INFORMATIQUE
TECHNIQUE

Expert

Angular et Node.js

Optimisez le développement de vos applications web avec une architecture MEAN

Fichiers complémentaires à télécharger

Pierre POMPIDOR

eni

Table des matières

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence ENI de l'ouvrage **EIANGNOD** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Chapitre 3

Les extensions à JavaScript pour utiliser des classes

Chapitre 4
La plateforme Node.js

Chapitre 5
Le SGBD NoSQL MongoDB

Chapitre 6
Introduction au framework applicatif Angular

Chapitre 7
Angular : les templates, bindings et directives

Chapitre 8

Angular et la connexion à Node.js : les services

Chapitre 9
Angular et la gestion des routes internes

Chapitre 10
Angular et la visualisation d'informations

Chapitre 11
Test et déploiement

Chapitre 4
La plateforme Node.js

1. Présentation de Node.js

Node.js est un environnement permettant d'exécuter du code JavaScript hors d'un navigateur. À l'heure de la rédaction de cet ouvrage, il repose sur le moteur JavaScript V8 développé par Google pour ses navigateurs Chrome et Chromium.

Son architecture est modulaire et événementielle. Il est fortement orienté réseau en possédant pour les principaux systèmes d'exploitation (Unix/Linux, Windows, macOS) de nombreux modules réseau (dont voici les principaux par ordre alphabétique : DNS, HTTP, TCP, TLS/SSL, UDP). De ce fait, il remplace avantageusement, dans le cadre qui nous intéresse ici (c'est-à-dire la création et la gestion d'applications web), un serveur web tel qu'Apache.

Créé par Ryan Lienhart Dahl en 2009, cet environnement est devenu rapidement très populaire pour ses deux qualités principales :

– Sa légèreté (en corollaire de sa modularité).
– Son efficacité induite par son architecture monothread (en corollaire de la gestion événementielle que propose nativement l'environnement JavaScript).

Intégrer Node.js dans le développement d'applications web participe donc à la logique actuelle de rendre les opérations d'accès aux données les moins bloquantes possible (pour dépasser la problématique dite du « bound I/O » selon laquelle, avant toute autre cause, la latence globale d'une application est due au temps de latence des accès aux données).

Node.js permet donc, pour les applications web, de créer des serveurs extrêmement réactifs.

Dans ce qui suit, vous allez :

– Installer et tester Node.js sous Linux, Windows ou macOS.
– Créer un serveur HTTP renvoyant une chaîne de caractères.
– Mettre en œuvre un module.
– Créer un serveur HTTP utilisant le module express invoqué sur une route REST et renvoyant des données formatées en JSON, d'abord en totalité, puis filtrées sur une propriété.

Dans ce chapitre, nous n'introduirons que quelques modules (et fonctions) de Node.js.

La documentation complète des modules est disponible à cette adresse : https://nodejs.org/api/

2. Installation et test de Node.js

2.1 Création du fichier de test

Pour tester Node.js, vous allez dans un premier temps créer un code JavaScript qui va être le plus simple possible, et le faire exécuter par Node.js.

▶ Créez donc le fichier testDeNode.js qui ne comprend qu'une ligne de code :

```
console.log("Test de Node");
```

2.2 Installation et test de Node.js sous Ubuntu

- Pour installer Node.js sous Ubuntu, le plus simple est d'utiliser la commande `curl` et le gestionnaire de paquets en ligne de commande (`apt-get`) :

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt-get install nodejs
```

- Un lien symbolique nommé `node` peut être créé pour lancer plus naturellement vos serveurs :

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

- Ouvrez un terminal (shell) et exécutez le fichier de test :

```
node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

Une procédure complète est en ligne sur http://doc.ubuntu-fr.org/nodejs.

2.3 Installation et test de Node.js sous Windows

L'installation de Node.js et son test sous Windows vont se dérouler en quatre étapes :

- Téléchargez l'installateur Windows Installer en vous rendant sur le site officiel de Node.js : https://nodejs.org/en/download
- Exécutez l'installateur (le fichier .msi précédemment téléchargé) en acceptant les conditions d'utilisation et le paramétrage par défaut.
- Redémarrez votre ordinateur.
- Ouvrez l'invite de commandes et exécutez le fichier de test :

```
node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

2.4 Installation et test de Node.js sous macOS

L'installation de Node.js et son test sous macOS vont se faire en trois étapes :

- Téléchargez le package d'installation pour macOS (Macintosh Installer) en vous rendant sur le site officiel de Node.js : https://nodejs.org/en/download/
- Ouvrez un terminal et installez le package :

```
pkg install nomPackage.pkg
```

- Exécutez le fichier de test :

```
node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

3. La modularité de Node.js

3.1 Les modules et les packages

Une des principales forces de Node.js est d'être modulaire (et notamment de proposer de nombreux modules réseau). Si certains de ces modules sont installés directement en même temps que Node.js, la plupart doivent être installés à la demande.

Lors de la création d'une application qui exige l'installation de modules, deux méthodes sont possibles pour effectuer celle-ci :

– Directement avec le gestionnaire de modules npm (et son option `install`).
– Indirectement (mais toujours avec npm) via la spécification des dépendances de l'application (c'est-à-dire des modules nécessaires à celle-ci) dans un fichier nommé package.json.

Un module est utilisé dans une application avec la fonction `require()` :

```
var moduleDansVotreApplication = require('<nomDuModule>');
```

Votre application Node.js peut être elle-même réutilisée comme module sous certaines conditions qui seront présentées ultérieurement.

Attardons-nous sur un point un peu subtil : la distinction entre modules et packages.

Les modules sont les briques conceptuelles d'une application Node.js. Un module peut être organisé en plusieurs codes JavaScript et dépendre d'autres modules. Ainsi, toutes les ressources nécessaires à un module (les codes, le fichier package.json spécifiant ses dépendances...) sont regroupées dans un package qui, de fait, est un dossier.

Donc, si les deux termes sont quasiment interchangeables, le terme « module » renvoie plus à la fonctionnalité globale, et « package » au dossier et à l'organisation des fichiers de code qui se trouvent dans celui-ci.

3.1.1 Le gestionnaire de modules de Node.js : npm

npm (*Node.js Package Manager*) est le gestionnaire de modules de Node.js (il est installé avec celui-ci).

Les modules sont installés globalement dans le dossier node_modules, situé au niveau des répertoires système si l'option `-g` est utilisée :

```
npm install -g <module>
```

ou sinon (sans l'option `g`) dans le répertoire courant (mais également dans un dossier nommé node_modules).

3.1.2 Spécification des dépendances : le fichier package.json

Pour spécifier les dépendances nécessaires à la création d'une application Node.js (c'est-à-dire les modules associés aux packages nécessaires à celle-ci), il est recommandé de créer un fichier nommé package.json.

Dans le contexte d'un fichier package.json, nous ne parlerons plus que de packages (et non de modules).

Voici un schéma minimal de ce fichier :

```
{
  "name":        "<nom de l'application>",
  "version":     "<version de l'application>",
  "description": "<description de l'application>",
  "author":      "<nom de l'auteur de l'application>",
  "main":        "<code à exécuter comme point d'entrée>",
```

```
  "scripts": {
      "start": "node <code à exécuter>"
  },
  "dependencies": {
    "<nom du package>": "<version minimale du package à installer>",
    ...
  }
}
```

Pour installer les modules nécessaires, la commande suivante doit être exécutée :

```
npm install
```

Et voici celle qui va lancer le serveur :

```
npm start
```

Pour créer un squelette de fichier *package.json*, utilisez la commande suivante :

```
npm init --yes
```

Dans ce cas, la valeur de la propriété `main` est initialisée à index.js. Expliquons un peu l'intérêt de cette propriété :

Si votre application devient un package (comprenant un ou plusieurs codes réutilisables), la propriété `main` désigne le code qui est le point d'entrée dans le package lors de l'exécution de l'instruction `require()`.

3.2 Création d'un premier serveur Node.js de test

Vous allez écrire votre premier serveur (le bien classique « Hello World ») en utilisant le module HTTP qu'offre Node.js.

▶ Saisissez le code de ce serveur dans le fichier helloAvecNode.js :

```
var http = require('http');

var server = http.createServer(function(request, response){
    response.end('Hello World de Node.js');
});

server.listen(8888);
```