PRODUCTION & DÉVELOPPEMENT

WEBPACK


GOALS

PRODUCTION
DEVELOPMENT

OBJECTIFS OPÉRATIONNELS

Comprendre Webpack et son Écosystème
Créer son premier Module Js avec Webpack
S'approprier la configuration de Webpack

Surikat Studio
COMMUNICATION AGENCY

INTRODUCTION

IL ÉTAIT UNE FOIS 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.

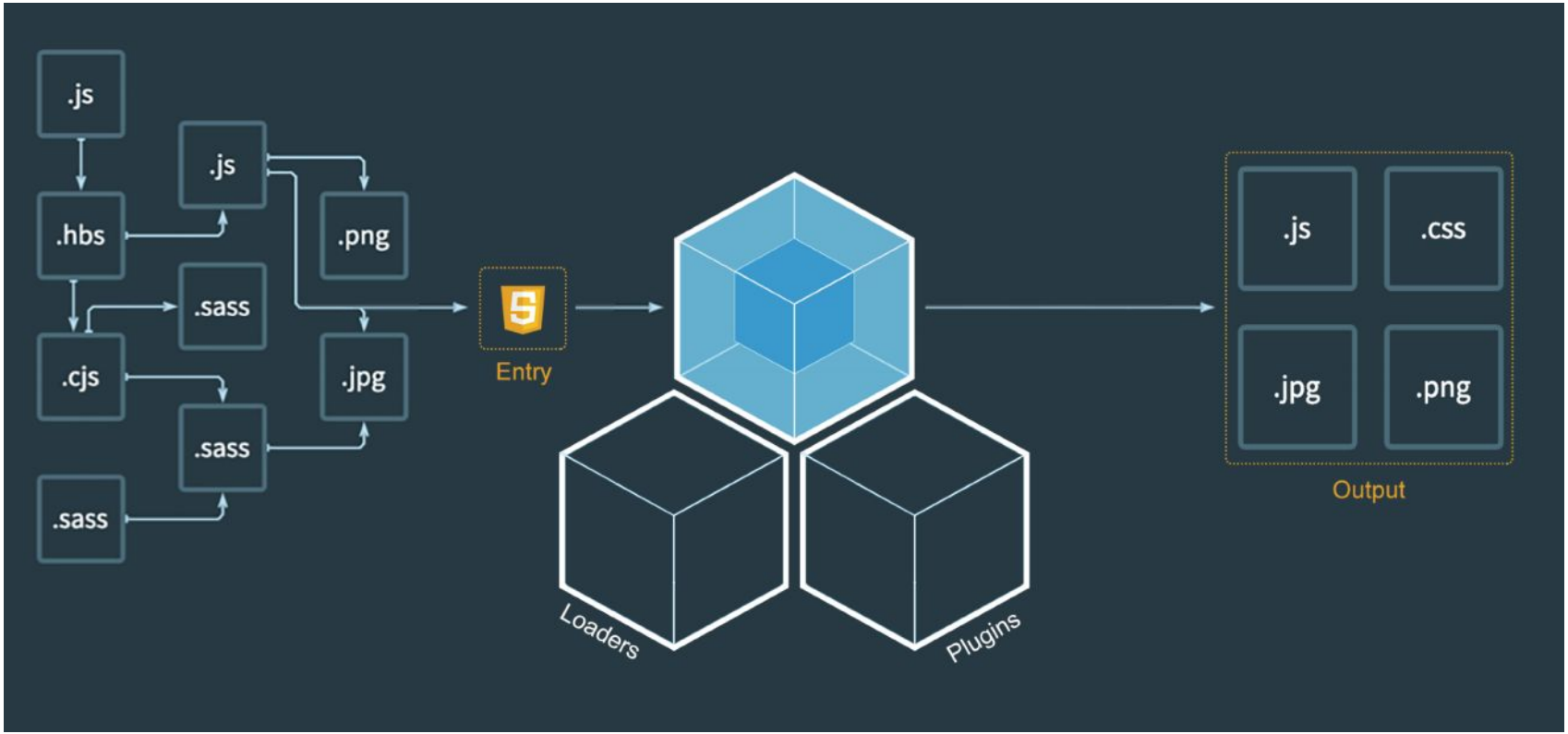
Il est initialement destiné au code JavaScript, mais sa forte modularité lui permet cependant de gérer beaucoup d'autres langages de programmation, notamment à l'aide de plugins tiers permettant d'étendre ses capacités.

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.

webpack

INTRODUCTION

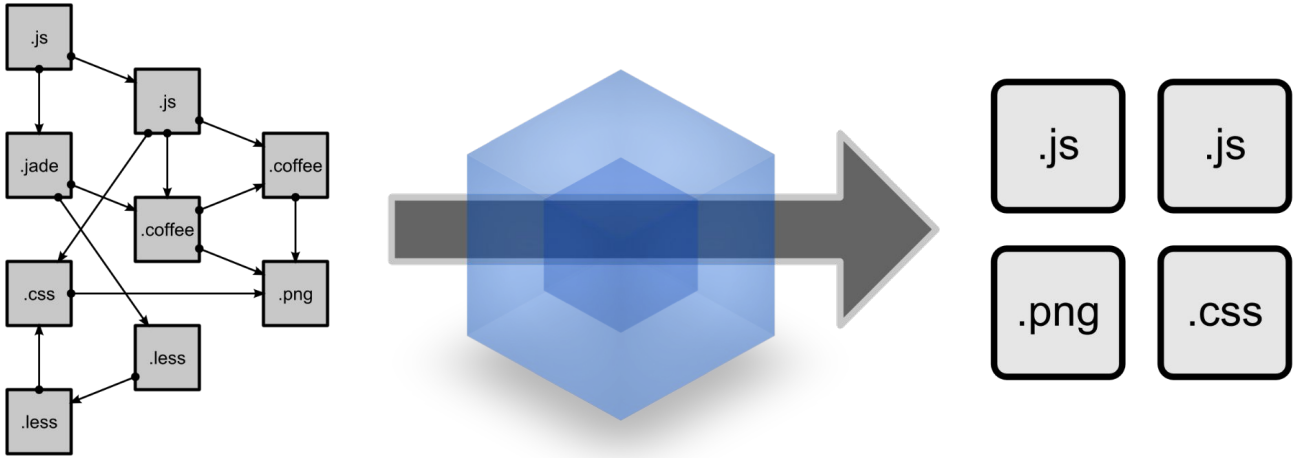
Comment ça marche ?



QU'EST-CE QUE WEBPACK ?

Webpack est un regroupeur de modules populaire largement utilisé dans le développement Web moderne.

Comment ça marche ?



QUELLES SONT LES DIFFÉRENCES ENTRE WEBPACK 4 ET WEBPACK 5 ?

Webpack 5 a introduit plusieurs fonctionnalités et améliorations notables par rapport à Webpack 4, notamment :

- Vitesse de construction améliorée : Webpack 5 offre des performances de construction améliorées avec des temps de compilation plus rapides et des mécanismes de mise en cache améliorés.
- Fédération de modules : Webpack 5 a introduit la fédération de modules, vous permettant de partager des modules sur plusieurs applications sans avoir besoin d'un bundle monolithique.
- Modules d'actifs : les modules d'actifs simplifient la gestion de divers actifs tels que les images, les polices et les fichiers multimédias, ce qui facilite leur gestion et leur optimisation.

QUELLES SONT LES PRINCIPALES FONCTIONNALITÉS DE WEBPACK ?

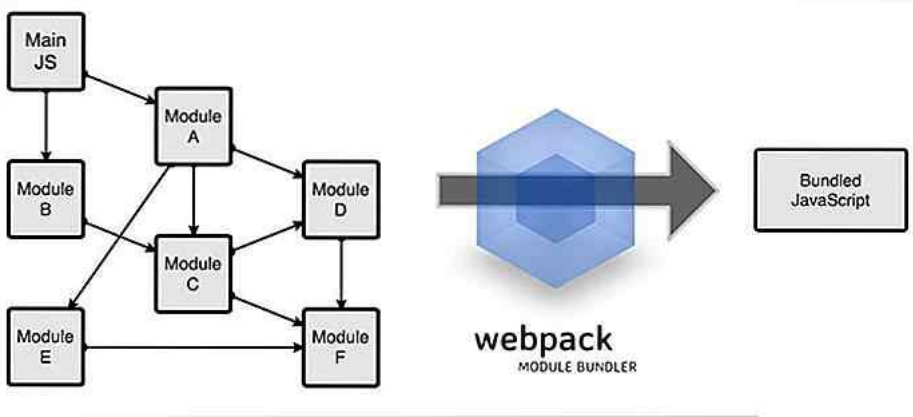
Webpack offre plusieurs fonctionnalités clés qui en font un choix populaire parmi les développeurs :

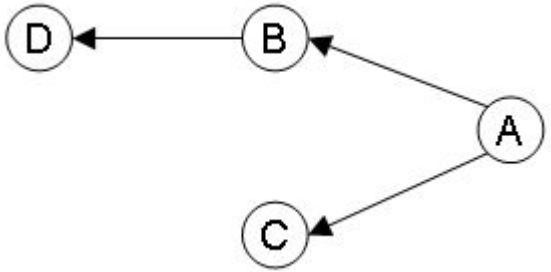
- Regroupement de modules : Webpack vous permet de regrouper toutes les dépendances, y compris JavaScript, CSS et d'autres actifs, dans un seul fichier ou quelques fichiers optimisés.
- Fractionnement du code : il prend en charge le fractionnement du code, ce qui vous permet de diviser votre code en morceaux plus petits et de les charger de manière asynchrone si nécessaire, réduisant ainsi les temps de chargement initiaux.
- Chargeurs : Webpack fournit des chargeurs pour prétraiter et transformer différents types de fichiers, tels que CSS, SCSS, TypeScript, etc., vous permettant de les intégrer de manière transparente dans votre projet.
- Plugins : les plugins améliorent les fonctionnalités de Webpack en fournissant des capacités supplémentaires d'optimisation, de génération de code et de gestion des actifs.

ARCHITECTURE ET FONCTIONNEMENT

COMMENT WEBPACK GÈRE-T-IL LES DÉPENDANCES ?

- Webpack utilise un graphe de dépendances pour gérer les dépendances.
- Il commence par un point d'entrée et suit récursivement les dépendances des modules pour construire un graphe complet.
- Chaque module est traité comme une entité distincte et peut avoir ses propres dépendances. Webpack analyse le graphique et regroupe toutes les dépendances dans un ou plusieurs fichiers de sortie.

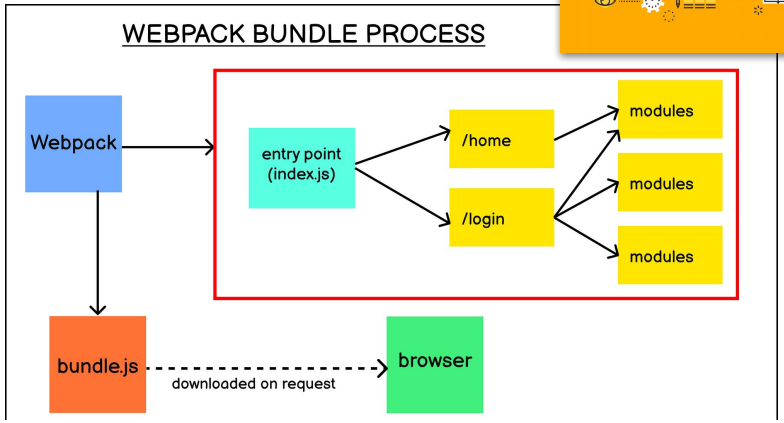


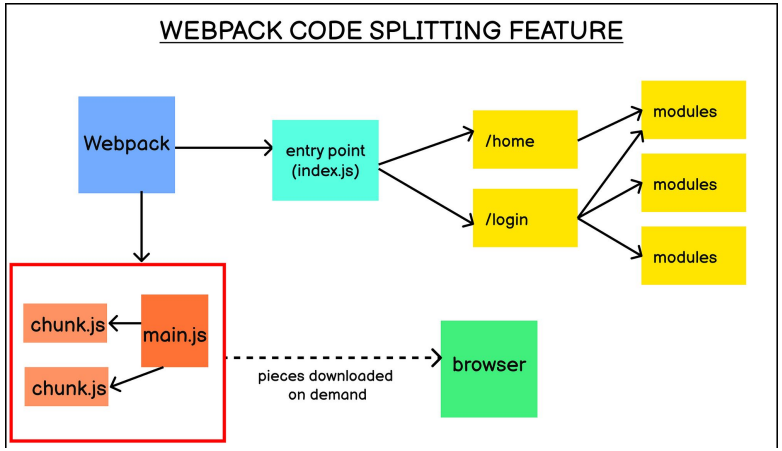


ARCHITECTURE ET FONCTIONNEMENT

LE CONCEPT DE FRACTIONNEMENT DE CODE DANS WEBPACK.

- Le fractionnement du code est une technique utilisée dans Webpack pour diviser le code en morceaux plus petits pouvant être chargés à la demande.
- Il permet de réduire le temps de chargement initial d'une application en chargeant uniquement le code essentiel requis pour la vue initiale et en chargeant du code supplémentaire si nécessaire.
- Webpack réalise le fractionnement du code via des importations dynamiques ou en configurant des points d'entrée pour des morceaux spécifiques.





ARCHITECTURE ET FONCTIONNEMENT

POUR OPTIMISER LA TAILLE DES BUNDLES WEBPACK, VOUS POUVEZ UTILISER PLUSIEURS STRATÉGIES :

- Minification : Webpack propose des plugins comme UglifyJSPlugin ou TerserPlugin pour minifier et compresser le code fourni, réduisant ainsi sa taille.
- Tremblement d'arbre : le tremblement d'arbre est un processus dans lequel Webpack élimine le code mort qui n'est pas utilisé dans l'application, ce qui entraîne des tailles de bundle plus petites.
- Fractionnement du code : en divisant le code en morceaux plus petits et en les chargeant à la demande, vous pouvez réduire la taille initiale du bundle et améliorer les performances.
- Utilisation des importations dynamiques : l'utilisation des importations dynamiques vous permet de charger des modules de manière asynchrone en cas de besoin, réduisant ainsi le temps de chargement initial.

COMMENT CONFIGURER WEBPACK ?

- Webpack peut être configuré à l'aide d'un fichier de configuration JavaScript (webpack.config.js).
- Dans ce fichier, vous définissez divers paramètres tels que les points d'entrée, les chemins de sortie, les chargeurs, les plugins et les options d'optimisation.
- Webpack fournit un système de configuration flexible qui vous permet de personnaliser le processus de regroupement en fonction des exigences spécifiques de votre projet.

Quelle est l'importance des fichiers de configuration tels que webpack.config.js, package.json, etc., lorsque vous travaillez avec Webpack ?

- Les fichiers de configuration tels que webpack.config.js et package.json sont importants lorsque vous travaillez avec Webpack car ils vous permettent de spécifier les différentes dépendances et paramètres que Webpack utilisera lors du regroupement de votre application.
- Sans ces fichiers, Webpack ne saurait pas comment construire correctement votre application.

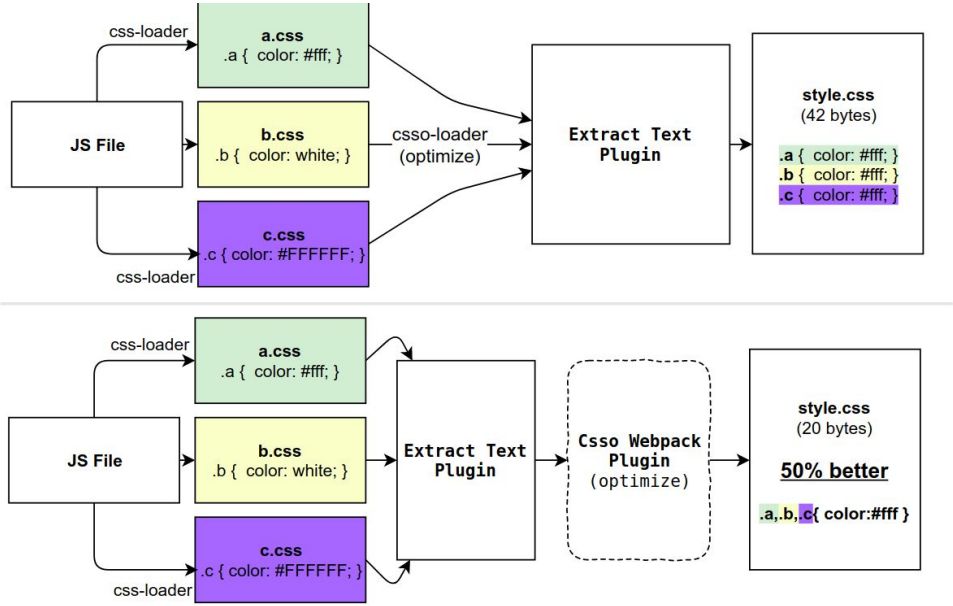
```
var path = require('path');
var webpack = require('webpack');

module.exports = {
  entry: {
    imax: './index.js'
  },
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].min.js',
    publicPath: '<prod path>', //prod publicPath
    library: 'IMAX',
  },
  plugins: [
    new webpack.optimize.UglifyJsPlugin()
  ],
  module: {
    rules: [{
      test: /\.js$/,
      exclude: [/node_modules/],
      use: {
        loader: 'babel-loader',
        options: {
          presets: ['es2015']
        }
      }
    }]
  }
};
```

LES LOADERS

QU'EST-CE QU'UN LOADER DANS WEBPACK ?

- Les Loaders (Chargeurs) dans Webpack sont des transformations appliquées aux fichiers sources au fur et à mesure qu'ils sont ajoutés au graphique de dépendances.
- Ils vous permettent de prétraiter les fichiers avant qu'ils ne soient regroupés. Par exemple, vous pouvez utiliser des chargeurs pour transpiler TypeScript en JavaScript, convertir SCSS en CSS ou optimiser et compresser des images.
- Les Loaders sont configurés dans le fichier de configuration Webpack et spécifiés à l'aide de règles.



LES PLUGINS

QU'EST-CE QU'UN PLUGIN DANS WEBPACK ?

- Les plugins sont en complément des loaders. Les plugins réalisent des actions plus complexes et font tout ce que les loaders ne peuvent pas faire comme l'optimisation de bundles, l'injection de variables de configuration (ou d'environnement) ou encore la gestion des assets.
- Les plugins sont l' épine dorsale du webpack. Webpack lui-même est construit sur le même système de plugins que vous utilisez dans la configuration de votre webpack !
- Un plugin webpack est un objet JavaScript qui possède une apply méthode. Cette applyméthode est appelée par le compilateur webpack, donnant accès à l' intégralité du cycle de vie de la compilation.

webpack.config.js

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

module.exports = {
  entry: './path/to/my/entry/file.js',
  output: {
    filename: 'my-first-webpack.bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        use: 'babel-loader',
      },
    ],
  },
  plugins: [
    new webpack.ProgressPlugin(),
    new HtmlWebpackPlugin({ template: './src/index.html' }),
  ],
};
```

LES PLUGINS

Nom	Description
BannerPlugin	Ajouter une bannière en haut de chaque morceau généré
ChunksWebpackPlugin	Créez des fichiers HTML avec des relations de points d'entrée et de morceaux pour servir vos bundles
CommonsChunkPlugin	Extraire les modules communs partagés entre les morceaux
CompressionWebpackPlugin	Préparez des versions compressées des ressources pour les servir avec Content-Encoding
ContextReplacementPlugin	Remplacer le contexte déduit d'une requireexpression
CopyWebpackPlugin	Copie des fichiers individuels ou des répertoires entiers dans le répertoire de construction
DefinePlugin	Autoriser les constantes globales configurées au moment de la compilation
DllPlugin	Divisez les bundles afin d'améliorer considérablement le temps de construction
EnvironmentPlugin	Raccourci pour utiliser les touches DefinePluginonprocess.env

LES PLUGINS

Nom	Description
HotModuleReplacementPlugin	Activer le remplacement du module à chaud (HMR)
HtmlWebpackPlugin	Créez facilement des fichiers HTML pour servir vos offres groupées
IgnorePlugin	Exclure certains modules des offres groupées
LimitChunkCountPlugin	Définir des limites min/max pour le chunking afin de mieux contrôler le chunking
MinChunkSizePlugin	Maintenir la taille du morceau au-dessus de la limite spécifiée
MiniCssExtractPlugin	crée un fichier CSS par fichier JS qui nécessite CSS
NoEmitOnErrorsPlugin	Ignorer la phase d'émission en cas d'erreurs de compilation
NormalModuleReplacementPlugin	Remplacer les ressources qui correspondent à une expression rationnelle
NpmInstallWebpackPlugin	Installer automatiquement les dépendances manquantes pendant le développement

LES PLUGINS

Nom	Description
ProgressPlugin	Progression de la compilation du rapport
ProvidePlugin	Utiliser des modules sans avoir à utiliser import/require
SourceMapDevToolPlugin	Permet un contrôle plus fin des cartes sources
EvalSourceMapDevToolPlugin	Permet un contrôle plus fin des cartes sources d'évaluation
SvgChunkWebpackPlugin	Générez des sprites SVG optimisés par SVGO en fonction des dépendances de votre point d'entrée
TerserPlugin	Utilise Terser pour réduire le JS dans votre projet

DIFFÉRENCES ENTRE PLUGINS ET LOADERS

QU'EST-CE QU'UN LOADER DANS WEBPACK ?

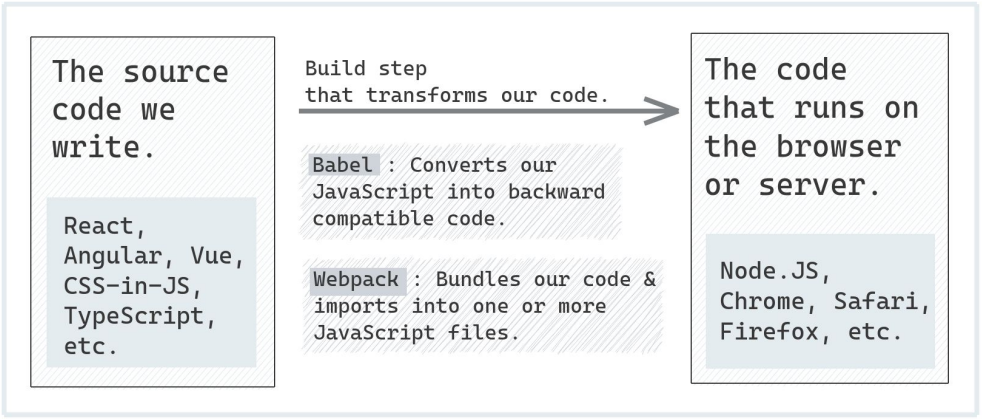
- Les Loaders (Chargeurs) dans Webpack sont des transformations appliquées aux fichiers sources au fur et à mesure qu'ils sont ajoutés au graphique de dépendances.
- Ils vous permettent de prétraiter les fichiers avant qu'ils ne soient regroupés. Par exemple, vous pouvez utiliser des chargeurs pour transpiler TypeScript en JavaScript, convertir SCSS en CSS ou optimiser et compresser des images.
- Les Loaders sont configurés dans le fichier de configuration Webpack et spécifiés à l'aide de règles.

Webpack Loaders & Plugins : What's the difference?

Loaders	Plugins
- Tell webpack how to load a certain import / source file. - Are generally file-type specific. - Eg - 'style-loader' loads CSS files & injects CSS.	- Tell webpack how to output bundles. - Work on chunk files at the end of bundling. - Eg : 'terser-webpack-plugin' minifies bundle files.

BABEL

- **B**abel est un transcompilateur JavaScript gratuit et open source qui est principalement utilisé pour convertir le code ECMAScript 2015+ (ES6+) en code JavaScript rétrocompatible pouvant être exécuté par des moteurs JavaScript plus anciens . Il permet aux développeurs Web de profiter des dernières fonctionnalités du langage. [4]
- Les développeurs peuvent utiliser les nouvelles fonctionnalités du langage JavaScript en utilisant Babel pour convertir leur code source en versions de JavaScript qu'un navigateur Web peut traiter.
- Babel peut également être utilisé pour compiler TypeScript en JavaScript.



COMMENT GÉRER LE CSS DANS WEBPACK ?

Webpack propose plusieurs façons de gérer CSS, notamment :

- Chargeurs CSS : vous pouvez utiliser des chargeurs CSS comme `style-loader`et `css-loader`pour importer des fichiers CSS et appliquer des styles à votre application.
- Préprocesseurs CSS : Webpack prend en charge les chargeurs pour les préprocesseurs CSS tels que Sass, Less et PostCSS, vous permettant d'écrire du CSS avec des fonctionnalités avancées et de les compiler en CSS standard.
- Extraction de CSS : Webpack vous permet d'extraire le CSS dans des fichiers séparés à l'aide de plugins tels que `mini-css-extract-plugin`, ce qui est utile pour les versions de production.

SYNTHÈSE

Étant donné un grand nombre de fichiers,
Webpack génère un seul fichier (ou quelques fichiers) qui exécutent votre application.

Il peut effectuer de nombreuses opérations:

- vous aide à regrouper vos ressources.
- surveille les changements et réexécute les tâches.
- peut exécuter la transpilation Babel vers ES5, vous permettant d'utiliser les dernières fonctionnalités de JavaScript sans vous soucier de la prise en charge du navigateur.
- peut transpiler CoffeeScript en JavaScript
- peut convertir des images en ligne en URI de données.
- vous permet d'utiliser require () pour les fichiers CSS.
- peut exécuter un serveur Web de développement.
- peut gérer le remplacement du module à chaud.
- peut diviser les fichiers de sortie en plusieurs fichiers, pour éviter d'avoir un énorme fichier js à charger dans la première page.
- peut effectuer secousse d'arbre.

Webpack ne se limite pas à être utilisé sur le frontend, il est également utile dans le développement backend de Node.js.

PRODUCTION
DEVELOPMENT

WEBPACK c’est quoi ?

Introduction

Installation Globale

Installation Locale

Configuration

THÈMES DU PROGRAMME

INSTALLATION GLOBALE

Avec yarn :

```
yarn global add webpack webpack-cli
```

Avec npm:

```
npm i -g webpack webpack-cli
```

Puis lancez la commande suivante dans votre terminal

```
webpack-cli
```

INSTALLATION LOCALE

Avec yarn :

```
yarn add webpack webpack-cli -D
```

Avec npm:

```
npm i webpack webpack-cli --save-dev
```

Puis ajouter les lignes suivantes à votre package.json

```
//…
{
 "scripts": {
   "build": "webpack"
 }
}
```

CONFIGURATION

Puis lancer la commande

Avec yarn :

```
yarn build
```

Avec npm:

```
npm run build
```

CONFIGURATION

Maintenant que Webpack est installé, nous allons devoir le configurer !

Pour cela, ouvrez `webpack.config.js` et ajoutez les lignes suivantes :

```
const webpack = require("webpack");
const path = require("path");
```

Nous créons ici une constante appelée `webpack`, qui requiert le module webpack. C'est la base pour que tout fonctionne. Puis nous créons la variable `path` qui va nous permettre d'indiquer les bons chemins vers nos fichiers.

Ensuite, il va falloir indiquer un point d'entrée (notre fichier `index.js`), c'est-à-dire le fichier qui sera lu et un point de sortie, à savoir le fichier qui sera compilé (`bundle.js`). Ensuite, nous allons exporter cette configuration :

```
let config = {
  entry: "./src/index.js",
  output: {
    path: path.resolve(  dirname, "./public"),
    filename: "./bundle.js"
  }
}

module.exports = config;
```

CONFIGURATION

Maintenant, n'oubliez pas de lier votre fichier `bundle.js` à votre fichier `index.html`

en tapant la ligne de code qui suit, juste avant la fermeture de votre balise `<body>` :

```
<script src="bundle.js"></script>
```

Rendez-vous désormais dans votre fichier `index.js` et tapez le code JS de votre choix. J'ai opté pour un simple :

```
document.write("Je débute avec Webpack !");
```

Maintenant, tapez simplement `webpack` dans votre terminal et laissez la magie opérer :)
Profitez-en pour ouvrir votre `index.html` avec le navigateur de votre choix pour voir le contenu de votre page !

Un peu d'automatisation

Tout ceci est bien beau, mais vous n'allez pas taper `webpack` dans votre terminal à chaque fois que vous voulez voir vos derniers changements, ça serait trop lourd ! Un flag existe pour cela : `--watch` ou `-w`. Allez-y, tapez `webpack --watch` et regardez ce qu'indique Webpack dans votre terminal :

Cela signifie que Webpack surveille les modifications que vous allez apporter à vos fichiers. Vous pourrez donc simplement rafraîchir votre page HTML à chaque fois que vous voudrez voir le rendu. D'ici quelques paragraphes, nous verrons comment mettre en place le *Hot Module Reload* ou HMR pour les intimes : vos pages seront automatiquement rafraîchies à chaque fois que vous sauvegarderez votre travail depuis votre éditeur de code.

Reprenez donc votre fichier index.js et éditez le texte que vous aviez écrit. Sauvegardez, rechargez la page : magie, ça fonctionne ! Mais nous pouvons encore améliorer ceci. Plutôt que de taper `webpack --watch`, nous allons rendre ça un peu plus sexy : rendez-vous dans votre fichier `package.json` et modifiez l'objet script comme suit

```
"scripts": {
    "watch": "webpack --watch"
  }
```

Maintenant, lorsque vous voudrez lancer la commande `watch` depuis votre terminal, tapez `npm run watch`.

PRODUCTION
DEVELOPMENT

Dans les profondeur de WEBPACK

Le point d'entrée

Le résultat

Chargeurs

Plugin

Le mode Webpack

Exécution de Webpack

Regarder les changements

Gestion des images

Traitez votre code SASS et transformez-le en CSS

Générer des cartes sources

THÈMES DU PROGRAMME