SOFIP
SE FORMER POUR L'AVENIR

Créer un serveur avec EXPRESS

1 - Démarrer un projet Node.js

Avant de démarrer, installer Node (et NPM) depuis le site officiel. Il vous prendre la dernière version stable qui correspond à votre OS.

Pour démarrer, on va créer une toute nouvelle application. Pour cela, la première étape sera de créer un package JSON. Il est utilisé comme une sorte de pièce d'identité qui contient les informations de votre projet comme le nom, la version et également les modules, c'est-à-dire les dépendances qui vont servir à exécuter votre projet.

On va également y spécifier le point d'entrée et les scripts. Les scripts, ce sont les commandes qui vont servir à effectuer des opérations comme exécuter des tests unitaires ou lancer votre application.

La première étape sera de vérifier que vous ayez bien Node installé sur votre machine.

On va le vérifier en ligne de commande et, pour cela, on va ouvrir le terminal intégré du dossier. On va effectuer cette commande qui est node -v.

Là, vous pourrez vérifier que vous l'avez bien installé sur votre machine Node et également vérifier quelle version est installée sur votre machine.

Pour créer mon package JSON, nous allons faire npm init et -y. À l'intérieur, vous allez trouver tous les détails de votre projet, donc le nom qui correspond automatiquement au nom du répertoire, la version, une description que vous pouvez renseigner, le point d'entrée : index.js et les scripts, c'est-à-dire les commandes que l'on va pouvoir exécuter sur notre projet.

Nous allons supprimer le script test et créer le script start, et ça va correspondre à node index.js, c'est-à-dire mon point d'entrée.

```
"scripts": {
  "start": "node index.js"
},
```

Nous allons créer mon point d'entrée, avec **touch index.js** en ligne de commande ou directement avec les outils Vs Code. Ce sera notre point d'entrée pour l' application Node.

On y indique avec un log, un texte : " Hello World ! ". On va lancer notre projet avec npm start. " Hello World ! " va apparaitre en console.

2 - Installation du framework Express

Express JS est un framework principalement utilisé pour construire des applications web, des serveurs avec Node et également des API REST.

On va donc pouvoir utiliser Express pour développer notre application backend et une API REST qui va servir d'interface pour exposer les fonctionnalités de notre application backend.

On va s'en servir également pour créer des routes pour donner accès à des ressources et des données. On va également les utiliser les opérations CRUD pour manipuler les données à partir de notre application client.

On pourra également tester les fonctionnalités de notre API REST avec des utilitaires comme **Curl** et des applications comme **Postman**. Plus tard nous allons sécuriser et contrôler les accès de notre API REST avec l'authentification.

Nous allons ajouter Express avec cette ligne de commande : **npm install express**. Dans notre dossier, va apparaître un nouveau répertoire qui contient les dépendances de notre projet : le **node_modules**.

Le package JSON aura désormais la propriété **dependencies** où l'on va lister les dépendances avec leurs versions qui vont servir à exécuter notre projet.

Nous allons créer un nouveau serveur basique avec Express.

```
1  const express = require("express");
2  const app = express();
3  const port = 3000;
4
5  app.get("/", (req, res) => {
6    res.send("Hello World!");
7  });
8
9  app.listen(port, () => {
10   console.log(`App listening on port ${port}`);
11 });
```

La ligne 1 permet d'ajouter Express à notre module.
La ligne 2 va créer une nouvelle instance d'Express.
La ligne 3 permet de spécifier le port sur lequel va tourner notre application.

De la ligne 5 à la ligne 7, nous aurons route dont le point de terminaison est la racine de notre API REST : on pourra lire comme réponse " Hello World ! ".

Des lignes 9 à 11, on utilise la commande **listen** qui va nous permettre d'écouter une fois notre serveur lancé avec npm.

Nous pouvons lire le message **app listening on port 3000** en console.

Nous allons saisir l'adresse localhost :3000 dans le navigateur et nous allons voir la réponse qui est retournée pour cette première route, le message " Hello World ! ".

Nous verrons comment créer des routes qui vont correspondre aux points de terminaison de notre API REST.

2 – Création des premières routes :

Nous allons créer les points de terminaison qui vont servir à afficher les ressources.

Nous allons ajouter le fichier JSON qui contient quelques données de films (titre et résumé). Cela simule une récupération de données depuis une base de données.

Nous récupérons depuis notre fichier index.js ces données.

Nous allons aussi modifier notre port pour l'écoute de **requêtes http**. Il utilisera la **variable d'environnement** sinon si elle n'est pas définie, grâce à l'**opérateur ||**, le port 5000.

Les variables d'environnements sont plus sûres.

Nous pouvons y déclarer des clé API, des jetons d'authentification ou des informations de connexion.

```
const data = require("./posts");
const PORT = process.env.PORT || 5000;
```

Nous allons créer une nouvelle route qui va afficher les données de notre fichier JSON.

Cette route est de type **GET**, il faudra saisir dans l'url **localhost :5000/movies**. Nous retournons une variable que nous allons nommer **data_posts**.

Pou mieux visionner le résultat sur votre navigateur, vous pouvez utiliser l'extension JSON Viewer.

```
app.get("/movies", (req, res) => {
  const data_posts = [...data];
  res.send(data_posts);
});
```

Pour ne pas devoir relancer le serveur à chaque nouvelle modification utiliser un utilitaire : **nodemon** et qui va permettre de redémarrer automatiquement le serveur dès qu'il y a un changement qui est détecté dans les fichiers sources.

Nous allons l'installer avec la commande npm i nodemon -D. Cette option permet d'installer cette dépendance en mode développeur car nous n'en aurons pas besoin en production.

Tester son API avec cURL :

Nous allons tester une fonctionnalité de notre API : la méthode post qui va servir à rajouter des nouvelles données aux données existantes.

Nous allons utiliser un utilitaire : cURL.

Nous allons modifier notre fichier d'entrée. Nous allons y déclarer un tableau qui contiendra tous les films, existants et ceux que nous allons ajouter.

Nous fusionnons aussi le retour de la méthode GET, pour actualiser à chaque nouvel ajout.

```
let allMovies = [];

app.get("/movies", (req, res) => {
  const data_posts = [...allMovies, ...data];
  res.send(data_posts);
});
```

Nous ajoutons une route de type POST (pour ajouter des données), nous lui précisons l'URL et mergeons le tableau existant et la nouvelle entrée .

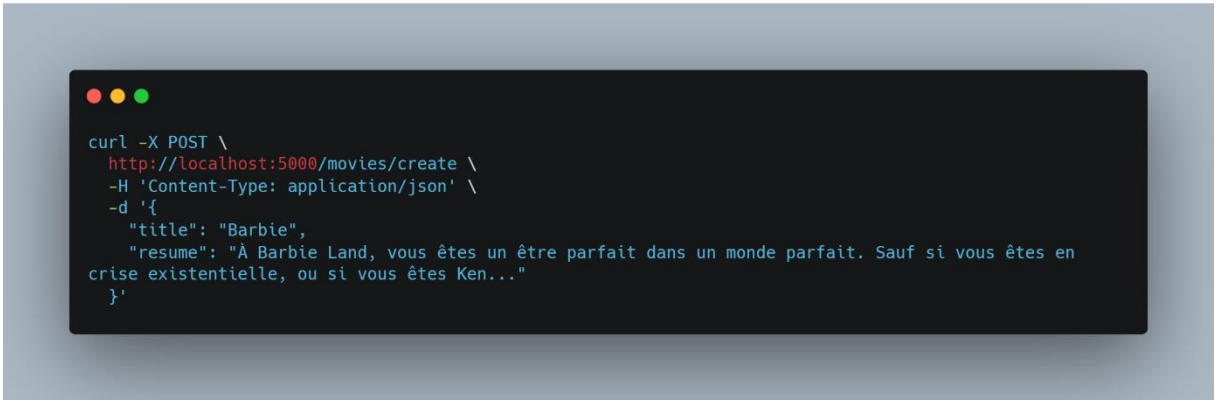
En retour, nous envoyons un message de feedback.

```
app.post("/movies/create", (req, res) => {
  allMovies = [...allMovies, req.body];
  res.send("new movie successfully added");
});
```

Nous allons aussi ajouter un middleware qui va analyser les données JSON reçues dans les requêtes et les transformer en objet JavaScript.

```
app.use(express.json());
```

Ajout de données avec cURL :



Nous allons utiliser le mot clé **curl** puis avec **-X** la méthode à savoir **POST**.

Puis l'url qui contient notre route **/movies/create**.

Ensuite l'option **-H** suivi de **'Content-Type : application/json'** précise que le corps de la requête est du **JSON**. Enfin l'option **-d** suivi d'un objet JSON correspond aux données fournies

Nous allons depuis une autre fenêtre lancer cette commande. Le message **new post successfully added** apparait.