NPM - WEBPACK

avril 2023

Cindy PONCIN

1

Composer

NPM - WEBPACK

1

Présentation de composer

“ Composer est un logiciel gestionnaire de dépendances libre écrit en PHP. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin. “

Avril 2011 : début du développement
Mars 2012 : première version sortie le 1er mars 2012.

Installation de composer

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash_file('sha384', 'composer-setup.php') ===
'55ce33d7678c5a611085589f1f3ddf8b3c52d662cd01d4ba75c0ee0459970c2200a51f492d557530c
71c15d8dba01eae') { echo 'Installer verified'; } else { echo 'Installer corrupt';
unlink('composer-setup.php'); } echo PHP_EOL;"

php composer-setup.php

php -r "unlink('composer-setup.php');"
```

Pour windows

Nécessité de télécharger composer

https://getcomposer.org/Composer-Setup.exe

Installation de composer

```
mv composer.phar /bin/composer
```

Si composer est installer localement (dans votre répertoire) il vaut mieux l'installer globalement

Fonctionnement composer

Composer est basé sur des commandes pour installer dans un répertoire **vendor** les packages indiqués dans le répertoire **composer.json**

Les packages installés s'appuient sur des dépôts : un des plus connus est Packagist.org.

composer.json

Le fichier composer.json contient l'**ensemble des dépendances d'un projet et leur version**.

```
{
    "name": "packaName",
    "description": "package short description ",
     "require": {
                "php": ">=8.0.2",
                "doctrine/annotations": "^1.13",
                "symfony/form": "6.0.*"
       }
}
```

Les dépendances sont aussi appelées "bibliothèque" ou package.

composer.json

Il est possible de spécifier des versions de packages grâce à une notation spécifique

Exactement

"vendor/packageName": "1.3.2"

Version >, <, <=, >=

"vendor/packageName": ">=1.3.2",
"vendor/packageName": "<1.3.2",

composer.json

Le fichier composer.json peut également **définir un projet**

```
{
    {
     "type": "project",
     "license": "proprietary",
     "minimum-stability": "stable",
     "prefer-stable": true,
     "require": {
         "php": ">=8.0.2"
    }
}
```

composer.json

Grâce à composer on peut définir :

- définir les contraintes d'environnement comme la version de php
- définir les versions des packages à installer

composer.json

L'ensemble des révisions d'une version spécifiée

"vendor/package": "1.3.*", // >=1.3.0 <1.4.0

L'ensemble des nouvelles révisions de la version spécifiée

"vendor/package": "~1.3.2", // >=1.3.2 <1.4.0

"vendor/package": "~1.3", // >=1.3.0 <2.0.0

Utiliser composer

Ajout d'un package au fichier composer.json

```
composer require packageName
```

Prérequis
Vous positionner au même niveau que le fichier composer.json avant de lancer la commande

Utiliser composer

Installation de toutes les packages du fichier composer.json

```
composer install
```

Prérequis
Vous positionner au même niveau que le fichier composer.json avant de lancer la commande

Arborescence des fichiers

Fichier composer.json
Liste des dépendances nécessaires à notre projet

Dossier vendor
Contient les sources de ces dépendances

Fichier composer.lock
Contient l'état exact (la version) de nos dépendances au moment où l'on travaille sur le projet

Utiliser composer

Suppression d'un package du fichier composer.json

```
composer remove packageName
```

Prérequis
Vous positionner au même niveau que le fichier composer.json avant de lancer la commande

Utiliser composer

Mise à jour de l'ensemble des packages du fichier composer.json

```
composer update
```

Prérequis
Vous positionner au même niveau que le fichier composer.json avant de lancer la commande

Composer et versionning

Fichier composer.json
Doit être versionné, sans ce fichier, les développeurs travaillant sur votre projet n'auront aucune idée des packages nécessaires au projet.

Dossier vendor
Ne doit pas être versionné, car il est généré automatiquement par composer

Fichier composer.lock
Ne doit pas être versionné car il y aura un conflit permanent avec les autres utilisateurs du dépôt

Les atouts de composer

- Allègement des projets : les dépendances ne sont pas versionnées
- Mise à jours facilitées

Créer un projet projet et installer quelques dépendances

TD

Autoloading de composer

Composer contient un système d'"autoloading"

```
{
    "autoload": {
        "psr-4": {"projectName\\": "src/"}
    }
}
```

Permet aux packages de spécifier la manière dont Composer peut faire correspondre un nom de classe avec un nom de fichier.

Autoloading de composer

Il suffit d'inclure vendor/autoload.php (le require n'est plus nécessaire)

```
require __DIR__ . '/vendor/autoload.php';
```

Créer un projet Wordpress en utilisant composer

Création d'un nouveau projet

Créer un répertoire qui contiendra votre nouveau projet et vous placer à la racine

```
mkdir monProjet

cd monProjet
```

Démarrage du projet composer

Créer un fichier composer.json à la racine de votre projet

```
{
}
```

Pré requis
Composer devra avoir été préalablement installé

Définition d’un gestionnaire de package par défaut

Indiquer le gestionnaire de package par défaut

```
{
    "repositories":[
        {
            "type":"composer",
            "url":"https://wpackagist.org"
        }
    ]
}
```

A noter
Nous définition ici le gestionnaire de package par défaut, wpackagist.org

Définition du core Wordpress

Indiquer le package wordpress à installer et sa version

```
{
      "require": {
            "johnpbloch/wordpress": "6"
       },
       "repositories":[

          {
             "type":"composer",
             "url":"https://wpackagist.org"
          }
     ]
}
```

A noter
Nous choisissons d'installer Wordpress 6

Définition du thème

Indiquer le thème à installer

```
{
      "require": {
            "johnpbloch/wordpress": "6",
            "wpackagist-theme/twentytwentytwo":"1.4"
       },
       "repositories":[

          {
             "type":"composer",
             "url":"https://wpackagist.org"
          }
     ]
```

A noter
Nous choisissons d'installer le thème twentytwo. Il est possible d'utiliser votre thème personnalisé

Définition des extensions

Indiquer les extensions wordpress à installer

```
{
      "require": {
            "johnpbloch/wordpress": "6",
            "wpackagist-plugin/contact-form-7":"5",
            "wpackagist-theme/twentytwentytwo":"1.4"

      },
      "repositories":[

         {
            "type":"composer",
            "url":"https://wpackagist.org"
         }
     ]
}
```

A noter
Nous choisissons d'installer le plugin contact-form-7 version 5

Définition des répertoires d'installation

Indiquer les chemins d'installation

```
{
      "extra": {
            "installer-paths": {
            "web/app/mu-plugins/{$name}/": ["type:wordpress-muplugin"],
            "web/app/plugins/{$name}/": ["type:wordpress-plugin"],
            "web/app/themes/{$name}/": ["type:wordpress-theme"]
            },
            "wordpress-install-dir": "web/wp"
            },

}
```

L'arborescence du projet sera différente d'un projet Wordpress classique

Configuration du projet Wordpress

A la racine du projet créer les fichiers

- index.php
- .htaccess
- wp-config.php

Configuration du projet Wordpress

Index.php

```
<?php
define('WP_USE_THEMES', true);
require( dirname( __FILE__ ) . '/wp/wp-blog-header.php' );
```

Configuration du projet Wordpress

.htaccess

```
# BEGIN WordPress

<IfModule mod_rewrite.c>

RewriteEngine On
RewriteBase /
RewriteRule ^index\.php$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.php [L]

</IfModule>

# END WordPress
```

Configuration du projet Wordpress

wp-config..php

```
<?php

$webroot_dir = 'http://localhost:8000/202304-composer/web';

define('WP_SITEURL', "{$webroot_dir}/wp");
define('WP_HOME',"{$webroot_dir}");

$httpHost =  isset($_SERVER['HTTP_HOST']) ? $_SERVER['HTTP_HOST']
: $webroot_dir;

define( 'WP_CONTENT_DIR', dirname( __FILE__ ) . '/app' );
define( 'WP_CONTENT_URL', 'http://' . $httpHost . '/app' );

define( 'DB_HOST', 'db');
define( 'DB_NAME','MYSQL_DATABASE');
define( 'DB_USER','MYSQL_USER');
define( 'DB_PASSWORD','MYSQL_PASSWORD');
```

Configuration du projet Wordpress

wp-config.php (suite)

```
//suite …
table_prefix = 'wp_';
/** Absolute path to the WordPress directory. */
if ( !defined('ABSPATH') ) {
    define('ABSPATH', dirname(__FILE__) . '/wp');
}

/** Sets up WordPress vars and included files. */
require_once(ABSPATH . 'wp-settings.php');
```

2

Initialisation du projet avec Node et NPM

NPM - WEBPACK

Présentation de NPM

“npm est le gestionnaire de paquets par défaut pour l'environnement d'exécution JavaScript Node.js ”

Janvier 2010 : sortie de la première version de NPM

Installation de NPM

Pour installer NPM, il faut installer Node.js qui l'inclut

https://nodejs.org/fr/download/

Pour vérifier la bonne installation de Node et NPM

```
node -v
npm -v
```

pour mettre à jour npm install npm -g

Fonctionnement de NPM

Composer est basé sur des commandes pour installer dans un répertoire **node_modules** les packages indiqués dans le répertoire **package.json**

Les packages installés s'appuient sur des dépôts : un des plus connus est www.npmjs.com

Utiliser NPM

Créer un fichier package.json

```
npm init
```

Prérequis
Vous positionner dans votre répertoire projet

package.json

Le fichier package.json contient l'**ensemble des dépendances d'un projet et leur version**.

```
{
  "name": "td",
  "version": "1.0.0",
  "description": "Package de test",
  "main": "test.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "CP",
  "license": "ISC"
}
```

Les dépendances sont aussi appelées "bibliothèque" ou package.

Utiliser NPM

Ajout d'un package spécifique

```
npm install packageName --save
```

--save permet d'ajouter le package au fichier au fichier package.json

Utiliser NPM

Ajout de l'ensemble des packages présent dans le fichier package.json

```
npm install
```

Utiliser NPM

Savoir si un package est à jour

```
npm outdated

OU

npm outdated -g
```

-g : paquets globaux

Utiliser NPM

Mettre à jour un package

```
npm update

OU

npm update -g
```

-g : paquets globaux

Utiliser NPM

Supprimer un package

```
npm uninstall packageName
```

Installation et configuration de Webpack

NPM - WEBPACK

Webpack

Webpack est un bundler qui permet en autre de compiler plusieurs fichiers en un seul, de générer un serveur local

Webpack permet ainsi d'améliorer les performances des applications.

TD

Créer un répertoire projet

td-webpack

Créer l’arborescence suivante dans le répertoire de votre projet

webpackTd

--public
----index.html
----td.js

--assets
----fonts
----icons
----images
----stylesheets
--src
----index.js
--webpack.config.js

Préparer votre votre projet à l'utilisation de NPM

Se placer à la racine du projet

```
npm init -y
```

A noter
Un fichier package.json a été créé à la racine de votre projet
Faire les modifications nécessaires

TD

Installer webpack

Se placer à la racine du projet

Installation environnement de développement

```
npm install --save-dev webpack@latest webpack-dev-server@latest
```

Installation globale

```
npm install -g webpack@latest
```

L’arborescence doit être la suivante

Configurer webpack - webpack.config.js

```
const webpack = require("webpack");
const path = require("path");
const isProduction = 'development';

let config = {
    entry: "./src/index.js",
    output: {
        filename: "td.js",
        path: path.resolve(__dirname, "public"),
        clean: false,
    }
  }
  module.exports = config;
```

Lier votre fichier td.js à votre fichier index.html

```
<html>
<head></head>
<body>
<script src="td.js"></script>
</body>
</html>
```

Modifier votre fichier index.js

```
document.write("Hello world!");
```

Compiler

4

Automatisation

NPM - WEBPACK

Automatisation

Pour automatiser l'utilisation de webpack, dans le fichier package.json ajouter :

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

L'automatisation permet d'éviter l'exécution manuelle de la commande webpack

Automatisation

Une fois la modification faite : pour lancer l'exécution automatique, il suffira à présent de lancer la commande

```
npm run watch
```

Compiler de l'ES6 en ES5

NPM - WEBPACK

Compiler de l'ES6 en ES5

Constat

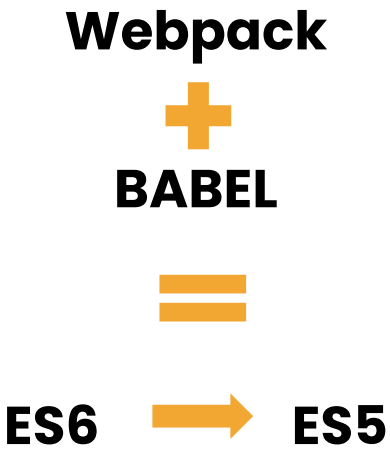
L'ES6 est la version la plus récente du langage Javascript, cependant elle n'est pas encore reconnue par tous les navigateurs

Exemple de spécificité lié à l'ES6 : let, les fonctions fléchées, etc.

ES6 évolue régulièrement

Compiler de l'ES6 en ES5

Solution



Compiler de l'ES6 en ES5

Installer des packages babel-loader et babel-core

```
npm install --save-dev babel-loader
npm install --save-dev babel-core


ou


npm install --save-dev babel-loader babel-core
```

Compiler de l'ES6 en ES5

Modifier la configuration webpack.config.json

```
module: {
    rules: [{
      test: /\.js$/,
      exclude: /node_modules/,
      loader: "babel-loader"
    }]
  }
```

A mettre après le Output

Compiler de l'ES6 en ES5

Indiquer à babel de compiler en ES5

```
npm install --save-dev babel-preset-env
```

Compiler de l'ES6 en ES5

Créer un fichier à la racine du projet .babelrc

```
{
  "presets": [
    ["env", {
      "targets": {
        "browsers": ["last 2 versions", "safari >= 7"]
      }
    }]
  ]
}
```

Compiler du SCSS en CSS

NPM - WEBPACK

Installation d’un compilateur

En ligne de commande

```
npm install -g sass
```

Pour vérifier la bonne installation

```
sass --version
```

Utilisation d’un compilateur

Compilation en ligne de commande

```
sass style.scss style.css
```

Le fichier style.css est la version compilée de votre fichier scss.
Un fichier style.css.map est généré

Utilisation d’une extension

Extension dans votre IDE

Exemple sur Visual Studio Code

Live Sass compiler

Avantage

Compilation automatique à l’enregistrement du fichier scss.
Possibilité de sélectionner le répertoire, les fichiers à compiler, et la destination du fichier.

Avec webpack

Installer les packages

```
npm install sass-loader sass webpack --save-dev
```

```
// Creates `style` nodes from JS strings
    "style-loader",
    // Translates CSS into CommonJS
    "css-loader",
    // Compiles Sass to CSS
    "sass-loader",
```

Avec webpack

Importer le fichier scss dans le fichier td.js

```
import "../assets/stylesheets/style.scss";
```

Avec webpack

Modifier le fichier webpack.config.js pour ajouter les modules

```
module: {
        rules: [{
            test: /\.js$/,
            exclude: /node_modules/,
            loader: "babel-loader"
        },
        {
            test: /\.s[ac]ss$/i,
            use: [
              "style-loader", "css-loader", "sass-loader",
            ],
        }
        ]
    }
```

Créer un fichier style.scss
dans le répertoire assets/stylesheets

7

Minifier les fichiers Javascripts

NPM - WEBPACK

Minifier les fichiers Javascript

Minification
des fichiers Javascript.

Taille de fichier
plus compacte.

Accélération du chargement
des pages web.

Fonctionnement de la minification

La minification est un système qui permet d'**enlever tous les caractères inutiles** d'un fichier :

- les espaces
- les sauts de lignes
- les commentaires
- les séparateurs

Les fichiers javascripts et css doivent être minifiés

Fonctionnement de la minification

Exemple de JS avant minification

```
function helloWorld( text ) {
  document.write( text );
}

helloWorld( 'Bonjour tout le monde' );
```

Fonctionnement de la minification

Exemple *Js après minification*

```
function hello(e){document.write(e)}hello("Welcome to the
article")
```

Minifier ses fichiers

Différents outils de minifications :

- en ligne
- extensions de votre IDE
- en ligne de commande

Minifier ses fichiers

En ligne

https://jscompress.com/

https://www.toptal.com/developers/javascript-minifier

https://codebeautify.org/minify-js

Il existe de nombreux autres outils de minification en ligne.

Minifier ses fichiers

Extension dans votre IDE

Exemple sur Visual Studio Code

JS & CSS Minifier (Minify)

Il existe de nombreuses extensions selon votre IDE

Minifier ses fichiers

En ligne de commande (UglifyJS)

```
uglifyjs --compress -- test.js > test.min.js
```

Pour installer uglify

```
npm install uglify-js -g
```

Il existe d'autres outils en ligne de commande

Avec webpack

Installer le package

```
npm install uglifyjs-webpack-plugin --save-dev
```

Avec webpack

Modifier le fichier webpack.config.json et ajouter

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

module.exports = {
  optimization: {
    minimizer: [new UglifyJsPlugin()],
  },
};
```