Programmation Web – Avancé

JavaScript & Node.js

Module bundler

Presentation template by SlidesCarnival

Module bundler

Comment gérer les dépendances de votre frontend de manière raisonnable ?

Pourquoi utiliser un module bundler ?

- **DEMO : Création d'un frontend pour une pizzeria : Step -2 / Option A : frontend classique HTML / CSS / JS**
  - Quelles sont les faiblesses de ce type de développement ?
  - Points forts ?

Pourquoi utiliser un module bundler ?

- DEMO : Création d'un frontend pour une pizzeria : Step -2 / Option B : frontend classique HTML / Bootstrap / JS

Pourquoi utiliser un module bundler ?

- Gestion moderne des dépendances (et de leur dépendances ;)
- Transformation des assets : images, SASS vers CSS...
- JS pour cibler des browsers spécifiques
- Besoin d'outils de développement plus avancés : hot reload...
- ...

Module bundlers

- Gestion aisée des modules, packages et de leurs dépendances
- Utilisation de packages faits pour **Node.js** via le browser => Installation de **Node.js** LTS [54.]
- Gestion aisée des assets, transpilation de code...
- Parmis les plus connus : **webpack** [98.], **browserify**, **parcel**

Gestion moderne d'une application et de ses packages

- **Option A** : Création et configuration de son application « from scratch » 
- **Option B** : Utilisation d'un boilerplate comprenant le squelette d'une app et ses dépendances
- **Option C** : Utilisation d'un Framework : React, Vue, Angular…

Option A : Création et configuration de son application « from scratch »

- Fichier de configuration de votre application : création d'un **package.json** pour votre application (après création d'un répertoire)
  - **npm init –y**
  - Enlever le « main entry »: – **"main": "index.js"**
- Installation de **webpack** via npm :
  - **npm i webpack-cli webpack -D**

Option A : Création et configuration de son application « from scratch »

- Création d'une structure standard pour votre application

```
./package.json
./webpack.config.js
./dist
   ./dist/index.html
./src
   ./src/index.js
```

Option A : Création et configuration de son application « from scratch »

- Intégration de votre « bundle file » ./dist/main.js généré par webpack au sein de votre fichier HTML

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

Option A : Création et configuration de son application « from scratch »

- Générer le bundle sans fichier de configuration : npx webpack

Option A : Création et configuration de son application « from scratch »

- Création d'un ficher de configuration webpack.config.js à la racine :

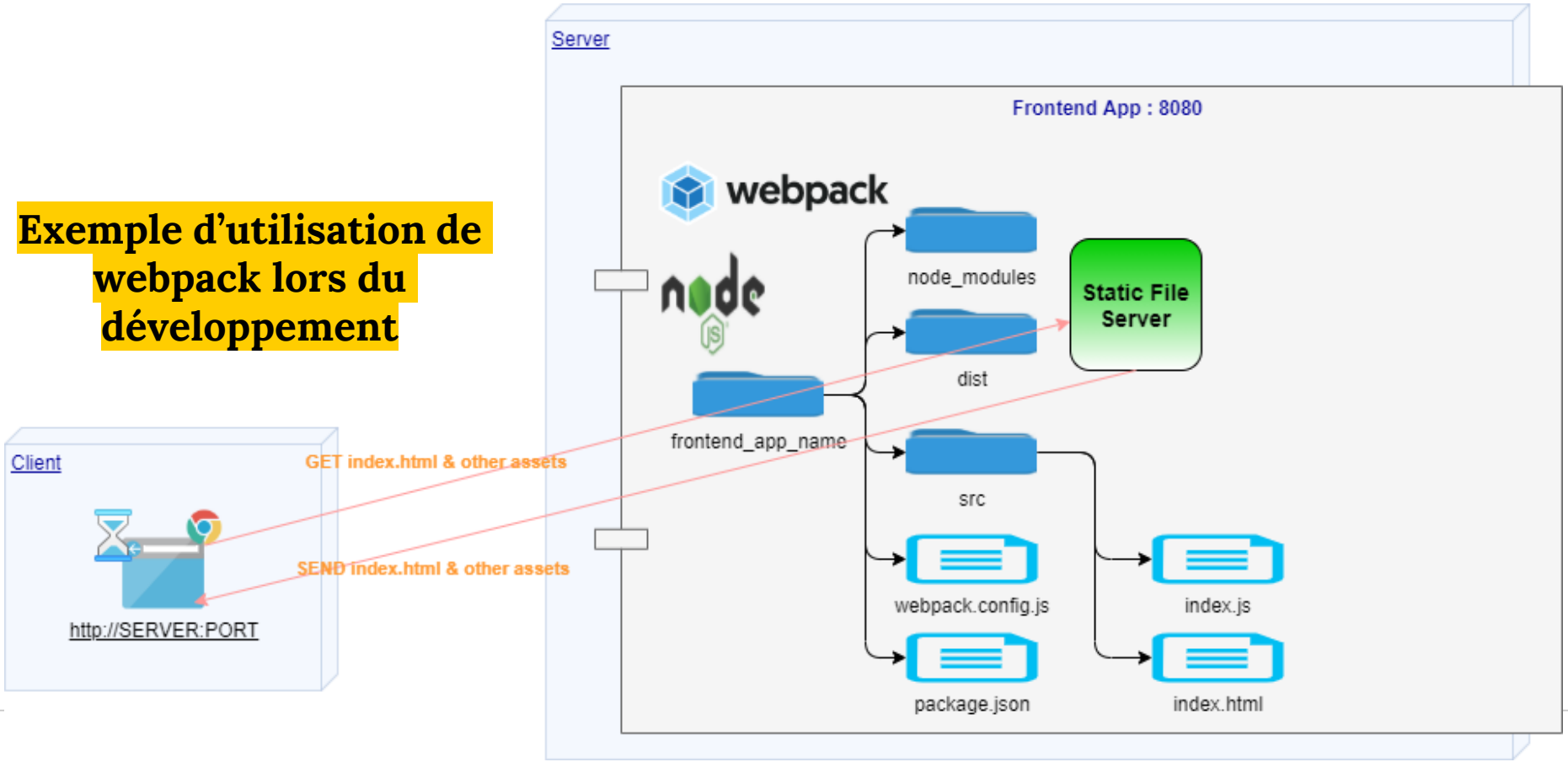
```
const path = require("path");

module.exports = {
  entry: "./src/index.js",
  output: {
    path: __dirname + "/dist",
    filename: "main.js",
  },
};
```

./package.json
./webpack.config.js
./dist
  ./dist/index.html
./src
  ./src/index.js

Option A : Création et configuration de son application « from scratch »

- NPM scripts : utiliser **npm run dev** ou **npm run build** au lieu de **webpack-cli**. Dans **./package.json**
  - \+ "build": "webpack" ou
  - \+ "build": "webpack --mode production"
  - \+ "dev": "webpack --mode development"

Exemple d'utilisation de webpack lors du développement



Option A : Création et configuration de son application « from scratch »

- DEMO : Création d'un frontend pour une pizzeria : Step -1 : migration partiel du frontend classique vers un frontend moderne utilisant Webpack sans fichier de configuration
  - README.md : reprend toutes les instructions pour générer « Part 1 » à partir de « Part 0 »

Option A : Création et configuration de son application « from scratch » : Configuration utile d'un boilerplate

- Installation & configuration d'un serveur de développement (**devServer**) :
  - proxy sous Webpack
  - Servir **index.html** quand route n'amène à aucun fichier
  - Rendre le serveur accessible de l'extérieur
  - Ouvrir le browser par défaut à l'URL « localhost »

Option A : Création et configuration de son application « from scratch » : Configuration utile d'un boilerplate

- Gestion du fichier **index.html** :
  - Création automatique d'un fichier **index.html** dans **/dist** sur base du template **/src/index.html**
  - **/dist** enlevé de git
- Charger les images à l'aide de Webpack de **/src/img** vers **/dist** (éventuellement les transformer)

Option A : Création et configuration de son application « from scratch » : Configuration utile d'un boilerplate

- Installation et configuration de la gestion du CSS : utilisation du **style loader** et du **css loader**
- Inclure Bootstrap dans un projet utilisant Webpack : [95.]
  - Installer Bootstrap et dépendances : **npm i bootstrap jquery @popperjs/core -D**
  - **index.js** :

Option A : Création et configuration de son application « from scratch » : Configuration utile d'un boilerplate

- En savoir plus sur webpack [98.]
  - Assets Management (CSS, images…)
  - Configuration pour le proxy du devServer
  - Installation des packages d'un boilerplate :
    npm i webpack-dev-server css-loader style-loader html-loader babel-loader @babel/core @babel/preset-env –D

Option A : Création et configuration de son application « from scratch »

- DEMO : Création d'un frontend pour une pizzeria : Step 0 : migration complète du frontend classique vers un frontend moderne utilisant Webpack
  - README.md : reprend toutes les instructions pour générer « Step 2 » à partir de « Step 1 »

Option B : Utilisation d'un boilerplate comprenant le squelette d'une app et ses dépendances

- DEMO : Création d'un frontend pour une pizzeria : Step 1 : frontend à partir d'un boilerplate et ajout moderne d'une librairie pour afficher une horloge
  - README.md : reprend toutes les instructions pour générer « Step 3 » à partir d'un boilerplate (« Step 2 »)

Webpack inclus

- Toutes les prochaines applications incluront Webpack pour un développement moderne