Webpack – VITE !

Mettez en place une architecture de projet conséquente

Animé par Mazen Gharbi

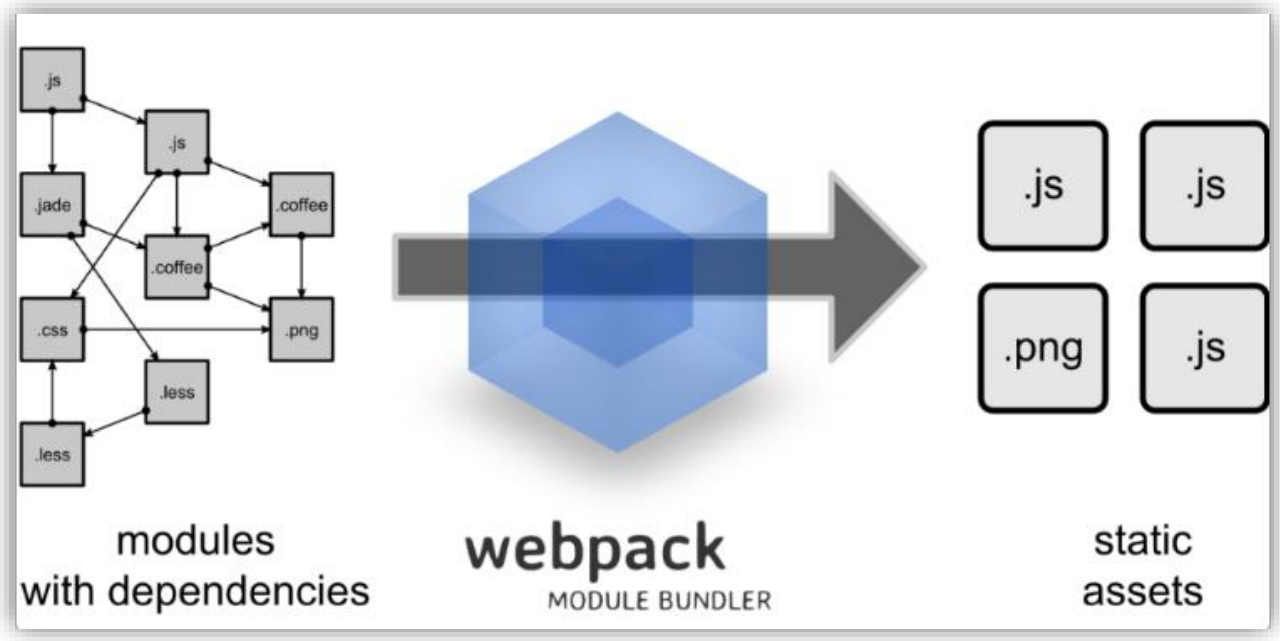
Pourquoi un nouvel outil ?

- VueJS n'est pas une simple librairie mais un framework complexe
- Contrairement à une librairie, il ne suffit pas d'include un fichier JS et de démarrer
- Nous allons donc utiliser « webpack » pour construire notre app
- On aimerait développer avec toutes les fonctionnalités EcmaScript 6+ !
- Webpack 4 est la dernière grosse mise à jour

Webpack, c’est quoi?

- Webpack est un « module bundler » ;

- Il prend des module javascript en paramètre ;
  - A nous de le configurer pour lui indiquer lesquels



Mise en place

- On va créer un projet vierge ;
- Notre projet utilisera « npm », un gestionnaire de packets ;
- Pour initialiser un projet npm, il suffit de lancer la commande :

```
> npm init
```

à la racine de votre projet !

- Puis les dépendances npm nécessaires à notre projet :

```
> npm install --save vue vue-router webpack webpack-cli
```

Architecture de nos fichiers

- Voici une architecture de fichiers que je vous propose :
  - C'est celle que l'on voit très souvent dans les projets conséquents VueJS

src/

main.js -> Porte d'entrée de notre application !
  - Parfois appelé App.js ou index.js

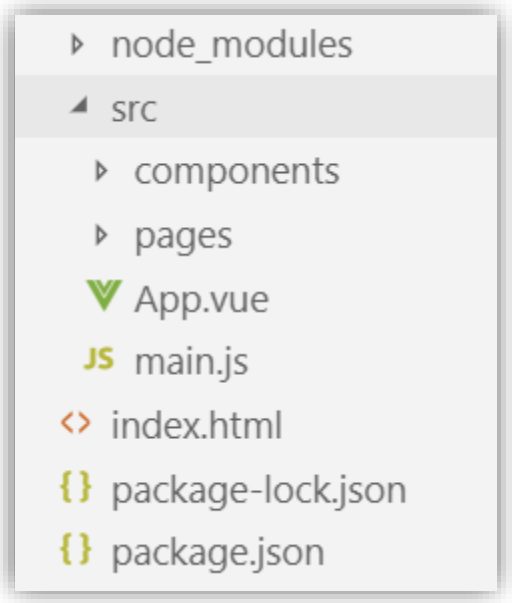
App.vue -> Composant principal

pages/ -> Dossier qui contiendra nos Smart components

components/ -> Dossier qui contiendra nos Dumb components

Architecture de nos fichiers

▷ Ce qui nous donne :



▷ L'extension .vue permettra à nos plugins de repérer nos entités vue spécifique afin de leur appliquer un traitement spécifique

Transpilation

▷ Commençons par le contenu du fichier « main.js » :

src/main.js

```
import Vue from 'vue'
import App from './App.vue'

new Vue({
        el: '#app',
        render: h => h(App)
});
```

Instance de « createElement » (*h pour hyperscript*)

▷ « render » est une fonction qui prend en paramètre un composant pour enclencher l'affichage sur la page

▷ Ici, on utilise des fonctionnalités ES6+ qui seront transpilées par la suite en ES5 ou moins à notre convenance

Notre composant principal

▷ Il va rester très simple pour l'instant :

src/App.vue

```
<template>
        <div>
                <h2>Bienvenue à la formation VueJS !</h2>
        </div>
</template>
```

▷ Comme vous pouvez le constater, on a intégrer du contenu HTML dans une page JS.

▷ Pas d'inquiétude, les plugins Webpack sauront interpréter tout ça

Configuration Webpack

- Webpack prend des points d'entrée et génère des bundle en résultat
  - Autant de bundle que de points d'entrée

- Une fois le point d'entrée récupéré, il va dérouler le fil et appliquer ses « plugins » & « loaders » sur les fichiers et lignes de code qu'il lira

- Le fichier de configuration webpack est définit par un fichier .js
  - On peut construire différent fichiers de configuration (dev / prod / test)

Configuration Webpack

- Créons un fichier « webpack.config.dev.js » à la racine de notre projet
  - Représentera la configuration de l'environnement de dev

webpack.config.dev.js

```
"use strict";

const { VueLoaderPlugin } = require("vue-loader");

module.exports = {
            mode: "development",
            entry: ["./src/main.js"], // Porte(s) d'entrée(s)
            module: {
                        rules: [
                        // Définition de nos loaders
                                    {
                                    // Appliquer le loaders vue-loader aux fichiers avec l'extension .vue
                                                test: /\.vue$/,
                                                use: "vue-loader"
                                    }
                        ]
            },
            // Chargement de nos plugins
            plugins: [new VueLoaderPlugin()]
};
```

Nécessaire pour utiliser vue-loader
(Agit également en tant que chef d'orchestre pour coordonner les autres plugins
Autour du fichier .vue actuellement lu)

Configuration Webpack

- Les plugins permettent d'appliquer un comportement générique
- Il faut les installer pour les utiliser :

```
> npm install --save-dev vue-loader vue-template-compiler vue-style-loader css-loader
```

Le plus important !

- Les loaders quant à eux agissent de manière locale
  - On peut également appliquer des loader sur nos lignes de code

Script build

- Avec npm, vous avez la possibilité de définir des scripts
  - Dans la propriété scripts du fichier « package.json »

package.json

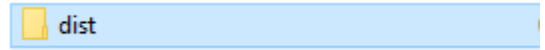
```
"scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "build:dev": "webpack --config webpack.config.dev.js"
},
```

- Pour lancer le script :

```
> npm run build:dev
```

- Vous constaterez qu'un dossier « dist » a été créé :

Contient le résultat du build



Brancher les fils

- Enfin, il va être nécessaire de constituer le contenu du fichier « index.html » !

index.html

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8" />
        <title>Webpack configuration ☺</title>
</head>

<body>
        <div id="app"></div>

        <script src="dist/main.js"></script>
</body>
</html>
```

- Et voilà !

Et maintenant?

- Webpack est configuré sur notre projet !
- Aucun changement pour l'instant
- Mise en place des composants monofichiers
- Configuration simplifiée
- Commençons par mettre en place la transpilation
  - Pour ce faire, on utilisera le très fameux babel

Mise en place de babel

```
module.exports = {
        ...
        module: {
                rules: [
                // Définition de nos loaders
                        {
                        // Appliquer le loaders vue-loader aux fichiers avec l'extension .vue
                                test: /\.vue$/,
                                use: "vue-loader"
                        },
                        // S'appliquera aux extensions '.js'
                        // Mais également aux blocs '<script>' présent dans les fichiers '.vue'
                        // C'est vue-loader qui s'occupera d'indiquer au plugin de s'appliquer aux <script>
                        {
                                test: /\.js$/,
                                loader: 'babel-loader'
                        },
                ]
        },
        ...
```

Doit évidemment être téléchargé au préalable
en tant que repo npm

```
> npm install -D babel-loader @babel/core @babel/preset-env
```

Les meilleurs plugins

- UglifyJS : Permet de minifier & d'obfusquer votre code tout en appliquant le principe de Tree shaking
  - Pour la mise en prod

- Babel avec « syntax-dynamic-import » : Ce loader fournit par Babel permet de « lazy-loadé » nos imports

```
// Pas de lazy load ! Tout est chargé au départ
import foo from 'foo'
import bar from 'bar'
import baz from 'baz'

const myfun = () => {
          // Utiliser nos modules
}
```



```
const myfun = () => {
          // Les imports sont chargé au runtime
          return Promise.all([import("foo"), import("bar"),
          import("baz")]).then(([foo, bar, baz]) => {
                    // Réaction une fois tout chargé
          });
};
```

VueCLI

▷ La VueCLI est un helper qui vous permettra de mettre en place vos architecture de projet très facilement ;

▷ Se base sur Webpack ;

▷ VueCLI vous fournit :

- Une architecture de base ;
- Le hot reload ;
- Des plugins adaptés à la majorité des projets VueJS ;
- Lance un serveur local ;
- Appliquer un watcher sur tous vos fichiers ;
- *etc.*

Installation

- Commençons par l'installer :

```
> npm install -g "@vue/cli"
```

- Puis créons un projet :

```
> vue create [NOM_PROJET]
```

- Ensuite, dirigez vous à la racine du projet et lancez :

```
> npm run serve
```

Composant généré

src/App.vue

```
<template>
        <div id="app">
                <img alt="Vue logo" src="./assets/logo.png">
                <HelloWorld msg="Welcome to Your Vue.js App"/>
        </div>
</template>

<script>
        import HelloWorld from "./components/HelloWorld.vue";

        export default {
                name: "app",
                components: {
                        HelloWorld
                }
        };
</script>

<style>
#app {
        font-family: "Avenir", Helvetica, Arial, sans-serif;
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
        text-align: center;
        color: #2c3e50;
        margin-top: 60px;
}
</style>
```

Permet de déclarer les composants qui seront utilisés dans cette instance

Composants monofichiers

▷ Comme vous pouvez le constater, vue / style et logique se trouvent dans le même fichier

▷ Tout comme React, c'est la bonne pratique en Vue

▷ Webpack se chargera d'interpréter ces 3 blocs séparément

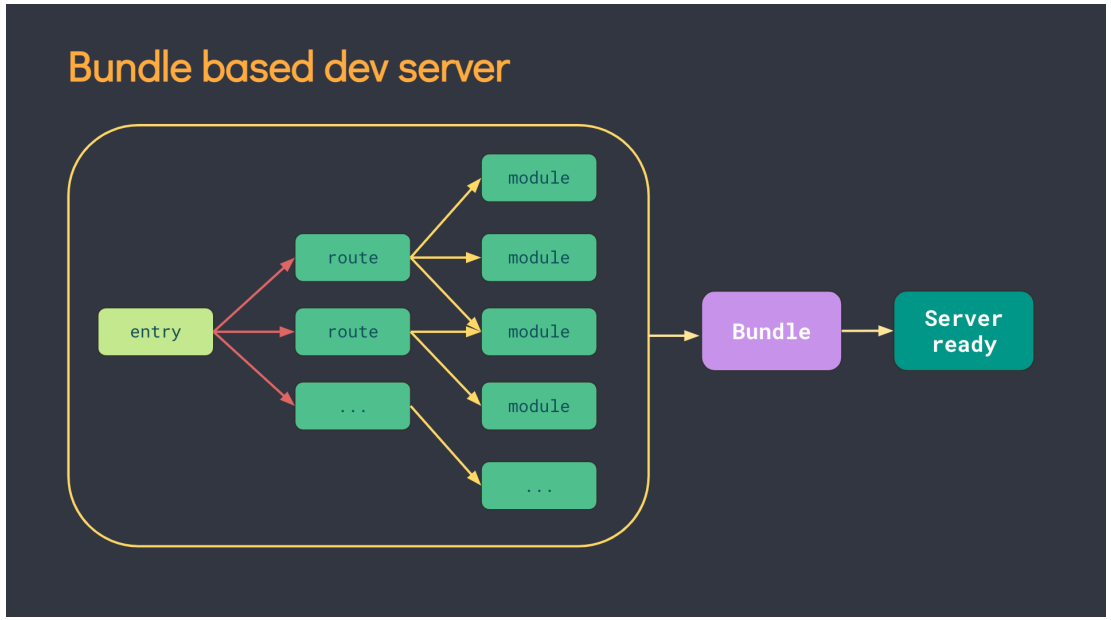
▷ On peut configurer la manière dont webpack gère le build, en savoir plus

Et Vite ?

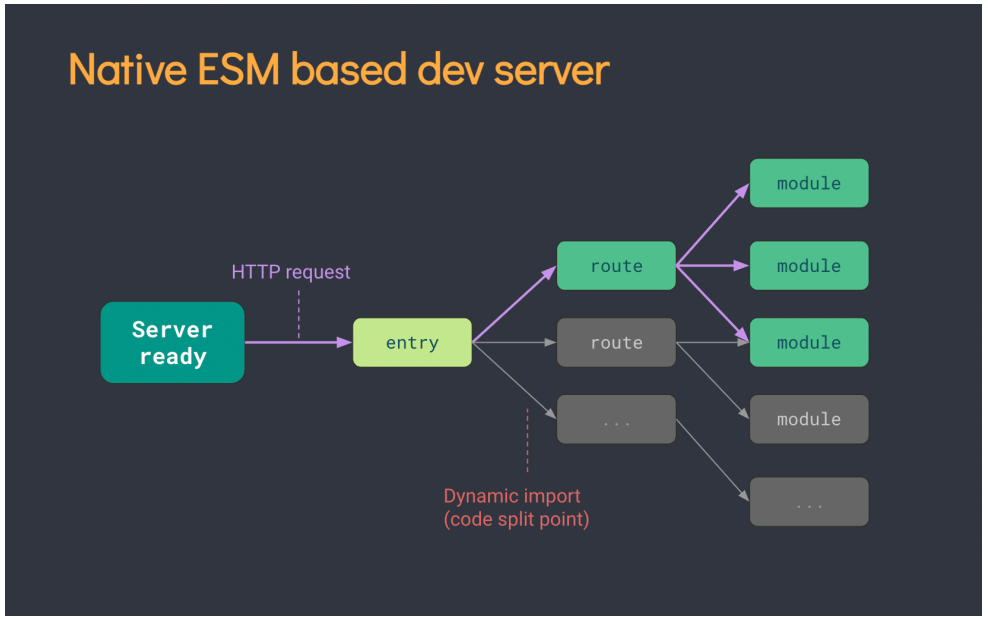
- Nouvel outil
  - Remplace complètement Webpack !
- Adapté pour Vue 
- Réduit la quantité de dépendances qui existaient avec Webpack
  - Ce qui provoquait de longue attente lors des build
  - La preuve avec Angular aujourd'hui…

ES Module in browser

Webpack



Vite

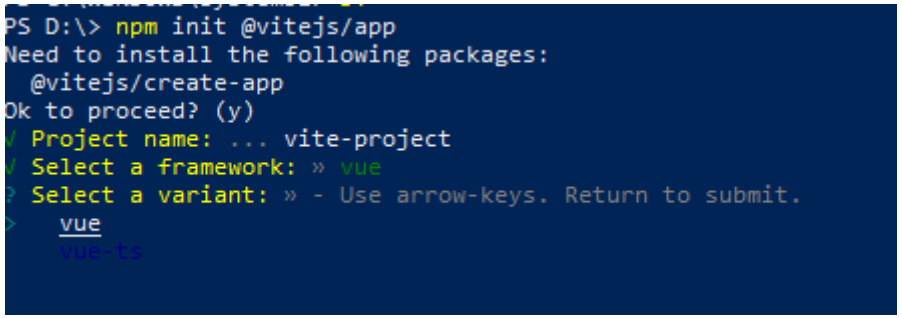


[Chargement des modules à la volée]

Mise en place

```sh
$ npm init @vitejs/app
```

- Il s'agira ensuite d'intéragir avec le terminal



Une logique presque similaire

```
<script lang="ts">
  import { ref, defineComponent } from 'vue'
  export default defineComponent({
      name: 'HelloWorld',
      props: {
          msg: {
              type: String,
              required: true
          }
      },
      setup: () => {
          const count = ref(0)
          return { count }
      }
  })
</script>
```

defineComponent renvoie simplement l'objet. Permet la compatibilité avec TypeScript. En savoir plus ici

C'est un template ref ! En phase avec le principe de composition Vue 3

```
<button type="button" @click="count++">count is: {{ count }}</button>
```

Migrer notre projet vers VueCLI

▷ Récupérer le projet précédemment codé pour le migrer vers un projet Vite

- Pour ce faire, commencez par générer un projet avec « npm init @vitejs/app »
- Puis créer un fichier .vue qui contiendra les informations des composants

Questions