Cours MERN - Semaine 1

Fondations Approfondies du Back-end : Node.js, Express et l'Architecture Projet

Abdelweheb GUEDDES & Mohamed Ben Jazia / Ecole Polytechnique Sousse

7 septembre 2025

Table des matières

1 Objectifs Pédagogiques Détaillés

À la fin de cette session, vous serez capable de :

- **Théoriser** l'architecture MERN et l'approche "API-First".
- **Concevoir** une architecture de projet back-end scalable (SoC).
- **Expliquer** le modèle d'exécution de Node.js (Event Loop, I/O non-bloquant).
- **Mettre en place** un environnement de développement complet (VS Code, Node.js, Postman).
- **Maîtriser** le cycle de vie d'un projet NPM.
- **Construire** un serveur Express qui gère les requêtes `GET` et `POST`.
- **Comprendre** le rôle du middleware `express.json()` pour parser le corps des requêtes.
- **Valider** des points de terminaison d'API avec Postman.
- **Optimiser** le flux de travail avec 'nodemon'.

2 Partie 1 : Concepts Architecturaux et Techniques (1h15)

2.1 L'Approche "API-First" et l'Architecture MERN

Dans le développement moderne, on ne construit pas une maison en commençant par la peinture. On commence par les fondations. Pour une application web, l'API est cette fondation.

Définition (API-First) : Une stratégie de développement où l'API est traitée comme un produit de première classe. Elle est conçue, documentée et construite avant le développement de toute application cliente qui la consommera.

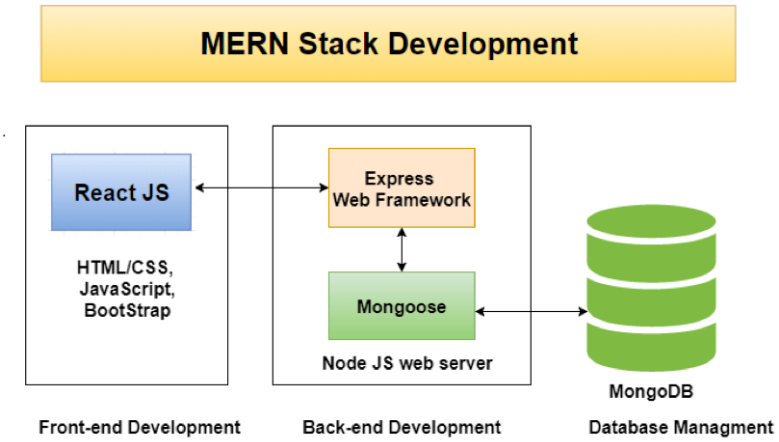

Figure 1 – MERN Stack

2.2 Architecture du Projet : La Séparation des Préoccupations

Un projet qui grandit sans structure devient rapidement un "plat de spaghettis" impossible à maintenir. Le principe de **Séparation des Préoccupations (Separation of Concerns - SoC)** est la clé. Chaque partie de l'application doit avoir une seule responsabilité.

Vision Cible : Même si nous commençons avec un seul fichier 'server.js', nous visons une architecture claire et évolutive. La comprendre dès maintenant donne un sens à chaque nouvelle étape.

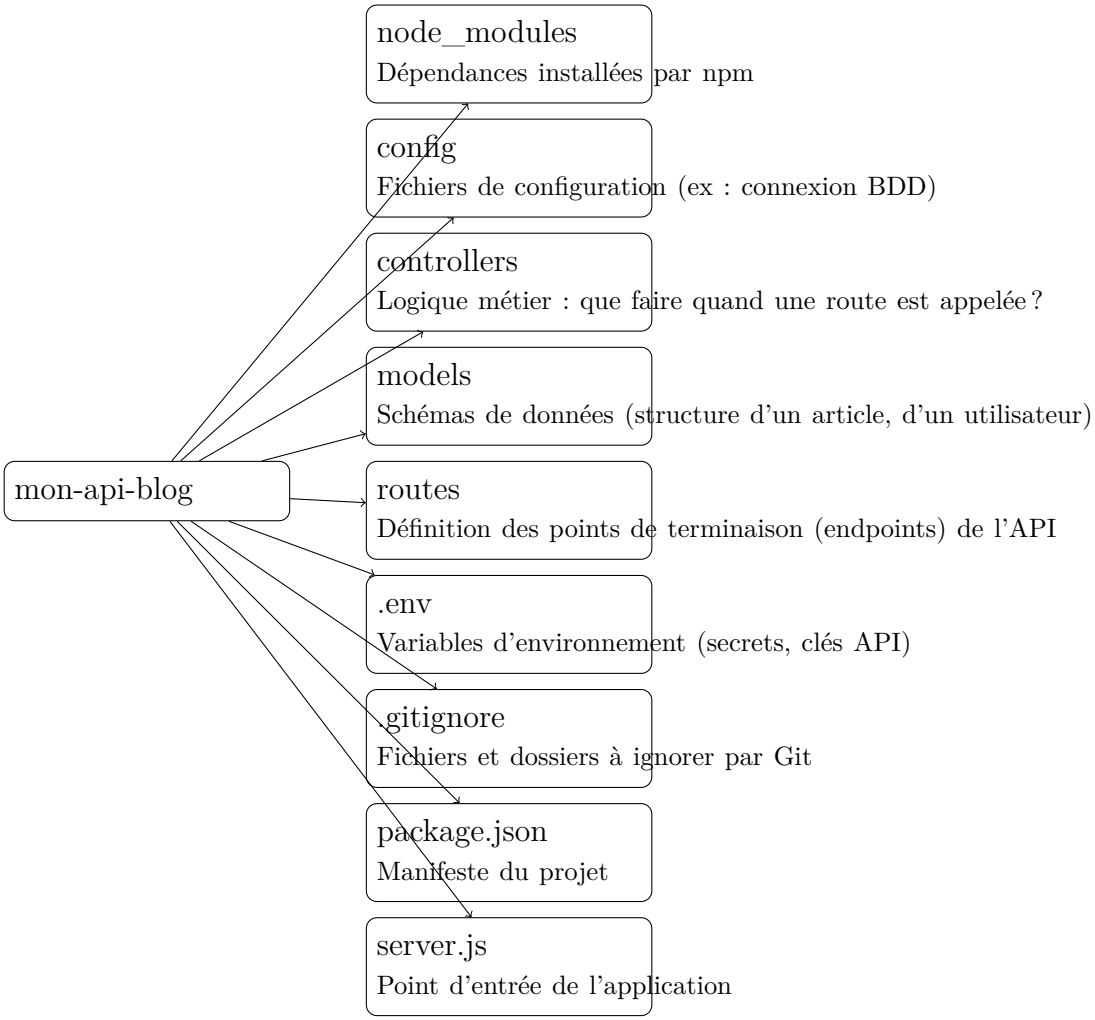

FIGURE 2 – Architecture cible d'un projet Express.js professionnel.

2.3 Plongée dans le Moteur Node.js

Comprendre ce qui rend Node.js si particulier est essentiel. Sa puissance ne vient pas de la vitesse brute d'exécution du code, mais de sa manière de gérer les opérations.

- **Le Moteur V8 :** Le cœur de Node.js est le moteur JavaScript V8 de Google.
- **Single-Threaded (Mono-processus) :** Node.js exécute tout le code dans un seul processus.

— **L'Event Loop et l'I/O non-bloquant :** La plupart des opérations dans une application web sont des opérations d'Entrée/Sortie (I/O).

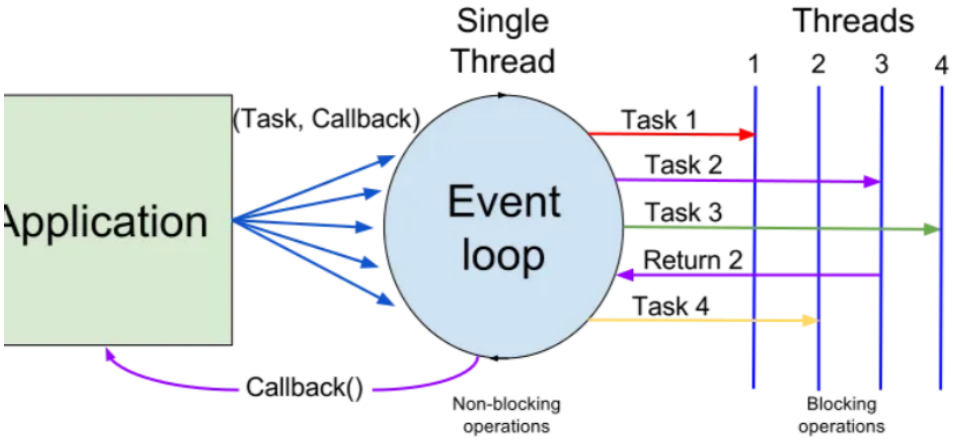

FIGURE 3 – Schéma simplifié de la boucle d'événements (Event Loop) de Node.js.

2.4 Express.js : La syntaxe élégante pour le web

Node.js pur fournit les modules 'http' et 'https' pour créer un serveur, mais leur utilisation est bas niveau et complexe. Express est une surcouche qui abstrait cette complexité et nous offre une API simple et puissante pour gérer le routage et les middlewares. Il transforme un code complexe en quelques lignes lisibles.

3 Partie 2 : Atelier Pratique et Mise en Place (1h45)

3.1 Étape 1 : Configurer un Environnement de Développement Professionnel

1. **Installer Node.js (Version LTS) :** Rendez-vous sur le site officiel de Node.js et téléchargez la version LTS. Vérifiez l'installation dans un terminal :

```
node -v
npm -v

```

2. **Installer et Configurer VS Code :** Installez Visual Studio Code et les extensions suivantes :
   `ESLint` Un "linter" qui analyse votre code en temps réel.
   `Prettier - Code formatter` Un formateur de code pour un style cohérent.
   `DotENV` Ajoute la coloration syntaxique pour les fichiers d'environnement (‘.env‘).
   `GitLens` Outil surpuissant pour l'intégration de Git.
3. **Installer Postman :** Téléchargez Postman depuis son site officiel. C'est un outil standard dans l'industrie pour tester, documenter et partager des API.

3.2 Étape 2 : Démarrage du Projet et Dépendances

1. **Initialisation :** Créez un dossier ‘mon-api-blog‘, ouvrez-le dans VS Code et son terminal intégré :

```
npm init -y

```

Listing 1 – Initialisation du projet Node.js

2. **Installation des dépendances :**

```
# Installer express et le sauvegarder dans "dependencies"
npm install express

# Installer nodemon et le sauvegarder dans "devDependencies"
npm install nodemon --save-dev

```

Listing 2 – Installation des dépendances

3.3 Étape 3 : Création et Amélioration du Serveur Express

1. **Création du fichier** ‘server.js‘ à la racine de votre projet.
2. **Ajout du middleware `express.json()`**. Par défaut, un serveur Express ne sait pas lire le corps (body) d'une requête `POST`. Ce middleware analyse les corps de requête au format JSON et les rend accessibles dans `req.body`. Il doit être placé avant la définition des routes qui en ont besoin.
3. **Code complet du serveur.** Remplissez ‘server.js‘ avec le code suivant :

```
const express = require('express');
const app = express();
const PORT = 3000;
```

```
4
5  // --- Middleware pour parser le JSON ---
6  // Cette ligne DOIT être placée AVANT la définition de vos routes
       POST.
7  app.use(express.json());
8
9  // --- Routes GET ---
10 app.get('/', (req, res) => {
11   res.status(200).send('<h1>Page d_accueil de notre API de Blog
       !</h1>');
12 });
13
14 app.get('/api/test', (req, res) => {
15     res.status(200).json({ message: 'Le test a fonctionné !',
       success: true });
16 });
17
18 // --- Route pour gérer les requêtes POST ---
19 app.post('/api/articles', (req, res) => {
20     // Grâce à express.json(), req.body contient les données
       envoyées par le client
21     const articleData = req.body;
22     console.log('Données reçues :', articleData);
23
24     // On simule la création d'un article en renvoyant les donné
       es reçues
25     // avec un nouvel ID et un statut 201 (Created).
26     res.status(201).json({
27         message: 'Article créé avec succès !',
28         article: { id: Date.now(), ...articleData }
29     });
30 });
31
32 app.listen(PORT, () => {
33   console.log(`Serveur démarré sur http://localhost:${PORT}`);
34 });
35
```

Listing 3 – server.js - Version complète avec gestion POST

4. **Configuration des scripts NPM** dans 'package.json' :

```
1 "scripts": { "start": "node server.js", "dev": "nodemon server.js
      " },
2
```

Listing 4 – package.json - Section scripts

5. **Lancement** en mode développement :

```
1 npm run dev
2
```

3.4 Étape 4 : Test de l'API avec Postman

1. Lancez Postman et ouvrez votre collection "API Blog".
2. **Test des routes GET :** Testez à nouveau les routes `/` et `/api/test` pour vérifier que tout fonctionne toujours.
3. **Test de la route POST :**
   — Créez une nouvelle requête.
   — Changez la méthode de `GET` à `POST`.
   — Entrez l'URL : `http://localhost:3000/api/articles`
   — Allez dans l'onglet **Body** sous l'URL.
   — Sélectionnez l'option **raw**.
   — Dans le menu déroulant à droite, choisissez **JSON**.
   — Collez le JSON suivant dans la zone de texte :

```
{
    "title": "Mon premier article",
    "content": "Ceci est le contenu de mon article.",
    "author": "John Doe"
}

```

Listing 5 – Exemple de corps de requête JSON

   — Cliquez sur "Send".
   — Analysez la réponse : vous devriez recevoir un statut `201 Created` et un corps JSON confirmant la création.
4. Sauvegardez cette nouvelle requête `POST` dans votre collection.

4 Conclusion et Vision pour la Suite

Aujourd'hui, nous avons posé des fondations robustes, mis en place un environnement de développement professionnel et établi une vision architecturale claire.

La semaine prochaine, nous commencerons à implémenter cette architecture en séparant les routes et les contrôleurs.

5 Travail Pratique Complémentaire (À faire par l'étudiant)

Pour renforcer les compétences acquises, étendez le serveur 'server.js'.

1. **Ajoutez une route "À Propos"** (`GET /about`).
2. **Ajoutez une route d'API pour les "utilisateurs"** (`GET /api/users`) qui renvoie un tableau JSON d'utilisateurs factices.
3. **Améliorez la route de "contact" `POST`** :
   — URL : `/contact` | Méthode : `POST`
   — Faites en sorte que cette route accepte un corps JSON contenant un `email` et un `message`.
   — La réponse JSON de succès doit inclure l'email reçu, par exemple : `{ message: "Message reçu de exemple@email.com !" }`.

4. **Testez tout avec Postman**, en particulier la nouvelle route POST avec un corps JSON.

Ce travail pratique est à inclure dans votre compte rendu.

Note Importante : Travail à Rendre

À la fin de chaque session, chaque étudiant doit rédiger et soumettre un compte rendu personnel. Ce document doit décrire en détail le travail que vous avez réalisé, les commandes utilisées, et expliquer avec vos propres mots les concepts que vous avez compris. Échéance : La soumission doit se faire au plus tard la veille de la prochaine séance, à 23h59 précises. Ce compte rendu est obligatoire et noté et fait partie intégrante de votre évaluation continue.