Partie 1: Introduction à Node.js

1. Qu'est-ce que Node.js ?

Explication :
Node.js est un environnement d'exécution pour JavaScript construit sur le moteur V8 de Chrome. Il permet d'exécuter du code JavaScript côté serveur, ce qui était traditionnellement réservé aux langages comme PHP ou Java. Node.js est non bloquant, piloté par les événements, ce qui le rend efficace et adapté pour des applications en temps réel.

Exemple de concept :
Pas d'exemple de code ici, car cette section est plus théorique.

2. Installation de Node.js et Configuration de l'Environnement

Explication :
Pour commencer à utiliser Node.js, vous devez d'abord l'installer sur votre machine. L'installation varie selon le système d'exploitation, mais des installateurs sont disponibles sur le site officiel de Node.js. Après l'installation, vous pouvez utiliser le terminal ou l'invite de commande pour vérifier la version de Node.js et de NPM (Node Package Manager), qui est installé automatiquement avec Node.js.

Exemple de commande :

```
node -v
npm -v
```

Ces commandes affichent les versions de Node.js et de NPM, confirmant leur installation réussie.

3. Premiers pas avec Node.js

Explication :
Une fois Node.js installé, vous pouvez écrire votre premier script. Un script de base peut simplement afficher un message dans la console. Vous pouvez exécuter des scripts Node.js depuis la ligne de commande en utilisant la commande `node` suivie du nom de votre fichier script.

Création d'un simple serveur HTTP :
Voici comment vous pouvez créer un serveur HTTP de base qui écoute sur le port 3000 et répond avec "Bonjour, Node.js !" à chaque requête.

serveur.js :

```
const http = require("http");

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  res.end("Bonjour, Node.js !");
```

```
});

server.listen(3000, () => {
  console.log("Serveur démarré sur http://localhost:3000");
});
```

Pour exécuter le serveur, utilisez la commande :

```
node serveur.js
```

Naviguez ensuite vers `http://localhost:3000` dans votre navigateur pour voir le message du serveur.

Cette première partie pose les bases nécessaires pour comprendre ce qu'est Node.js, comment l'installer et commencer à créer des scripts simples, incluant un serveur web de base. Ces connaissances serviront de fondations pour les sections suivantes du cours, qui exploreront des concepts plus avancés et des fonctionnalités de Node.js.

Pour passer à la syntaxe ESM (ECMAScript Modules) dans Node.js, qui permet d'utiliser `import` et `export` comme dans le JavaScript moderne côté client, vous devrez apporter quelques modifications à votre projet Node.js. Voici les instructions pour effectuer cette transition, suivies de l'adaptation des exemples précédents à la syntaxe ESM.

Passer à la Syntaxe ESM dans Node.js

1. **Modifier le fichier `package.json`** :
   Ajoutez `"type": "module"` dans votre fichier `package.json`. Cela indique à Node.js de traiter les fichiers `.js` comme des modules ECMAScript au lieu de scripts CommonJS.

   ```
   {
     "name": "mon-projet-node",
     "version": "1.0.0",
     "type": "module",
     "scripts": {
       "start": "node serveur.js"
     },
     "dependencies": {}
   }
   ```

2. **Utiliser `import` et `export`** :
   Au lieu de `require()` pour charger des modules et `module.exports` pour exporter, utilisez `import` pour importer des modules et `export` pour exporter des fonctions, des classes ou des variables.

Adaptation des Exemples à la Syntaxe ESM

Reprenons l'exemple de la création d'un serveur HTTP et adaptons-le à la syntaxe ESM.

Création d'un Simple Serveur HTTP avec ESM

serveur.js :

```
import http from "http";

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  res.end("Bonjour, Node.js avec ESM !");
});

server.listen(3000, () => {
  console.log("Serveur démarré sur http://localhost:3000");
});
```

Pour exécuter ce serveur, la commande reste la même :

```
node serveur.js
```

La Syntaxe ESM

Tout au long du reste du cours, nous utiliserons la syntaxe ESM pour tous les exemples de code. Cela implique :

- L'utilisation de `import` pour importer des modules (par exemple, des bibliothèques tierces, des fichiers de configuration, des modèles de données, etc.).
- L'exportation de modules, de fonctions, de classes ou d'objets avec `export` ou `export default`.

La transition vers ESM rendra votre code plus moderne et aligné avec les standards actuels de JavaScript, facilitant l'intégration avec d'autres projets JavaScript et des outils de développement modernes.

Fondamentaux de Node.js

4. Gestion des Modules en Node.js

Explication :
Avec ESM, l'importation et l'exportation de modules deviennent plus intuitives et alignées avec les standards du JavaScript moderne. Apprenez à organiser votre code en modules réutilisables et à les intégrer dans votre projet Node.js.

Exemple d'importation et d'exportation :

utils.js :

```
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
```

app.js :

```
import { add, subtract } from "./utils.js";

console.log(add(4, 3)); // Affiche 7
console.log(subtract(10, 5)); // Affiche 5
```

5. Travailler avec le Système de Fichiers (FS)

Explication :
Node.js permet de manipuler le système de fichiers de manière synchrone ou asynchrone. Apprenez à lire et écrire des fichiers, à travailler avec des flux et à gérer des opérations de fichier complexes.

Exemple de lecture de fichier :

readFile.js :

```
import { readFile } from "fs/promises";

async function readMyFile(filePath) {
  try {
    const data = await readFile(filePath, { encoding: "utf8" });
    console.log(data);
  } catch (error) {
    console.error(`Erreur lors de la lecture du fichier:
${error.message}`);
  }
}

readMyFile("./monFichier.txt");
```

6. Création d'un Serveur Web avec Node.js

Explication :
Node.js est idéal pour créer des serveurs web grâce à sa capacité à gérer des opérations non bloquantes et à sa vitesse. Vous apprendrez à construire un serveur HTTP qui peut servir des pages web et gérer des requêtes réseau.

Exemple de serveur web :
(Voir l'exemple fourni précédemment dans la section sur l'introduction à Node.js et la syntaxe ESM.)

7. Utilisation de NPM (Node Package Manager)

Explication :
NPM est l'écosystème de packages de Node.js. Il facilite le partage et l'utilisation de code entre développeurs du monde entier. Apprenez à trouver, installer et gérer des packages pour votre projet.

Exemple d'installation d'un package :
Pour installer le package Express, utilisez la commande suivante dans le terminal :

```
npm install express
```

Ensuite, vous pouvez l'importer dans votre projet :

```
import express from "express";

const app = express();

app.get("/", (req, res) => {
  res.send("Bonjour avec Express et ESM!");
});

app.listen(3000, () =>
  console.log("Serveur démarré sur http://localhost:3000")
);
```

Le fichier `package.json`

Le fichier `package.json` est au cœur de tout projet Node.js. Il sert de manifeste pour votre application. Ce fichier contient des métadonnées pertinentes concernant le projet, telles que le nom du projet, la version, la description, l'auteur, les licences, etc. Il détaille également les dépendances du projet, c'est-à-dire les packages externes sur lesquels le projet s'appuie pour fonctionner correctement. En plus de cela, `package.json` peut contenir divers scripts utilisés pour exécuter des tâches comme démarrer le serveur, tester l'application, et plus encore. Lorsque vous installez un nouveau package avec NPM dans votre projet, il est automatiquement ajouté à la liste des dépendances dans ce fichier.

Le fichier `package-lock.json`

Le fichier `package-lock.json` est généré automatiquement pour tout projet qui utilise npm pour gérer les paquets (c'est-à-dire dès que vous exécutez `npm install` pour la première fois). Ce fichier verrouille les versions de toutes les dépendances installées, ce qui garantit que vous et toute autre personne travaillant sur le projet utiliserez exactement les mêmes versions des dépendances, même si de nouvelles versions sont publiées. Cela aide à éliminer les incohérences entre les environnements de développement, de test et de production, et facilite le débogage et le test des applications. `package-lock.json` conserve également une structure d'arbre de toutes les dépendances et sous-dépendances de votre projet.

Le répertoire `node_modules`

Le répertoire `node_modules` est où NPM stocke les packages et leurs dépendances. Chaque fois que vous installez un package externe via NPM, il est téléchargé et placé dans ce répertoire. Cela permet à votre application d'importer et d'utiliser ces packages comme spécifié dans vos fichiers de code source. Le répertoire `node_modules` peut devenir assez volumineux, car il contient non seulement les packages que

vous avez explicitement installés mais aussi toutes leurs dépendances. Pour cette raison, il est courant d'exclure ce dossier du contrôle de version en l'ajoutant à `.gitignore`.

En résumé, `package.json` sert de manifeste pour votre projet, définissant les dépendances et les scripts; `package-lock.json` assure la cohérence des versions des packages installés à travers tous les environnements de développement; et `node_modules` contient physiquement les packages installés. Ces éléments travaillent ensemble pour gérer les dépendances et assurer la stabilité de l'environnement de développement de votre projet Node.js.

Développement d'Applications Web avec Node.js

Dans cette partie du cours, nous allons plonger dans le développement d'applications web avec Node.js, en se concentrant sur l'utilisation du framework Express.js, la gestion des routes et des middleware, et l'introduction à MySQL pour la gestion de données. Nous allons également explorer l'upload de fichiers et l'utilisation du moteur de template EJS pour générer des vues dynamiques. Voici le contenu détaillé de cette section, en utilisant la syntaxe ESM pour tous les exemples de code.

8. Framework Express.js

Explication :
Express.js est un framework web rapide, sans opinion et minimaliste pour Node.js. Il facilite la création d'applications web et d'API robustes grâce à une riche collection de fonctionnalités disponibles.

Exemple de base :
Installation d'Express :

```
npm install express
```

app.js :

```
import express from "express";

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

app.get("/", (req, res) => {
  res.send("Bonjour, monde avec Express!");
});

app.listen(port, () => {
  console.log(`Serveur démarré sur http://localhost:${port}`);
});
```

9. Gestion des Routes et des Middleware avec Express

Explication :
Les routes permettent de définir des réponses à différentes requêtes HTTP. Les middlewares sont des

fonctions qui ont accès à l'objet de requête (req), l'objet de réponse (res), et la fonction middleware suivante dans le cycle de requête-réponse de l'application.

Exemple de route et middleware :
app.js (suite) :

```
app.use(express.json()); // Middleware pour parser le JSON

app.post("/message", (req, res) => {
  console.log(req.body);
  res.send(`Message reçu : ${req.body.message}`);
});
```

Les middlewares dans Express, exemples

Les middleware sont des fonctions qui ont accès aux objets de requête (request), de réponse (response), et à la fonction de middleware suivante dans le cycle de requête-réponse de l'application. Ils sont utilisés pour exécuter du code, effectuer des changements aux requêtes et aux réponses, terminer le cycle de requête-réponse, ou appeler le prochain middleware dans la pile.

Les middleware peuvent effectuer diverses tâches telles que le parsing des données de requête, la gestion des sessions, la vérification de l'authentification des utilisateurs, le logging, etc. Ils sont fondamentaux dans le développement d'applications avec Express car ils permettent une grande modularité et réutilisabilité du code.

Logging de Chaque Requête

Un middleware simple pour logger des détails de chaque requête (méthode HTTP et URL) :

```
import express from "express";

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

const loggerMiddleware = (req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next(); // Passe au prochain middleware ou route handler
};

app.use(loggerMiddleware);

app.get("/", (req, res) => {
  res.send("Page d'accueil");
});

app.listen(port, () =>
  console.log(`Serveur lancé sur http://localhost:${port}`)
);
```

Vérification de l'Authentification

Middleware pour vérifier si un utilisateur est authentifié avant de lui permettre d'accéder à une route protégée :

```
const authMiddleware = (req, res, next) => {
  // Supposons que req.isAuthenticated est une fonction vérifiant
l'authentification
  if (req.isAuthenticated()) {
    next();
  } else {
    res.status(403).send("Non autorisé");
  }
};

// Utilisation du middleware sur une route spécifique
app.get("/profil", authMiddleware, (req, res) => {
  res.send("Profil de l'utilisateur");
});
```

Parsing du Corps de la Requête

Express.js lui-même utilise le concept de middleware pour le parsing du corps des requêtes. Pour les requêtes POST ou PUT, par exemple, où vous avez besoin d'accéder aux données envoyées (comme un formulaire), vous pouvez utiliser `express.json()` et `express.urlencoded()` qui sont des middleware intégrés pour parser le corps de la requête.

```
app.use(express.json()); // Pour parser les corps des requêtes en JSON
app.use(express.urlencoded({ extended: true })); // Pour parser les corps
des requêtes venant des formulaires

app.post("/login", (req, res) => {
  console.log(req.body.username); // Accès au nom d'utilisateur envoyé
dans le corps de la requête
  res.send("Login réussi");
});
```

Gestion des Fichiers Statiques

Express fournit un middleware intégré, `express.static`, pour servir des fichiers statiques tels que des images, des fichiers CSS, et des fichiers JavaScript.

```
app.use(express.static("public")); // Servira les fichiers dans le
répertoire "public"
```

Dans cet exemple, si vous avez un fichier `public/images/logo.png`, il sera accessible via `http://localhost:3000/images/logo.png`.

10. Introduction à MySQL et Intégration dans Node.js

Explication :
MySQL est un système de gestion de base de données relationnelle. L'intégration de MySQL avec Node.js permet de stocker et de récupérer des données pour vos applications web.

Exemple de connexion à MySQL :
Premièrement, installez le package mysql :

```
npm install mysql2
```

database.js :

```
import mysql from "mysql2/promise";

async function connect() {
  const connection = await mysql.createConnection({
    host: "localhost",
    user: "mon_user",
    database: "ma_base_de_donnees",
    password: "mon_mot_de_passe",
  });
  console.log("Connecté à la base de données MySQL!");
  return connection;
}

export default connect;
```

11. Gestion de l'Upload de Fichiers

Explication :
L'upload de fichiers dans une application web permet aux utilisateurs de télécharger des fichiers sur le serveur.

Exemple avec `multer` :
Installation de Multer :

```
npm install multer
```

app.js (suite) :

```
import multer from "multer";
const upload = multer({ dest: "uploads/" });

app.post("/upload", upload.single("fichier"), (req, res) => {
  console.log(req.file);
  res.send("Fichier téléchargé avec succès");
});
```

12. Templating avec EJS

Explication :
EJS est un moteur de template simple qui vous permet de générer du HTML dynamique côté serveur avec JavaScript.

Exemple d'utilisation de EJS :
Installation de EJS :

```
npm install ejs
```

app.js (suite) :

```
app.set("view engine", "ejs");

app.get("/profil", (req, res) => {
  res.render("profil", { nom: "John Doe" });
});
```

profil.ejs :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Profil</title>
  </head>
  <body>
    <h1>Bonjour, <%= nom %>!</h1>
  </body>
</html>
```

Cette partie du cours fournit une fondation solide pour développer des applications web complètes en utilisant Node.js. En maîtrisant Express.js, la gestion des routes et des middleware, l'intégration avec MySQL, l'upload de fichiers, et le templating avec EJS, les apprenants seront équipés pour construire des applications web dynamiques et interactives.

Le pattern MVC (Modèle-Vue-Contrôleur)

L'architecture Modèle-Vue-Contrôleur (MVC) est un schéma de conception logiciel qui sépare une application en trois composants principaux, chacun ayant des responsabilités spécifiques : le Modèle (gère les données et la logique métier), la Vue (présente les données à l'utilisateur dans un format spécifique), et le Contrôleur (interprète les entrées de l'utilisateur, en commandant le modèle pour effectuer des actions ou des mises à jour, et en actualisant la vue en conséquence). Réorganiser une application Express pour suivre le modèle MVC peut améliorer considérablement sa maintenabilité, sa scalabilité et sa clarté. Voici comment vous pouvez structurer une application Express avec EJS en utilisant le modèle MVC :

1. Structure de Dossier

Commencez par organiser votre application en dossiers séparés pour les modèles, les vues, et les contrôleurs.

```
mon-application/
|-- controllers/
|   |-- userController.js
|-- models/
|   |-- userModel.js
|-- views/
|   |-- user/
|       |-- index.ejs
|       |-- profile.ejs
|-- routes/
|   |-- userRoutes.js
|-- app.js
```

- **models/** : Contient les définitions des modèles de données (interaction avec la base de données).
- **views/** : Contient les fichiers EJS pour la présentation (l'interface utilisateur).
- **controllers/** : Contient la logique de traitement des requêtes, la manipulation des modèles, et la sélection des vues à afficher.
- **routes/** : Définit les itinéraires et associe les requêtes entrantes aux contrôleurs appropriés.

2. Modèles

Les modèles représentent la structure des données. Ils interagissent généralement avec la base de données pour créer, lire, mettre à jour, et supprimer des informations.

userModel.js :

```
class User {
  constructor(id, name, email) {
    this.id = id;
    this.name = name;
    this.email = email;
  }
```

```
  // Méthodes pour interagir avec la base de données
}

export default User;
```

3. Vues

Les vues sont des fichiers EJS qui définissent comment les données doivent être présentées à l'utilisateur. EJS permet d'insérer des données dynamiques dans du HTML.

views/user/profile.ejs :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Profil de l'utilisateur</title>
  </head>
  <body>
    <h1>Bienvenue, <%= user.name %>!</h1>
  </body>
</html>
```

4. Contrôleurs

Les contrôleurs prennent les entrées de l'utilisateur, manipulent les modèles et sélectionnent les vues à afficher.

userController.js :

```
import User from "../models/userModel.js";

export const userProfile = (req, res) => {
  const user = new User(1, "John Doe", "john@example.com");
  res.render("user/profile", { user });
};
```

5. Routes

Les routes associent les chemins d'URL à leurs contrôleurs spécifiques.

userRoutes.js :

```
import express from "express";
import * as userController from "../controllers/userController.js";

const router = express.Router();
```

```
router.get("/profile", userController.userProfile);

export default router;
```

6. Intégration dans l'Application Principale

Intégrez le tout dans votre fichier principal `app.js` pour lancer l'application.

app.js :

```
import express from "express";
import userRoutes from "./routes/userRoutes.js";

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

app.set("view engine", "ejs");
app.use("/user", userRoutes);

app.listen(port, () =>
  console.log(`Serveur lancé sur http://localhost:${port}`)
);
```

Cette structure MVC permet de séparer clairement les concepts dans votre application, rendant le code plus propre, plus facile à maintenir et à étendre.

Fonctionnalités Avancées

13. **Authentification et Sécurité**
    - Stratégies d'authentification.
    - Sécurisation des applications Node.js.
14. **Test et Débogage d'Applications Node.js**
    - Introduction aux tests unitaires avec Mocha et Chai.
    - Débogage d'applications Node.js.

Projet Final

16. **Création d'une Application e-Commerce**
    - Récapitulatif des concepts appris.
    - Guide étape par étape pour construire une application e-commerce avec Node.js, Express, MySQL, et EJS.
    - Gestion des produits, des utilisateurs, des paniers d'achat, et des commandes.
    - Implémentation de l'upload de photos de produits et de l'authentification des utilisateurs.