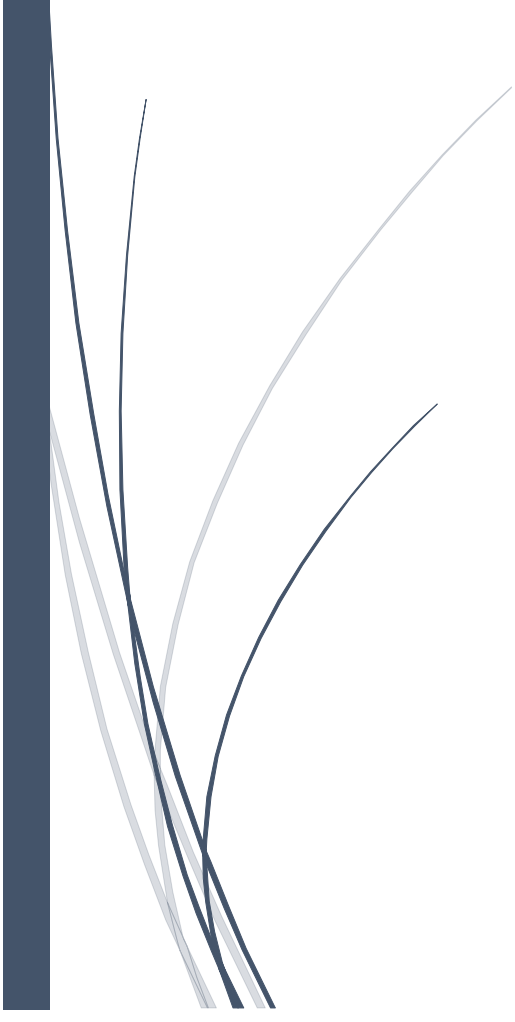
07/01/2025

# Express JS



HP

MICROSOFT

# Table des matières

# I. NodeJS

**Nodejs** est un logiciel permettant d'exécuter du JavaScript côté serveur, contrairement à ce qu'on a l'habitude de voir avec le JavaScript côté client.

# II. NodeJS Express

## 2.1. Définition

Express JS ou tout simplement un framework web open source permettant de construire des applications avec node JS facilement et rapidement.

## 2.2. Les avantages

Avantages de l'utilisation d'Express JS

- Express est libre, et vous pouvez le personnaliser.
- Un seul langage est utilisé pour le développement sur le frontend et le backend.
- Express se lie rapidement à des bases de données comme MySQL, MongoDB, etc.

## 2.3. Installation

Apres l'installation du node.JS, nous pouvons installer le package express.js dans notre système. Pour l'installer globalement, vous pouvez utiliser la commande ci-dessous :

```
npm install -g express
```

Les commandes en ligne suivantes seront exécutés dans le dossier racine du projet que nous voulons crée :

```
1. npm init
```

Cette commande vous posera quelques questions pour générer un fichier package.json dans le racine du dossier de notre projet, ce fichier décrit toutes les dépendances de votre projet et sera mis à jour lors de l'ajout de nouvelles dépendances au cours du processus de développement.

```
2. npm install express
```

Cette commande installe express dans le répertoire de votre projet.

## 2.4. Première application

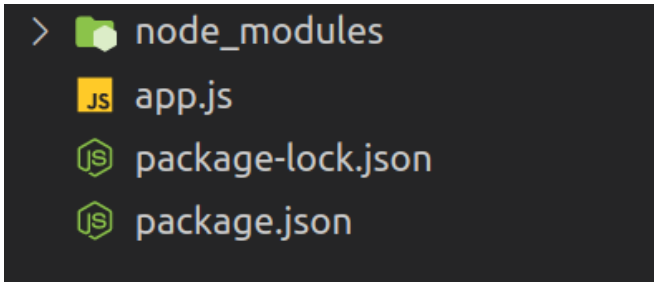
**Etape 1 :** Lancer la commande suivante dans un terminal :

```
npm init
```

**Etape 2 :** Installer Express dans votre projet

```
npm i express
```

**Etape 3 :** Voici la structure du projet que vous devez avoir (Le fichier app.js est ajoute manuellement):




**Etape 4 :** Dans le fichier app.js, Ecrire le programme suivant :

```
import express from "express";

const app = express();
const port = 3000;

app.use(express.json())

app.get('/home', (req, res) => {
    res.send('Hello')
})

app.listen(port,(error) => {
    if(!error)
        console.log(`Ecoute dans le port ${port}`);
    else
        console.log(`Erreur de lanceement`);
} )
```

L'application démarre un serveur et écoute le port 3000 à la recherche de connexions. L'application répond "Hello World!" aux demandes adressées à l'URL racine (/) ou à la ***route*** racine. Pour tous les autres chemins d'accès, elle répondra par **404 Not Found**.

- Une fonction middleware : fonction qui a accès aux objets request et response dans une application requete/reponse.

- La fonction app.use() est utilisé pour charger une fonction middleware au chemin spécifié.

- express.json : La fonction express.json() est une fonction middelware utilisée dans les applications Express.js pour analyser les données JSON des requêtes http.

**Etape 5 :** Démarrer votre projet avec la commande suivante :

```
node app.js
```

## 2.5. Routage des requêtes

### 2.5.1. Presentation

Une route est une section du code Express qui associe une méthode HTTP (GET, POST, PUT, DELETE, etc.), un chemin/un modèle d'URL et une fonction appelée pour traiter ce modèle.

La syntaxe de définition d'une route est la suivante :

**app.METHOD(PATH, HANDLER)**

Avec :

- **app :** est un objet de l'express.
- **METHOD** est une méthode de requête HTTP, en minuscules.
- **PATH** est un chemin sur le serveur.
- **HANDLER** est la fonction exécutée lorsque la route est trouvé.

Voici un exemple d'une route /home définie pour une requête GET.

```
const express = require('express');
const app = express();

// GET method route
app.get('/home', (req, res) => {
  res.send('OK');
});
```

### 2.5.2. Les méthodes route

Une méthode de route est dérivée d'une des méthodes HTTP, et est attachée à une instance de la classe express. Il existe une méthode pour chaque verbe HTTP, les plus couramment utilisées étant les suivantes :

- GET ➔ .get()
- POST ➔ .post()
- PUT ➔ .put()
- DELETE ➔ .delete()

### 2.5.3. La méthode route .all()

Il existe une méthode de routage spéciale app.all(). Elle est utilisée pour charger les fonctions middleware à un chemin pour toutes les requêtes HTTP.

```
app.all('/secret', (req, res, next) => {
    console.log('Accessing the secret section ...');
    next(); // pass control to the next handler
  });
```

### 2.5.4. Chemin de route

Les chemins d'accès avec une méthode de requête définissent les points de terminaison auxquels les requêtes peuvent être effectuées. Il peut s'agir de chaînes de caractères, de modèles de chaînes de caractères ou d'expressions régulières.

**Exemple 1 : Ce chemin d'accès correspondra aux demandes faites à /**

```
app.get('/', (req, res) => {
    res.send('root');
  });
```

**Exemple 2 : Ce chemin d'accès correspondra aux demandes faites à /home**

```
app.get('/home', (req, res) => {
    res.send('root');
  });
```

**Exemple 3 : Ce chemin d'accès correspondra aux demandes faites aux /acd et abcd**

```
app.get('/ab?cd', (req, res) => {
    res.send('ab?cd');
  });
```

**Exemple 4 : Ce chemin d'accès correspondra aux demandes faites aux /abcd, abbcd, etc.**

```
app.get('/ab+cd', (req, res) => {
    res.send('ab+cd');
  });
```

**Exemple 5 : Ce chemin d'accès correspondra aux demandes faites aux /abcd, abxcd, abSASASKcd etc.**

```
app.get('/ab*cd', (req, res) => {
    res.send('ab*cd');
  });
```

### 2.5.5. Les paramètres de route

Une route peut accepter des valeurs dynamiques dans un chemin, appelées paramètres de route. Les paramètres de route sont des segments d'URL nommés qui sont utilisés pour capturer les valeurs spécifiées à leur position dans l'URL. Les valeurs capturées sont intégrées dans l'objet **req.params**, avec le nom du paramètre de route comme clé.

**Exemple :**

```
app.post('/livres/:nom/:prix',(req, res) => {
    console.log(req.params);
    res.end();
});
```

### 2.5.6. Méthodes de réponse

Les méthodes de l'objet de réponse (`res`) décrites dans le tableau suivant peuvent envoyer une réponse au client, et mettre fin au cycle de demande-réponse. Si aucune de ces méthodes n'est appelée par un gestionnaire de routage, la demande du client restera bloquée.

| Méthode | Description |
|---|---|
| res.end() | Met fin au processus de réponse. |
| res.json() | Envoie une réponse JSON. |
| res.redirect() | Redirige une demande. |
| res.send() | Envoie une réponse de divers types. |
| res.sendStatus() | Définit le code de statut de réponse et envoie sa représentation sous forme de chaîne comme corps de réponse. |

### 2.5.7. Exercice Pratique

1. Créez une route GET /produits qui renvoie une liste de produits.
2. Ajoutez une route POST /ajouter qui accepte des données JSON et les affiche dans la console.
3. Testez l'application avec Postman ou via le navigateur.

**Solution :**

```
const produits = ['Livre', 'Stylo', 'Ordinateur'];

app.get('/produits', (req, res) => {
    res.json(produits);
});

app.post('/ajouter', (req, res) => {
    const produit = req.body.nom;
    produits.push(produit);
    res.send(`${produit} ajouté avec succès !`);
});
```

## III. Conclusion

Express.js simplifie le développement d'applications web en utilisant Node.js. En combinant les routes, les middlewares et la gestion des requêtes HTTP, vous pouvez créer des applications web rapides et évolutives.