Rapport TP

Express.JS

CHRIT Adam
Filière : Génie Informatique
Année universitaire : 2024/2025

I. Introduction

I.I Qu’est-ce qu’Express.js ?

Express.js est un framework minimaliste et flexible pour construire des applications web et des APIs en utilisant Node.js. Il fournit une couche d'abstraction au-dessus de Node.js, facilitant ainsi le développement d'applications web. Il permet aux développeurs de créer des applications avec une structure claire et concise, de gérer les routes, les requêtes HTTP, les réponses, et bien plus encore.

Express.js est souvent utilisé pour créer des applications back-end pour gérer les interactions entre un client (comme une application front-end ou un navigateur) et le serveur. Il permet aussi d'écrire des APIs REST, qui sont des interfaces permettant aux applications d'échanger des données sur le web.

I.II Ce que nous pouvons faire avec Express.js

Avec Express.js, on peut :

- Créer des applications web (comme des blogs, des systèmes de gestion de contenu, etc.).
- Développer des APIs RESTful pour communiquer avec des bases de données et d'autres systèmes.
- Gérer des requêtes HTTP et des réponses (par exemple, des requêtes GET, POST, PUT, DELETE).
- Utiliser des middlewares pour traiter des requêtes ou des réponses avant qu'elles ne soient envoyées.
- Gérer les sessions d'utilisateurs, les cookies, et la sécurité (via des mécanismes d'authentification et de validation).

II. Middlewares

Les middlewares dans Express.js sont des fonctions qui exécutent un traitement sur les objets request (requête) et response (réponse), ou qui terminent le cycle de la requête-réponse. Ils peuvent aussi passer le contrôle à la fonction middleware suivante dans la chaîne.

Un middleware peut effectuer diverses tâches comme :

- La validation des requêtes (vérifier si les données envoyées par le client sont correctes).
- La gestion des erreurs (capturer les erreurs survenues pendant le traitement d'une requête).
- La gestion des sessions (identifier et suivre les utilisateurs via des cookies ou des tokens).

II.I Exemple 1 : Middleware de journalisation

Un middleware de journalisation enregistre chaque requête effectuée sur le serveur dans la console. Cela permet de suivre l'activité des utilisateurs et de déboguer plus facilement.

```
app.use((req, res, next) => {
    console.log(`Requête reçue : ${req.method} ${req.url}`);
    next();
});
```

II.II Exemple 2 : Middleware pour traiter données JSON

Ce middleware utilise express.json() pour convertir les données JSON envoyées par le client dans un objet JavaScript, afin qu'elles puissent être facilement manipulées sur le serveur.

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

III. Application CRUD

III.I Importation et configuration d'Express.js

```
const express = require('express'); // Importation d'Express.js
const app = express(); // Création d'application avec objet app
const PORT = 3000; // Port d'accès
```

III.II Middleware pour traiter données JSON

```
app.use(express.json()); // Transformation corps JSON en objets JS
```

III.III Endpoint POST pour ajouter

```
app.post('/items', (req, res) => {
    const newItem = {
        id: items.length + 1,
        ...req.body
    };
    items.push(newItem);
    res.status(201).json(newItem);
});
```

- **app.post() :** Gestion des requêtes HTTP POST envoyées à /items.
- **req.body** : Contient les données envoyées par client dans le corps de la requête.
- **items.push() :** Ajout du nouvel objet au tableau items.
- **res.status(201).json(newItem) :** Envoi de réponse avec le code HTTP 201 (Créé) et retourne l'objet nouvellement ajouté au format JSON.

III.IV Endpoint GET pour récupérer tous les items

```
app.get('/items', (req, res) => {
    res.json(items); // Envoi du tableau items au format JSON
});
```

III.V Endpoint GET pour récupérer un item par ID

```
app.get('/items/:id', (req, res) => {
    const id = parseInt(req.params.id);
    const item = items.find(item => item.id === id);
    if (item) {
        res.json(item);
    } else {
        res.status(404).json({ message: "Item non trouvé" });
    }
});
```

- **req.params.id :** Récupère l'ID de l'item à partir de l'URL.
- **items.find() :** Recherche d'item correspondant à l'ID donné dans le tableau items.
- Si l'item est trouvé, il est renvoyé au format JSON ; sinon, une réponse avec message est renvoyée.

III.VI Endpoint PUT pour mettre à jour un item

```
app.put('/items/:id', (req, res) => {
    const id = parseInt(req.params.id);
    const index = items.findIndex(item => item.id === id);
    if (index !== -1) {
        items[index] = { ...items[index], ...req.body, id };
        res.json(items[index]);
    } else {
        res.status(404).json({ message: "Item non trouvé" });
    }
});
```

- **app.put() :** Gestion des requêtes HTTP PUT envoyées à /items.
- **items.findIndex() :** Recherche l'index de l'item à mettre à jour dans le tableau items.

III.VII Endpoint DELETE pour supprimer un item

```
app.delete('/items/:id', (req, res) => {
    const id = parseInt(req.params.id);
    const index = items.findIndex(item => item.id === id);
    if (index !== -1) {
        const deletedItem = items.splice(index, 1);
        res.json(deletedItem[0]);
    } else {
        res.status(404).json({ message: "Item non trouvé" });
    }
});
```

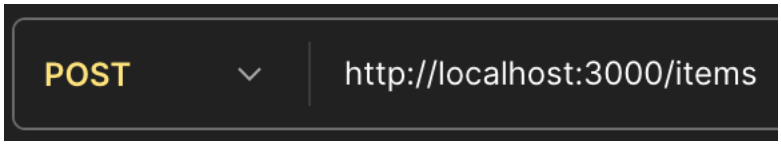
- **app.delete() :** Gestion des requêtes HTTP DELETE envoyées à /items.
- **items.splice() :** Supprime l'item du tableau items et retourne l'item supprimé au client.

III.VIII Lancement du serveur

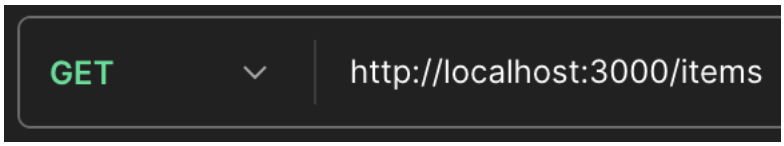
```
app.listen(PORT, () => { // Démarre le serveur sur port défini
    console.log(`Serveur démarré sur http://localhost:${PORT}`);
});
```

IV. Tests sur POSTMAN

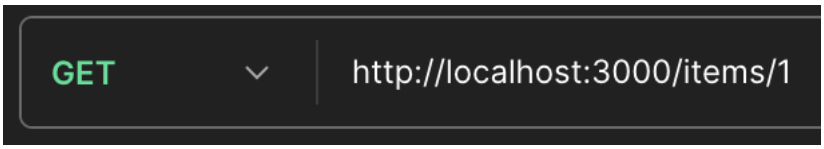
IV.I POST



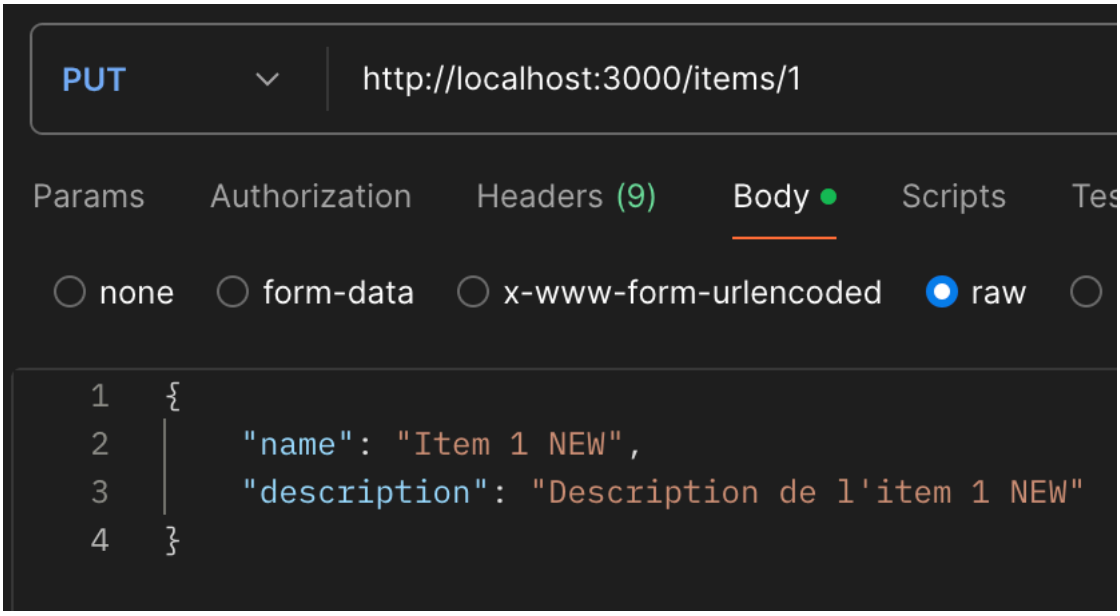
IV.II GET



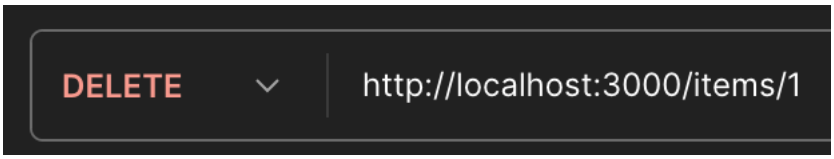
IV.III GET by ID

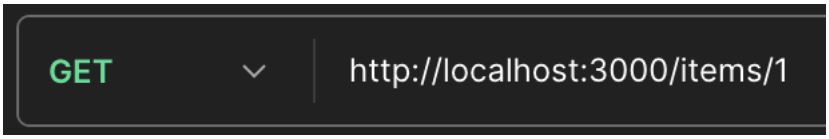


IV.IV PUT



IV.V DELETE





V. Conclusion

Express.js est un outil puissant et flexible pour le développement d'applications back-end. Ce rapport a présenté les fonctionnalités de base d'Express.js, l'utilisation des middlewares, des snippets des code de notre application CRUD, et des tests POSTMAN.