

NodeJS et NPM

1WEBD – Javascript Web Development



Sommaire

1. Introduction.
2. NodeJS.
3. NPM.



1. Introduction

1. Introduction

Définition

- Node.js est un environnement d'exécution open-source qui permet d'exécuter du code JavaScript côté serveur
- il est construit sur le moteur JavaScript V8 de Chrome

1. Introduction

Histoire

- JavaScript est née comme langage client (donc à exécuter côté utilisateur, dans son navigateur)
- NodeJS est créé en 2009 par Ryan Dahl
- Complètement Open Source

1. Introduction



2. NodeJS

2. NodeJS

Installation

- la façon la plus simple est d'aller sur nodejs.org pour télécharger l'installateur pour votre os
- la méthode plus intéressante est d'utiliser volta.sh ou [nvm](https://nvm.js.org) (permettent d'avoir plusieurs versions installées en même temps)
- pour vérifier que l'installation ai réussi, on peut taper **node -v** dans un terminal pour afficher la version de NodeJS qu'on a installé

2. NodeJS

Création d'un projet

- pour créer un nouveau projet, on peut utiliser la commande **npm init**
- cette commande aide à générer un fichier **package.json**
- les questions sont optionnelles sauf dans des cas particulier (ex: créer un projet qui exporte un exécutable)

2. NodeJS

Package.json – Structure de base

- **name:** Nom de votre projet.
- **version:** Version actuelle de votre projet, suivant le formatage sémantique.
- **description:** Brève description de votre projet.
- **main:** Point d'entrée principal du projet (fichier de démarrage).

2. NodeJS

Package.json – Dependances

- **dependencies:** Liste des paquets NPM requis pour l'exécution de l'application.
- **devDependencies:** Liste des paquets nécessaires uniquement pour le développement.

2. NodeJS

Package.json – Autres champs

- repository: Informations sur le dépôt de code source.
- author: Auteur du projet.
- license: Type de licence sous laquelle le projet est publié.

2. NodeJS

Exécution

- supposons avoir un fichier app.js contenant ce code
- on peut exécuter ce fichier avec la commande **node app.js**



```
console.log("hello world")
```

2. NodeJS

Exécution

- on peut aussi rajouter un script dans l'objet scripts
- par exemple, on peut créer un script **start** qui exécutera **node app.js**
- on peut donc maintenant lancer **npm run start** pour exécuter app.js

2. NodeJS

Exécution

```
{
  "name": "node_demo",
  "version": "1.0.0",
  "description": "a simple demo for Supinfo",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node app.js"
  },
  "author": "",
  "license": "MIT"
}
```

2. NodeJS

Import Export

- on peut avoir plusieurs fichiers dans un projet NodeJS
- les fichiers peuvent exporter et importer des valeurs d'autres fichiers

2. NodeJS

Import Export avec require

- la méthode traditionnelle consiste à déclarer les exports dans l'objet **module.exports**
- ensuite, on peut l'importer avec **require**

2. NodeJS

Import Export avec require

```
function addition(a, b) {  
    return a + b;  
}  
  
function soustraction(a, b) {  
    return a - b;  
}  
  
module.exports = {  
    addition,  
    soustraction  
};
```

```
const math = require('./math.js');  
  
console.log(math.addition(5, 3)); // Affiche 8  
console.log(math.soustraction(5, 3)); // Affiche 2
```

2. NodeJS

Import Export avec des modules

- avec ES6 et l'introduction, on peut utiliser les mots-clés **export** et **import**
- NB: selon les versions de NodeJS, il faut spécifier « 'type': 'module' »

Import Export avec des modules

```
export function addition(a, b) {  
    return a + b;  
}  
  
export function soustraction(a, b) {  
    return a - b;  
}
```

```
import { addition, soustraction } from './math.js';  
  
console.log(addition(5, 3)); // Affiche 8  
console.log(soustraction(5, 3)); // Affiche 2
```

2. NodeJS

Type de fichiers

- Dans les dernières version NodeJS (18 et plus), on peut lire des fichiers .js, .cjs et .mjs

2. NodeJS

Type de fichiers - CJS

- Les fichiers .cjs sont associés au format CommonJS, qui était le standard dans Node.js avant l'introduction des modules ES6
- Fonctionne nativement dans NodeJS
- Ne supporte pas certaines fonctionnalisées modernes (comme la syntaxe import/export)

2. NodeJS

Type de fichiers - MJS

- Les fichiers .mjs sont utilisés pour les modules ECMAScript (ESM), la norme dans le développement JavaScript moderne
- Type de fichier plus moderne
- Même support sur les navigateurs, donc facilite le partage de code
- Supporte le chargement asynchrone et le tree shaking
- Peut causer des problèmes lors de l'intégrations avec des modules CommonJS
- Peut nécessiter une configuration supplémentaire pour être utilisé dans des projets plus anciens

2. NodeJS



3. NPM

3. NPM

Definition

- NPM (Node Package Manager)
- système de gestion de paquets par défaut pour Node.js
- il permet aux développeurs de partager et d'utiliser des modules de code
- facilite la construction de projets NodeJS

3. NPM

Fonctionnement

- quand on installe un paquet, il est téléchargé dans le dossier **node_modules** du projet
- on peut ensuite l'emporter en suivant la documentation du module

3. NPM

Gestion des Pacquets - package.json

- cœur de tout projet NodeJS
- contient toutes les métadonnées du projet (nom, membres etc)
- quand un paquet est installé avec **npm**, il est ajouté à la liste des dépendances dans le **package.json**

3. NPM

Gestion des Pacquets - Installation de paquets

- pour installer un paquet, on peut utiliser la commande **npm install <nom_du_paquet>**
- - on peut aussi spécifier le tag (la version en quelque sorte) du paquet avec un @
npm install test@0.0.1

3. NPM

Bonnes Pratiques

- utiliser des versions spécifiques de paquets (ou des plages de versions) permet d'éviter de casser l'application à cause d'une dépendance
- ajouter **node_modules** au fichier **.gitignore** permet de faire ignorer le dossier par Git (dossier très lourd et qui peut causer des conflits au moment de merge ou pull)

3. NPM



