TD/TP Node.js + Express.js
JS/TS en Back-End

M. Mohammed BELATAR

m.belatar@emsi.ma

avec Mme. FZ MOUTAI et M. YF EBOBISSE

3ème Année Ingénierie en Informatique et Réseaux

2 décembre 2024

Sommaire

Exercice 1 : Création d'un Serveur Simple avec Express

- Initialisez un projet TypeScript avec Node.js.
- Installez les dépendances nécessaires :

Commande

```
npm install express @types/express ts-node
```

- Configurez un fichier `tsconfig.json`.
- Créez un serveur Express.js simple avec une route `/` qui renvoie :

Exemple

```
"Bienvenue sur votre serveur Node.js avec TypeScript !"
```

Extension : Ajoutez une route `/status` qui retourne un objet JSON avec l'état du serveur.

Exercice 2 : API REST avec Express.js

- Implémentez une API REST pour gérer des utilisateurs :
  - **GET /users** : Retourne une liste statique d'utilisateurs.
  - **POST /users** : Ajoute un utilisateur (stockez les utilisateurs en mémoire).
  - **GET /users/ :id** : Retourne un utilisateur spécifique.
- Utilisez TypeScript pour définir des types ou des interfaces.

Extension : Ajoutez des validations pour vérifier que les données POST incluent un `name` et un `email`.

Exercice 3 : Middleware avec Express.js

- Implémentez un middleware pour logger toutes les requêtes entrantes avec leur méthode HTTP et leur URL.
- Ajoutez un middleware pour valider que chaque requête POST vers `/users` inclut un champ `name`.
- Testez ces middlewares avec différentes requêtes.

Extension : Implémentez un middleware qui vérifie la présence d'un header `X-Auth-Token` et rejette la requête avec un code 403 si ce n'est pas le cas.

Exercice 4 : Connexion à une Base de Données avec MySQL2

- Configurez une base MySQL locale avec une table `users` :
  - `id` (INT, AUTO_INCREMENT, PRIMARY KEY)
  - `name` (VARCHAR(50))
  - `email` (VARCHAR(100))
- Connectez-vous à cette base de données avec `mysql2`.
- Créez une route **GET /users** pour lister tous les utilisateurs.

Extension : Ajoutez une route **POST /users** pour insérer un utilisateur dans la base.

Exercice 5 : Gestion des Erreurs

- Implémentez une gestion d'erreurs centralisée dans l'API :
  - Retournez un code 400 pour une requête invalide.
  - Retournez un code 404 pour une route inexistante.
- Ajoutez un middleware global pour gérer toutes les erreurs.

Extension : Ajoutez des logs pour les erreurs dans un fichier externe.

Exercice 6 : CRUD Complet avec MySQL2

- Implémentez les routes suivantes pour gérer les utilisateurs :
  - **GET /users** : Liste tous les utilisateurs.
  - **POST /users** : Ajoute un utilisateur.
  - **GET /users/ :id** : Retourne un utilisateur spécifique.
  - **PUT /users/ :id** : Modifie un utilisateur existant.
  - **DELETE /users/ :id** : Supprime un utilisateur.
- Validez les données avant de les insérer ou les modifier.

Extension : Implémentez des validations pour vérifier que les champs `name` et `email` sont valides.

Exercice 7 : Authentification Simple

- Implémentez une route **POST /login** :
  - Accepte un `email` et un `password` fictif.
  - Génère un token si les identifiants sont corrects.
- Ajoutez un middleware pour vérifier que les requêtes vers `/users` incluent un token valide.

Extension : Stockez les utilisateurs dans une base de données et vérifiez les identifiants à partir de cette table.

Résumé des Exercices

- **Objectifs couverts** :
  - Utilisation de TypeScript en Back-End.
  - Création d'APIs REST avec Express.js.
  - Connexion à une base MySQL avec `mysql2`.
  - Gestion des erreurs et implémentation de middlewares.