SERVEUR NODE.JS

Node et npm

1. Node

Node (ou Node. js) est un environnement d'exécution open-source, multi-plateforme, qui permet de créer des applications et des outils côté serveur (back-end) en JavaScript. Node ajoute également des fonctionnalités que le JavaScript du navigateur standard ne possède pas, comme par exemple l'accès au système de fichiers local. Il s'agit bien sûr des fichiers et des répertoires stockés sur l'ordinateur ou le serveur où Node.js est en cours d'exécution.

2. NPM

NPM (Node Package Manager), est un gestionnaire de packages (ou modules) open source largement utilisé dans l'écosystème Node.js.

NPM est un outil qui permet aux développeurs Node.js de gérer et de distribuer des packages de code JavaScript réutilisables. Ces packages peuvent contenir des bibliothèques, des frameworks, des modules, des dépendances et d'autres ressources utiles pour le développement d'applications Node.js.

NPM offre plusieurs fonctionnalités essentielles, notamment :

- ✓ Installation de packages : Les développeurs peuvent utiliser NPM pour installer des packages tiers depuis le registre NPM, ce qui simplifie l'intégration de bibliothèques tierces dans leurs projets.
- ✓ Gestion des dépendances : NPM permet de spécifier les dépendances requises par un projet, de manière à garantir que les bonnes versions des packages sont utilisées. Il génère également un fichier "package.json" pour suivre ces dépendances.
- ✓ Publication de packages : Les développeurs peuvent publier leurs propres packages sur le registre NPM, les rendant ainsi accessibles à d'autres utilisateurs.
- ✓ Scripts personnalisés : NPM permet de définir des scripts personnalisés pour automatiser des tâches courantes, comme la construction, les tests, le déploiement, etc.

NPM est généralement inclus avec l'installation de Node.js, ce qui en fait un outil incontournable pour le développement JavaScript côté serveur. Il facilite la gestion des dépendances, la collaboration avec d'autres développeurs et la distribution de code, contribuant ainsi à l'écosystème Node.js dynamique et à la productivité des développeurs.

Initier le projet

3. Pré-requis

- ✓ Vérifiez que node.js est bien place sur votre machine en tapant dans un terminal la commande `node --version` (ou `node -v`) .
- ✓ Vérifiez que npm est bien place également sur votre machine en tapant dans un terminal la commande `npm --version` (ou `npm -v`) .

4. Préparation du projet

- ✓ Créez le dossier du projet et ouvrez le avec **VSCode**.
- ✓ Ouvrez une console dans le dossier du projet à partir de VSCode (ctrl + ù).
- ✓ Initialisez un dépôt **Git** en exécutant la commande `git init`.
- ✓ Créez un fichier `.gitignore` contenant la ligne `node_modules/` pour éviter d'envoyer ce dossier sur votre dépôt distant.

Les dépendances

Nous allons utiliser le framework **Express** (http://expressjs.com/) et **Nodemon** (https://nodemon.io/) pendant le développement de cette partie du projet.

1. Express

L'utilisation d'Express.js est une approche très populaire et efficace pour créer des serveurs **HTTP** en **JavaScript**. Express.js simplifie la création de serveurs web en fournissant une interface plus conviviale et des fonctionnalités avancées pour la gestion des routes, des requêtes et des réponses.

2. Nodemon

Nodemon est un utilitaire qui surveille les fichiers de votre application, y compris le fichier de démarrage (dans cet exemple, `server.js`), ainsi que les fichiers de l'application, tels que les routes, les contrôleurs, les modèles, etc. L'objectif de Nodemon est de redémarrer automatiquement le serveur dès qu'il détecte un changement dans l'un de ces fichiers.

Si vous apportez des modifications au fichier `server.js` ou à tout autre fichier dans l'application, Nodemon détectera ces modifications et redémarrera le serveur pour refléter les nouvelles modifications. Cela permet de développer plus rapidement sans avoir à redémarrer manuellement le serveur à chaque modification.

L'utilisation de Nodemon est particulièrement utile pendant la phase de développement pour améliorer la productivité. Cependant, il est courant de ne pas utiliser Nodemon en production, car il est principalement conçu pour le développement.

3. Initialiser le projet

- ✓ Tapez `npm init` dans la console pour initialiser le projet. Laissez les options par défaut, nous les modifierons par la suite.
- ✓ Ce processus génère le fichier **package.json** à la racine du projet.
- ✓ Tapez `npm install express nodemon --dev` et patientez.
- ✓ Créez un fichier **server.js** qui contiendra le code de l'application Express, où vous définirez le comportement du serveur et des routes.

4. package.json

- ✓ Ouvrez le fichier **package.json**
- ✓ Modifiez le ainsi :

```
{
  "name": "tasks-management",          (personnaliser le nom de projet)
  "version": "1.0.0",
  "description": "",                   (personnaliser la description projet)
  "main": "index.js",
  "scripts": {
    "start": "nodemon server.js",      (ajouter ce script)
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Eric",                    (personnaliser le nom de l'auteur)
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
    "nodemon": "^3.0.1"
  }
}
```

5. Explication sur ce fichier package.json

- ✓ `"name": "tasks-management"` : C'est le nom du projet ou de l'application. Il doit être unique au sein du registre npm si vous prévoyez de publier votre projet.
- ✓ `"version": "1.0.0"` : La version actuelle du projet. Cela suit généralement une convention de versionnement (par exemple, Semantic Versioning) pour indiquer les changements importants dans votre projet au fil du temps.
- ✓ `"description": ""` : Une description optionnelle du projet. Vous pouvez y mettre une brève description de ce que fait l'application.
- ✓ `"main": "index.js"` : Le fichier JavaScript principal de l'application. Lorsque quelqu'un utilise le module en tant que dépendance, ce fichier est le point d'entrée par défaut.
- ✓ `"scripts"` : Cette section définit des scripts que vous pouvez exécuter à l'aide de la commande `npm run <script-name>`. Dans ce fichier :

- "start": "nodemon server.js" : Cela définit un script nommé "start". Lorsque vous exécutez `npm start`, il utilise **nodemon** pour exécuter le fichier **server.js**.
- "test": "echo \"Error: no test specified\" && exit 1" : Cela définit un script nommé "test". Lorsque vous exécutez `npm test`, il affiche le message d'erreur spécifié, indiquant qu'aucun test n'est spécifié, puis il quitte le processus avec un code d'erreur (1).

✓ "author": "Eric" : L'auteur du projet, c'est-à-dire la personne qui a créé le fichier package.json.

✓ "license": "ISC" : La licence sous laquelle votre projet est distribué. En l'occurrence, il s'agit de la licence ISC, qui est une licence open source permissive.

✓ "dependencies" : Cette section répertorie les dépendances du projet, c'est-à-dire les bibliothèques tierces dont l'application a besoin pour fonctionner correctement. Dans ce cas, deux bibliothèques :

- "express": "^4.18.2" : La dépendance vers Express.js, un framework web pour Node.js.
- "nodemon": "^3.0.1" : La dépendance vers Nodemon, un utilitaire qui facilite le développement en redémarrant automatiquement le serveur Node.js lorsque des modifications de fichiers sont détectées.

Le fichier **package.json** contient des métadonnées du projet, des scripts pour faciliter son exécution et des informations sur les dépendances nécessaires à son bon fonctionnement. Ces informations sont essentielles pour gérer, exécuter et distribuer une application Node.js.

Le serveur

1. server.js

✓ Ouvrez le fichier **server.js**

✓ Et prenez le code suivant :

```
const express = require("express");
const port = process.env.PORT || 5000;

const app = express();

app.get('/', (req, res) => {
    res.send("Hello World!");
});

app.listen(port, () => {
    console.log("Server is online!");
});
```

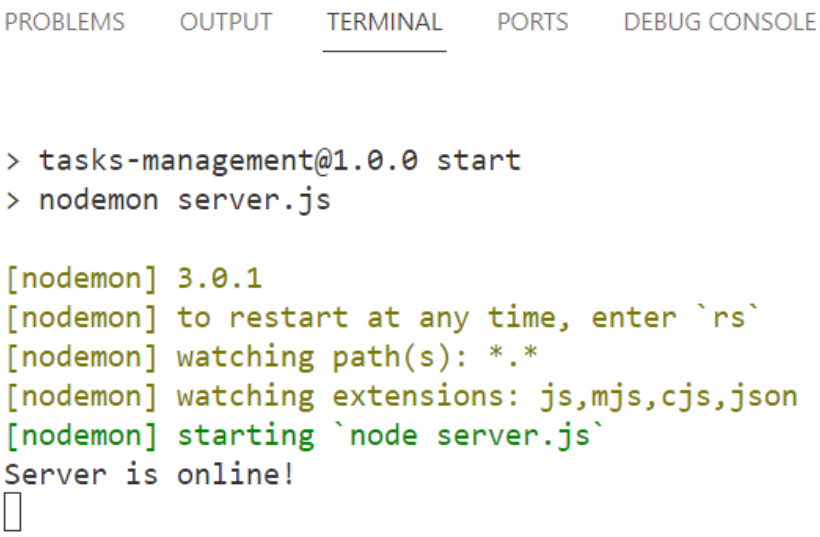
2. Explication sur le fichier server.js

- ✓ `const express = require("express")` : Cela importe le module Express.
- ✓ `const port = process.env.PORT || 5000` : Cela définit un port sur lequel le serveur écoutera les demandes entrantes. Il vérifie d'abord si une variable d'environnement PORT a été définie (ce qui est courant dans les environnements d'hébergement), sinon, il utilise le port 5000 par défaut.
- ✓ `const app = express();` : Ici, une instance d'Express est créée et stockée dans la variable app. Cette instance est l'objet principal à utiliser pour configurer votre application web.
- ✓ app.get('/', (req, res) => { res.send("Hello World!"); }); : Cette route configure une gestionnaire de requête pour le chemin racine '/'. Lorsqu'un utilisateur accède à la racine du site, cette fonction est exécutée. Elle envoie simplement la réponse "Hello World!" au navigateur de l'utilisateur.
- ✓ app.listen(port, () => { `console.log("Server is online!");` }): Cette ligne démarre le serveur Express sur le port spécifié (port) et utilise une fonction de rappel pour afficher un message dans la console une fois que le serveur est en ligne.

Ce code crée un serveur web très simple qui écoute sur un port spécifié (ou le port 5000 par défaut) et répond avec "Hello World!" lorsque vous accédez à la page racine du site. Il peut également servir des fichiers statiques à partir du répertoire "public". C'est un exemple de base pour démarrer une application web Node.js avec Express.

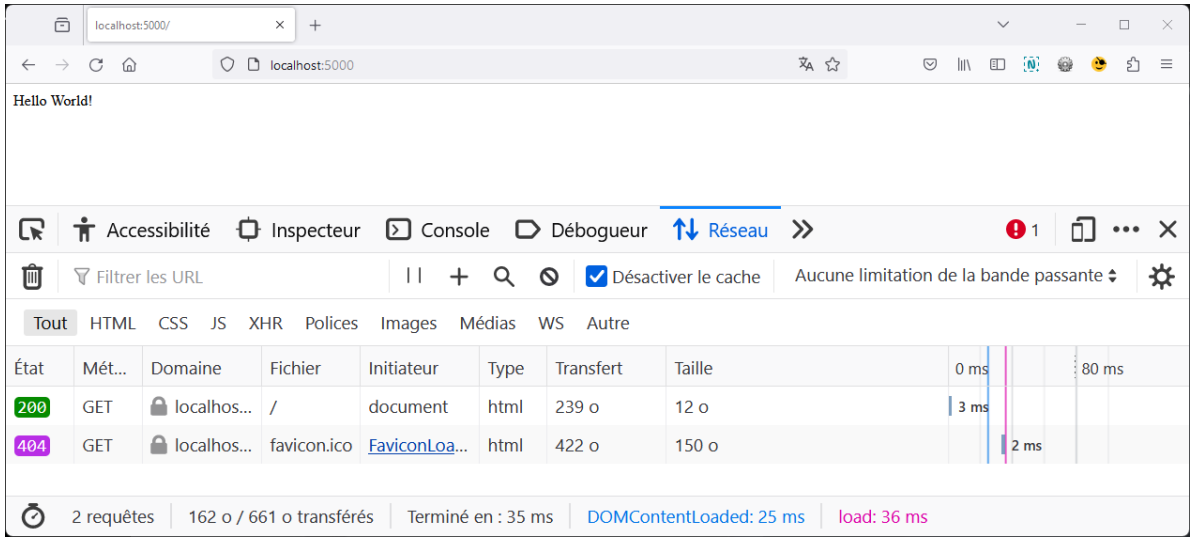
3. Lancer le serveur

- ✓ Dans le terminal de VSCode tapez la commande `npm start`



Résultat dans le terminal

✓ Ouvrez votre navigateur et tapez l'url localhost :5000



Résultat dans le navigateur

Il est possible de renvoyer le statut envoyer au navigateur en réponse à la méthode GET.

✓ Modifiez cette partie de code :

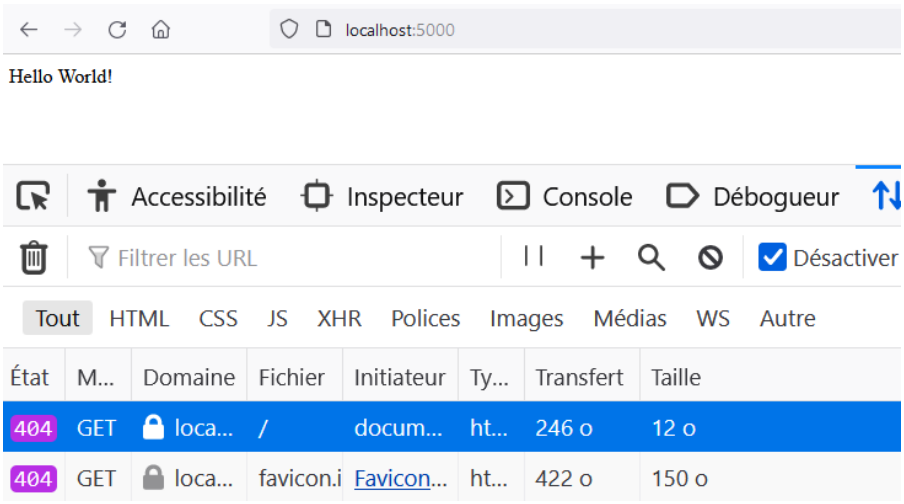
```
app.get('/', (req, res) => {
    res.send("Hello World!");
});
```

✓ Par :

```
app.get('/', (req, res) => {
    res.status(404).send("Hello World!");
});
```

✓ Le serveur est automatiquement relancer.

✓ Rafraichissez la page du navigateur.



4. Le dossier public

- ✓ Créez un dossier `public` dans lequel vous pourrez placer vos images, css et script JavaScript
- ✓ Créez un fichier `index.html` et mettez lui un titre en h1.
- ✓ Modifiez le code du fichier `server.js`

```
const express = require("express");
const port = process.env.PORT || 5000;

const app = express();

// ajoutez cette ligne pour spécifiez le dossier public
app.use(express.static('public'));
app.get('/', (req, res) => {
    // ajouter cette ligne et supprimez l'ancienne
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

app.listen(port, () => {
    console.log("Server is online!");
})
```

- ✓ Rafraichissez la page.