '.txt' à l'intérieur d'une instruction require()/ import, utilisez le raw-loader pour le transformer avant de l'ajouter au bundle."

Il est important de se rappeler que lors de la définition des règles dans la configuration webpack, on les définit sous ***module.rules*** et non ***rules***. Mais webpack nous avertira si cela est mal fait.

iv. Les plug-ins

Alors que les loaders sont utilisés pour transformer certains types de modules, les plugins peuvent être exploités pour effectuer un plus large éventail de tâches telles que l'optimisation des bundles, la gestion des actifs et l'injection de variables d'environnement. Pour utiliser un plugin, on aura besoin de ***require()*** et l'ajoutez au tableau *plugins*. La plupart des plugins sont personnalisables via des options. Étant donné que nous pouvons utiliser en utiliser un plusieurs fois dans une configuration à des fins différentes, nous devons en créer une instance en l'appelant avec l'opérateur ***new***. Par exemple:

webpack.config.js

```
const HtmlWebpackPlugin = require('html-webpack-plugin'); //installed via npm
```

```
const webpack = require('webpack'); //to access built-in plugins

module.exports = {
  module: {
    rules: [{ test: /\.txt$/, use: 'raw-loader' }],
  },
  plugins: [new HtmlWebpackPlugin({ template: './src/index.html' })],
};
```

Dans l'exemple ci-dessus, le ***html-webpack-plugin*** génère un fichier HTML pour votre application et injecte automatiquement tous vos bundles générés dans ce fichier.

v. *Le mode*

En définissant le paramètre ***mode*** sur *development*, *production* ou *none*, nous pouvons activer les optimisations intégrées de webpack qui correspondent à chaque environnement. La valeur par défaut est *production.* Par exemple :

```
module.exports = {
  mode: 'production',
};
```

c. Points forts

- Son principal avantage réside dans sa gestion extensible de toute une panoplie de modules, d'outils et de langages de programmation, ainsi que dans sa simplicité de mise en place, très adaptable et qui peut également être entièrement paramétrée pour répondre aux besoins spécifiques d'une application. 4

- Webpack prend en charge tous les navigateurs compatibles ES5 (IE8 et versions antérieures ne sont pas pris en charge). Webpack a besoin ***Promise*** de ***import()*** et ***require.ensure()***. Si nous souhaitons prendre en charge des navigateurs plus anciens, il suffit de charger un *polyfill* avant d'utiliser ces expressions.

- IIFE - Expressions de fonction immédiatement invoquées : Les IIFE résolvent les problèmes de cadrage pour les grands projets ; lorsque les fichiers de script sont encapsulés par un IIFE, nous pouvons les concaténer ou combiner en toute sécurité des fichiers sans se soucier de la collision de portée.

- Webpack s'exécute sur Node.js, un environnement d'exécution JavaScript qui peut être utilisé sur des ordinateurs et des serveurs en dehors d'un environnement de navigateur.

- ESM - Modules ECMAScript : De nos jours, les modules des projets web deviennent une fonctionnalité officielle de la norme ECMAScript.

- Collecte automatique des dépendances : Les anciens Task Runners et même Google Closure Compiler vous obligent à déclarer manuellement toutes les dépendances à l'avance. Alors que les bundlers comme webpack créent et déduisent automatiquement votre graphique de

dépendance en fonction de ce qui est importé et exporté. Ceci, avec d'autres plugins et chargeurs, offre une excellente expérience de développement G

d. Points faibles

- ESM - Modules ECMAScript : Il constitue un atout pour webpack. Cependant, la prise en charge du navigateur est incomplète et le regroupement est toujours plus rapide et actuellement recommandé par rapport à ces premières implémentations de modules

- Vitesses en mode Dev : Webpack doit regrouper tous les modules lorsque l'on démarre le serveur de développement. Pour cette raison, cela peut être très lent, prenant de 2 à 30 secondes généralement, voire jusqu'à 150, pour se terminer lorsque vous démarrez le serveur de développement. Pour accélérer les choses, les bundlers tels que Parcel utilisent la mise en cache, que Webpack a maintenant implémentée dans la version 5, mais n'est pas encore pris en charge dans tous les frameworks et plugins, et certains bundlers, comme Vite et Snowpack, ont réussi à devenir plus rapides, atteignant 250 ms temps de rechargement, en utilisant l'ESM natif

- Les fichiers de configuration sont extrêmement compliqués à comprendre

- Taille du paquet : Parce qu'il nécessite des polyfills pour charger les modules, la taille du bundle de Webpack peut être beaucoup plus grande que celle des certains autres bundlers H

4. Comment fonctionne Webpack ?

Même un simple projet contient des fichiers HTML, CSS et JavaScript. En outre, il peut contenir des éléments tels que des polices, des images, etc. Ainsi, un flux de travail Webpack typique comprendrait la configuration d'un fichier index.html avec les liens CSS et JS appropriés, ainsi que les ressources nécessaires. De plus, si l'on a de nombreux modules CSS et JS qui dépendent les uns des autres, ils doivent être optimisés et correctement combinés en une seule unité prête pour la production.

Pour faire tout cela, webpack s'appuie sur la configuration. À partir de la version 4 et des versions ultérieures, webpack fournit des valeurs par défaut raisonnables prêtes à l'emploi, il n'est donc pas nécessaire de créer un fichier de configuration. Cependant, pour tout projet non trivial, l'on devra fournir un fichier webpack.config.js spécial, qui décrit comment les fichiers et les actifs doivent être transformés et quel type de sortie doit être généré. Ce fichier peut rapidement devenir monolithique, ce qui rend difficile de comprendre comment webpack fait son travail à moins de connaître les principaux concepts derrière son fonctionnement.

Sur la base de la configuration fournie, webpack démarre à partir des points d'entrée et résout chaque module qu'il rencontre lors de la construction du graphe de dépendance. Si un module contient des dépendances, le processus est exécuté de manière récursive sur chaque dépendance jusqu'à ce que le parcours soit terminé. Ensuite, webpack regroupe tous les modules du projet en un petit nombre de bundles - généralement un seul - à charger par le navigateur. E

5. Quoi de neuf dans Webpack ?

Une version webpack 5 a été annoncée en octobre 2020. Le post est assez long et explore toutes les modifications apportées à webpack. Il est impossible de mentionner tous les changements depuis la

première version. Au lieu de cela, nous avons dressé une petite liste avec quelques points saillants généraux les plus récents et intéressants :

- Les performances de construction sont améliorées avec la mise en cache persistante. Les développeurs peuvent désormais activer un cache basé sur le système de fichiers, ce qui accélérera les versions de développement.

- La mise en cache à long terme est également améliorée. Dans webpack 5, les modifications apportées au code qui n'affectent pas la version réduite du bundle (commentaires, noms de variables) n'entraîneront pas d'invalidation du cache. En outre, de nouveaux algorithmes ont été ajoutés qui attribuent des ID numériques courts aux modules et aux morceaux et des noms courts aux exportations de manière déterministe. Dans webpack 5, ils sont activés par défaut en mode production.

- Taille de paquet améliorée, grâce à un meilleur Tree Shaking et à une meilleure génération de code. Grâce à la nouvelle fonctionnalité Nested Tree-Shaking, webpack est désormais capable de suivre l'accès aux propriétés imbriquées des exportations. Le CommonJs Tree Shaking nous permet d'éliminer les exports CommonJs inutilisés.

- La base de code est nettoyée. Tous les éléments marqués comme obsolètes dans le webpack 4 sont supprimés.

- -Les polyfills automatiques Node.js sont supprimés. Les versions précédentes de webpack incluaient des polyfills pour les bibliothèques Node.js natives comme crypto. Dans de nombreux cas, ils sont inutiles et augmentent considérablement la taille du paquet. C'est pourquoi webpack 5 arrête de polyremplir automatiquement ces modules de base et se concentre sur les modules compatibles front-end.

- En tant qu'amélioration du développement, webpack 5 nous permet de passer une liste de cibles et également de prendre en charge les versions de cible. Il permet la détermination automatique du chemin public. De plus, il offre un nommage automatique et unique, ce qui empêche les conflits entre plusieurs exécutions de packs Web qui utilisent la même variable globale pour le chargement de blocs.

- Des modules d'actifs sont introduits, qui remplacent les utilisations de file-loader, raw-loader, et url-loader. F

6. Comparaison avec des outils similaires

a. Browserify

Browserify est principalement un outil pour transformer les appels require() qui fonctionnent dans Node.js en appels qui fonctionnent dans le navigateur. Il s'agit d'un graphe de dépendances pour notre code source uniquement. On va lui spécifier un fichier d'entrée et se charger du reste. Sa simplicité d'utilisation est son plus grand atout. Il y'aura d'autres choses à gérer donc comme la minification, le regroupement, les tests en cours entre autres. Il devra donc être associé à des tasks runner pour inclure ses automatisations.

b. Grunt

Grunt est un task runner (coureur de tache) construit sur Node.js qui permet l'automatisation des tâches répétitives telles que : la minification, la compilation, les tests unitaires, l'optimisation d'images entre autres grâce à de nombreux plugins. Étant donné que Grunt met l'accent sur la configuration par rapport au code, la configuration de Grunt est un processus quelque peu simple. Cependant, pour cette raison, les fichiers de configuration Grunt ont tendance à devenir volumineux et gonflés. Bien que la configuration du Webpack soit un peu plus complexe, Webpack offre certains avantages par rapport à Grunt. Par exemple, avec la prise en charge de Webpack ES6, les loaders simples peuvent être canalisés ensemble pour créer des transformations plus complexes et Webpack permet de diviser votre base de code en morceaux chargés sur demande, réduisant ainsi les temps de chargement.

c. Gulp

Gulp est aussi un task runner. Son pont point est qu'il utilise des streams (flus de données en mémoire) et qu'il limite au maximum l'utilisation des fichiers et est donc beaucoup plus rapide dans l'exécution que Grunt. Aussi, il est basé sur le code par rapport à la configuration avec des plugins simples ce qui permet d'avoir un fichier de tâches plus propre et plus facile à lire avec une plus grande cohérence entre les tâches.

d. Rollup

Rollup est un bundler un peu différent des autres car il se focalise avant tout sur le code généré. Il supporte par défaut le Tree Shaking ce qui permet d'alléger encore plus le code n'incluant que le strict minimum. Au niveau de la configuration, le fonctionnement est assez proche de celui de browserify avec un point d'entrée et une série de plugins pour les transformations. L'absence du support du Code Splitting ou le Hot Module Reload rend Rollup moins intéressant pour bundler le code d'une application. En revanche, son support du Tree Shaking et la possibilité de choisir différents formats de sortie en fait un outil plus intéressant pour publier le code d'une librairie.

e. Parcel

Parcel est un projet qui est encore tout récent (Décembre 2017) mais qui propose une approche une approche assez originale. L'objectif est de proposer des temps de builds plus rapides grâce à la parallélisation mais surtout de retirer la configuration en se basant sur un fichier html comme point d'entrée. Il va automatiquement détecter les types de fichiers utilisés et télécharger les dépendances nécessaires à leurs traitements. Si votre manière de travailler correspond aux options proposées par défaut, Parcel peut être redoutablement efficace. En revanche, si vous rencontrer un cas particulier, l'absence de configuration peut devenir un réel problème.

Il existe plusieurs autres alternatives à Webpack, mais ce dernier reste le meilleur compromis et connait aujourd'hui une popularité ascendante. Dans le tableau ci-dessous, sous sont réunis quelques différences spécifiques entre les outils présentés ci-haut et d'autres :

Prise en charge	**Webpack**	**jrburke/requirejs**	**substack/node-browserify**	**jspm/jspm-cli**	**rollup**	**Brunch**
Des morceaux supplémentaires sont chargés à la demande	Oui	Oui	Non	System.import	Non	Non
DMLA *define*	Oui	Oui	deamdify	Oui	rollup-plugin-amd	Oui
DMLA *require*	Oui	Oui	Non	Oui	Non	Oui
AMD *require* charge à la demande	Oui	Avec configuration manuelle	Non	Oui	Non	Non
CommonJS *exports*	Oui	Seulement en enveloppant *define*	Oui	Oui	CommonJS plugin	Oui
CommonJS *require*	Oui	Seulement en enveloppant *define*	Oui	Oui	CommonJS plugin	Oui
CommonJS require.resolve	Oui	Non	Non	Non	Non	Non
Prise en charge du débogage	SourceUrl, SourceMaps	Non requis	SourceMaps	SourceUrl, SourceMaps	SourceUrl, SourceMaps	SourceMaps
Concat dans exiger *require("./fi"+"le")*	Oui	Non	Non	Non	Non	Non
Dépendances	19Mo/127 paquets	11Mo/118 paquets	1,2Mo/1 paquet	26Mo/131 paquets	Aucune info	Aucune info
ES2015 import/export	Oui	Non	Non	Oui	Oui	Oui
Expressions dans require(gratuit) *require(moduleName)*	Avec configuration manuelle	Non	Non	Non	Non	Non
Générer un seul paquet	Oui	Oui	Oui	Oui	Oui	Oui
Chargez chaque fichier séparément	Non	Oui	Non	Oui	Non	Non
Minimiser	Terser	uglify, compilateur de fermeture	Enlaidir	Oui	uglify-plugin	UglifyJS-brunch
Plugins	Oui	Oui	Oui	Oui	Oui	Oui
Bibliothèques intégrées Node.js require("path")	Oui	Non	Oui	Oui	node-resolve-plugin	Non

7. Pourquoi choisir Webpack ?

Pour comprendre pourquoi utiliser webpack, récapitulons comment nous utilisions JavaScript sur le Web avant que les bundlers n'interviennent.

Il existe deux manières d'exécuter JavaScript dans un navigateur. Tout d'abord, en incluant un script pour chaque fonctionnalité ; cette solution est difficile à faire évoluer car le chargement d'un trop grand nombre de scripts peut provoquer un goulot d'étranglement du réseau. La deuxième option consiste à utiliser un gros fichier .js contenant tout le code de notre projet, mais cela entraîne des problèmes de portée, de taille, de lisibilité et de maintenabilité. À ce niveau, l'on voit donc l'importance d'avoir un outil qui nous permettra non seulement d'écrire des modules, mais aussi de prendre en charge n'importe quel format de module (au moins jusqu'à ce que nous arrivions à ESM) et de gérer les ressources et les actifs en même temps.

C'est pourquoi webpack existe. C'est un outil qui nous permet de regrouper vos applications JavaScript (prenant en charge à la fois ESM et CommonJS), et il peut être étendu pour prendre en charge de nombreux actifs différents tels que des images, des polices et des feuilles de style,... Webpack se soucie des performances et des temps de chargement ; il s'améliore constamment ou ajoute de nouvelles fonctionnalités, telles que le chargement de blocs asynchrone et la prélecture, pour offrir la meilleure expérience possible à un projet et à des utilisateurs.

II. IMPLEMENTATION

1. Outils utilisés pour notre travail

2. Procédure de travail

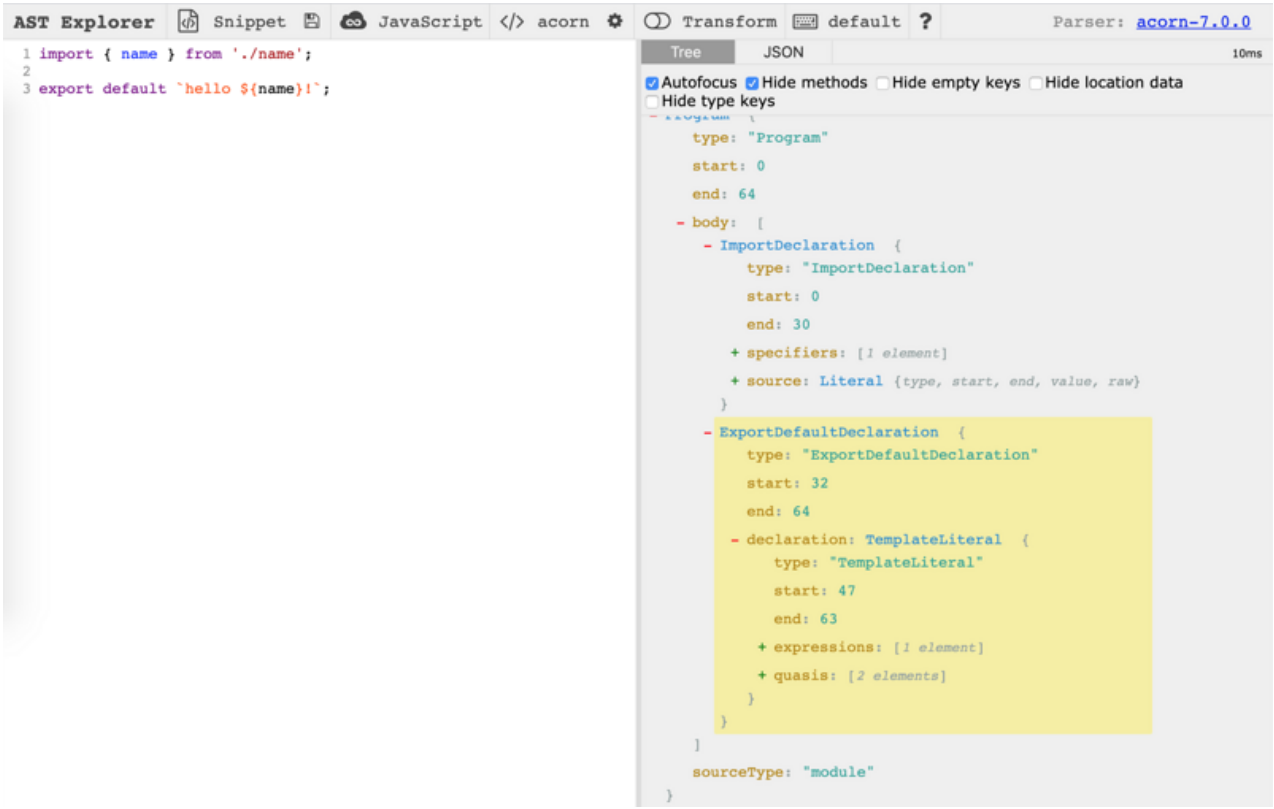
Selon notre idée d'implémentation, nous devons d'abord pouvoir analyser le contenu des fichiers JS et extraire leurs dépendances. On peut lire le contenu du fichier sous forme de chaînes et l' utilisation régulière pour obtenir les déclarations import, export, mais cette méthode est évidemment pas élégant et efficace. Une meilleure façon est d'utiliser l'analyseur JS pour analyser le contenu du fichier. L'analyseur JS peut analyser le code JS et le transformer en un modèle d'ordre élevé d'arbre de syntaxe abstraite (AST). L'arbre de syntaxe abstraite consiste à décomposer le code JS en une structure arborescente et peut en tirer plus de détails sur l'exécution du code.

3. Difficultés rencontrées

4. Captures d'écran et détails sur l'implémentation

- **Prise en main de l'analyseur JS**

Sur le site Web ast Explorer, nous pouvons afficher les résultats de l'analyse du code JS dans un arbre de syntaxe abstraite. Par exemple, Greeting.js Le contenu de l'analyseur d'Acron Les résultats sont les suivants



L'on peut voir que l'arbre de syntaxe abstraite est en fait un objet JSON, avec un pour chaque nœud les propriétés *type, import, export*, le résultat de l'analyse de l'instruction et ainsi de suite. Il est plus pratique d'extraire les informations clés après avoir converti le code en un arbre syntaxique abstrait.

Ensuite, nous avons généré, puis introduit un analyseur JS dans le projet. Nous avons choisi Babylon (Babylon est également l'analyseur JS utilisé en interne par Babel et existe actuellement dans l'entrepôt principal de Babel sous le nom @ Babel / analyseur). Ceci est fait avec la commande :

- Génération du parser

```
npm install --save-dev @babel/parser
```

- Ensuite, l'on a créé un fichier bundler.js pour introduire babel dans le projet :

```
const parser = require('@babel/parser');
```

❖ **Génération un arbre de syntaxe abstraite**

Avec l'analyseur JS, il est très simple de générer un arbre de syntaxe abstrait. Nous avons seulement besoin d'obtenir le contenu du fichier source JS et de le transmettre à l'analyseur pour l'analyse

CONCLUSION

Parvenus au terme de notre travail, il convient de rappeler son bienfondé et les étapes clés qui nous ont amenées jusqu'ici. Il était question pour nous d'effectuer un travail d'analyse puis une implémentation relative à Webpack. Pour mener à bien notre tâche, nous avons tout d'abord rappeler le contexte global du travail, puis les généralités sur Webpack et afin, nous avons mis sur pied une petite implémentation de ce module bundler à l'aide de React Js. Notre travail ainsi achevé, il est naturel de nous interroger sur comment l'intégrer sur des projets de plus grande portée.

RÉFÉRENCES

Liens utiles

- https://www.digital-campus.fr/glossaire-du-web/definition-developpement-web#:~:text=le%20d%C3%A9veloppement%20web%20fait%20r%C3%A9f%C3%A9rence%20au%20processus%20d%E2%80%99%C3%A9criture%20d%E2%80%99un%20site%20ou%20d%E2%80%99une%20page%20web%20dans%20un%20langage%20technique.%20Il%20s%E2%80%99agit%20d%E2%80%99une%20%C3%A9tape%20incontournable%20pour%20qu%E2%80%99un%20contenu%20soit%20mis%20en%20ligne%20et%20atteigne%20ses%20lecteurs 1
- https://fr.wikipedia.org/wiki/Webpack#:~:text=Webpack%20est%20un%20outil%20logiciel%20open%2Dsource%20de%20type%20%C2%AB%C2%A0module%20bundler%C2%A0%C2%BB%20(litt%C3%A9ralement%2C%20%C2%AB%C2%A0groupeur%20de%20modules%C2%A0%C2%BB)%2C%20con%C3%A7u%20pour%20faciliter%20le%20d%C3%A9veloppement%20et%20la%20gestion%20de%20sites%20et%20d%27applications%20web%20modernes. 2
- https://fr.wikipedia.org/wiki/Webpack#:~:text=Ce%20logiciel%20permet,d%27%C3%A9tendre%20ses%20capacit%C3%A9s. 3
- https://fr.wikipedia.org/wiki/Webpack#:~:text=Son%20principal%20avantage,sp%C3%A9cifiques%20d%27une%20application. 4
- https://bts-sioformation.com/javascript/webpack#:~:text=Avant%20d%E2%80%99intaller%20web pack,fonctionne%20avec%20Webpack
- https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/#:~:text=L%27ensemble%20de%20la%20communaut%C3%A9%20des%20d%C3%A9veloppeurs%20%C3%A9tait%20impliqu%C3%A9%20dans%20une%20qu%C3%AAte%20constante%20d%27am%C3%A9lioration%20de%20l%27exp%C3%A9rience%20globale%20des%20utilisateurs%20et%20des%20d%C3%A9veloppeurs%20autour%20de%20l%27utilisation%20et%20de%20la%20cr%C3%A9ation%20d%27applications%20javascript/web. A
- https://developpaper.com/a-brief-history-of-webpack-construction-and-development/#:~:text=Au%20cours%20de%20la%20derni%C3%A8re%20d%C3%A9cennie%2C%20avec%20la%20popularisation%20rapide%20des%20t%C3%A9l%C3%A9phones%20mobiles%2C%20des%20tablettes%20mobiles%20et%20des%20appareils%20portables%2C%20le%20d%C3%A9veloppement%20de%20l%27interface%20Web%20est%20%C3%A9galement%20pass%C3%A9%20du%20d%C3%A9veloppement%20traditionnel%20de%20pages%20Web%20sur%20PC%20au%20d%C3%A9veloppement%20multi%2Dterminal. B
- https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/#:~:text=Par%20cons%C3%A9quent%2C%20nous,de%20modules%20statiques. C
- https://www.sitepoint.com/webpack-beginner-guide/#:~:text=Dans%20un%20projet,vos%20fichiers%20HTML. D
- https://www.sitepoint.com/webpack-beginner-guide/#:~:text=M%C3%AAme%20un%20simple,par%20le%20navigateur. E

- https://www.sitepoint.com/webpack-beginner-guide/#:~:text=Les%20performances%20de,et%20url%2Dloader. F
- https://webpack.js.org/concepts/why-webpack/#:~:text=et%20de%20maintenabilit%C3%A9.-,IIFE%20%2D%20Expressions%20de%20fonction%20imm%C3%A9diatement%20invoqu%C3%A9es,d%27autres%20plugins%20et%20chargeurs%20%2C%20offre%20une%20excellente%20exp%C3%A9rience%20de%20d%C3%A9veloppement.,-Ne%20serait%2Dce G
- https://javascript.plainenglish.io/why-you-should-not-use-webpack-f07f4fd7c116#:~:text=1.%20Vitesses%20en,n%27est%20pas%20possible. H

Moteurs de recherche

ANNEXE

1. Guide d'installation et de création de projets

Installation de WebPack

Avant d'installer webpack il faut tout d'abord installer Node.js car webpack fonctionne grâce à Node. De plus l'on a aussi besoin de NPM le gestionnaire de module inclut dans Node. Pour ce faire, il suffit de lancer l'installation, si l'on ne touche à, rien Node est installé de manière globale et un chemin est ajouté au PATH pour que l'on puisse appeler Node de n'importe où.

Test de l'installation de Node

Il suffit d'ouvrir une fenêtre « Invite de Commandes » (la console) sur Windows et taper :

node --version

Si tout fonctionne l'on doit voir la version de Node installée.

Création d'un projet

Nous allons d'abord créer un répertoire (« TestWebPack » par exemple) puis s'y placer avec la fenêtre « Commande ». Dans la suite toutes les commandes sont tapées dans la fenêtre « Commande » sont les suivantes :

npm init

La commande npm init va nous aider un créer un fichier de configuration package.json. C'est le fichier de configuration de NPM. À chaque fois que nous téléchargerons un fichier avec NPM, nous pourrons l'ajouter au fichier package.json. Ainsi si nous voulons reconstruire votre répertoire sur un autre PC, nous prendrons juste le fichier package.json et NPM va importer tous les fichiers directement (npm install). A chaque question de l'utilitaire nous pouvons taper sur retour. À la fin des questions il faut taper sur « y ». Un fichier package.json a été créé avec nos informations dans le répertoire « testWebPack ». Désormais on peut télécharger webpack. Pour ce faire, l'on tape :

npm install --save-dev webpack

--save-dev signifie que l'on va sauvegarder le nom du module et la version dans le fichier package.json. Tous les modules téléchargés à partir de NPM se trouve dans le répertoire « Node modules ». Webpack étant en place, on peut s'attaquer à la programmation.

2. Dictionnaire technique