ENSA
ECOLE NATIONALE DES SCIENCES APPLIQUÉES

KHOURIBGA

Université Sultan Moulay Slimane

Filière: Génie Informatique

Présenté par:
IMAD EDDINE ASSOULI

Encadré par:
Pr. OURDOU AMAL

Rapport TP : Express.Js

Année Universitaire: 2024/2025

Introduction

Ce TP a pour but de développer une application CRUD (Create, Read, Update, Delete) en utilisant le framework Express.js dans un environnement Node.js. Express.js est un framework minimaliste pour Node.js, utilisé pour créer des applications web et des API facilement et rapidement.

Express.js et les Middlewares

1 Qu'est-ce qu'Express.js ?

Express.js est un framework pour Node.js qui simplifie la création d'applications web et de serveurs HTTP. Il permet de définir des routes pour répondre aux requêtes HTTP, de gérer les middlewares et d'interagir avec des bases de données ou d'autres services web.

Express.js est largement utilisé pour construire des API REST performantes, des sites web dynamiques et des services backend évolutifs. Il est également couramment utilisé pour les applications temps réel, en combinaison avec des technologies comme WebSockets

2 Les Middlewares

Un middleware dans Express.js est une fonction qui intercepte les requêtes HTTP avant qu'elles n'atteignent les routes définies, ce qui permet de les modifier ou d'effectuer certaines actions (ex : journalisation, validation, gestion d'erreurs).

Exemple de Middleware :

1. Logger Middleware : Ce middleware enregistre chaque requête reçue par le serveur.

```
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next(); // Passe à la route suivante
});
```

2. Middleware de gestion d'erreurs : Ce middleware gère les erreurs générées dans l'application.

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Erreur interne du serveur');
});
```

3 Création de l'application CRUD

3.1 Initialisation du projet

3.1.1 Création du répertoire : Dans le terminal, j'ai créé un répertoire pour le projet :

```
mkdir express-crud
cd express-crud
```

3.1.2 Initialisation du projet Node.js : J'ai ensuite initialisé un projet Node.js avec la commande suivante :

```
npm init -y
```

3.1.3 Installation d'Express.js : Pour installer Express, j'ai utilisé la commande :

```
npm install express
```

3.2 Mise en place d'Express.js et démarrage du serveur

J'ai créé un fichier `app.js` avec le code suivant pour démarrer un serveur sur le port 3000 :

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

app.use(express.json());

// Démarrage du serveur
app.listen(3000, () => {
  console.log('Le serveur est démarré sur le port 3000');
});
```

Le serveur a été démarré avec la commande suivante dans le terminal :

```
node app.js
```

3.3 Implémentation des routes CRUD

L'application dispose de 5 routes : **POST**, **GET**, **GET par ID**, **PUT**, et **DELETE**.

Route POST - Ajouter un élément Cette route permet d'ajouter un nouvel élément dans une liste stockée en mémoire :

```
let items = [];

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

Route GET - Récupérer tous les éléments Cette route retourne tous les éléments de la liste :

```
app.get('/items', (req, res) => {
  res.status(200).send(items);
});
```

Route GET par ID - Récupérer un élément par son ID Cette route retourne un élément spécifique à partir de son identifiant (ID) :

```
app.get('/items/:id', (req, res) => {
  const item = items.find(i => i.id === parseInt(req.params.id));
  if (!item) return res.status(404).send('Item not found');
  res.send(item);
});
```

Route PUT - Mettre à jour un élément Cette route permet de mettre à jour un élément en utilisant son ID :

```
app.put('/items/:id', (req, res) => {
  const item = items.find(i => i.id === parseInt(req.params.id));
  if (!item) return res.status(404).send('Item not found');

  item.name = req.body.name;
  res.send(item);
});
```

Route DELETE - Supprimer un élément Cette route supprime un élément en utilisant son ID :

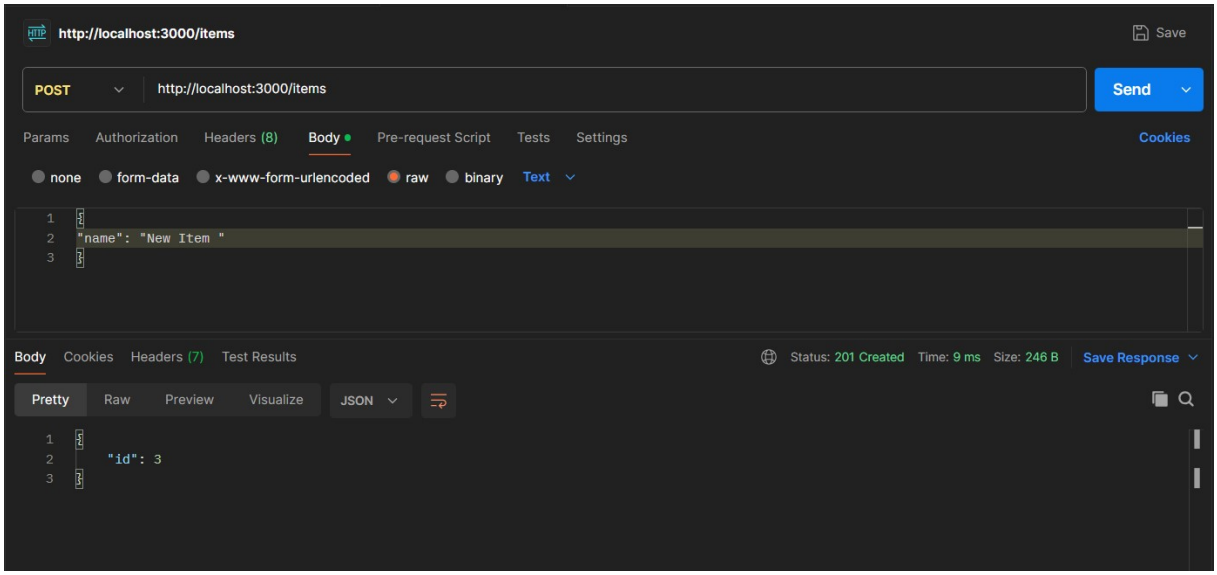
```
app.delete('/items/:id', (req, res) => {
  const itemIndex = items.findIndex(i => i.id === parseInt(req.params.id));
  if (itemIndex === -1) return res.status(404).send('Item not found');

  const deletedItem = items.splice(itemIndex, 1);
  res.send(deletedItem);
});
```

4 Test des routes avec Postman

J'ai testé chaque route à l'aide de Postman. Voici les détails des requêtes et des réponses.

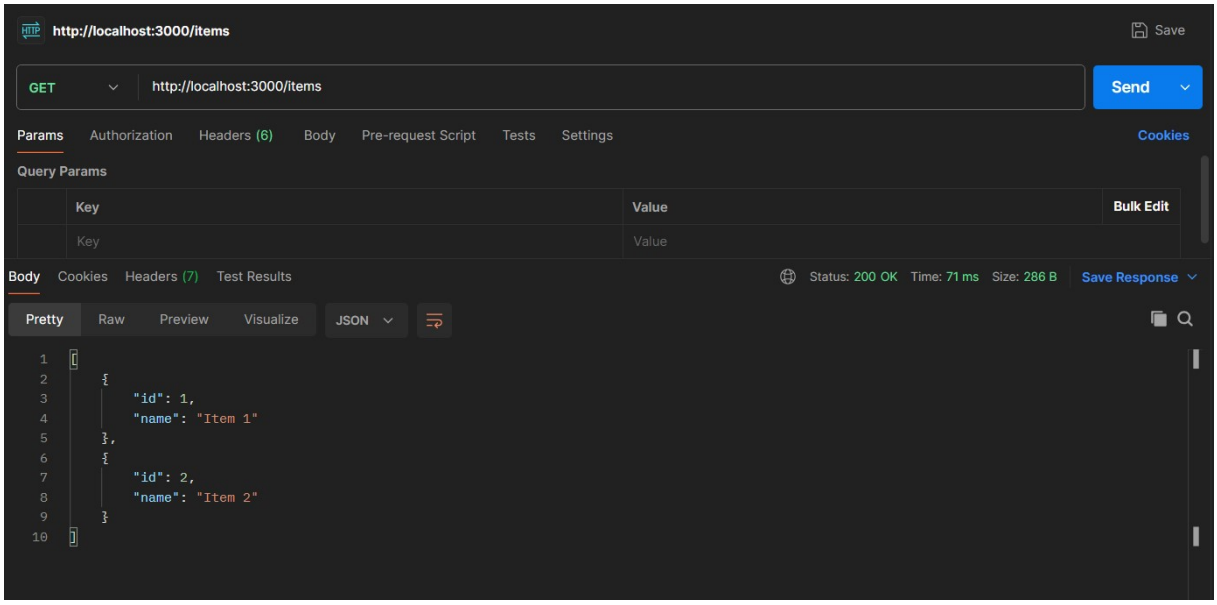
1. Route POST (Ajouter un élément)

Méthode : POST
URL : `http://localhost:3000/items`



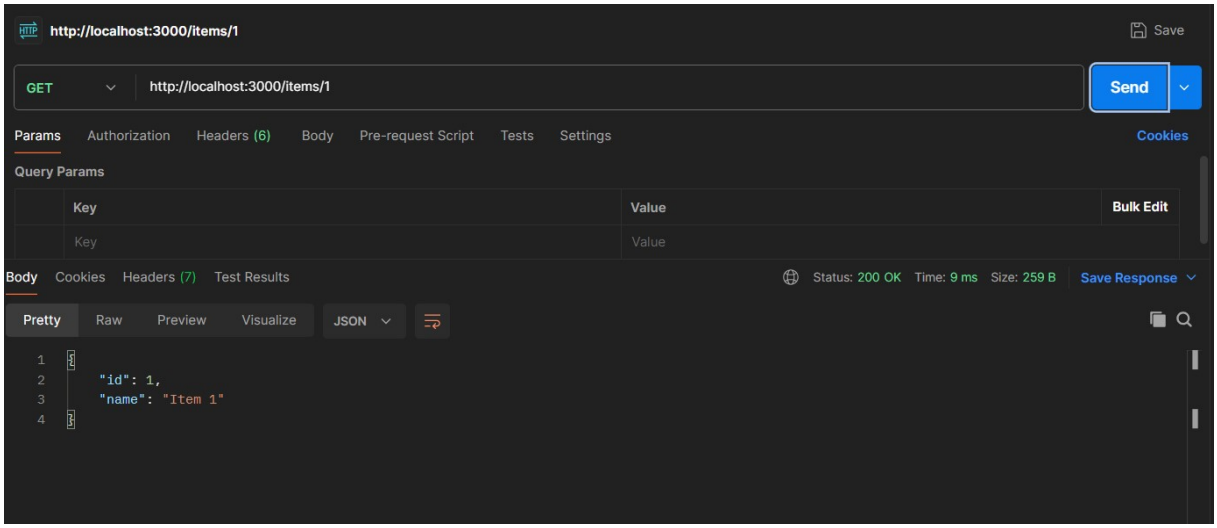
2. Route GET (Récupérer tous les éléments)

Méthode : GET
URL : `http://localhost:3000/items`



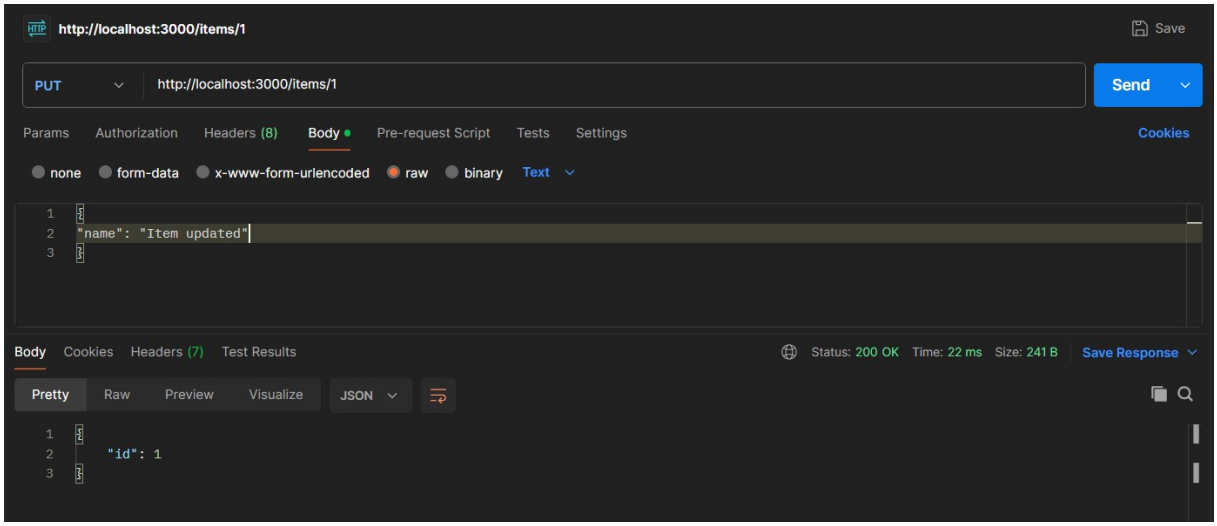
3. Route GET par ID (Récupérer un élément par son ID)

Méthode : GET
URL : `http://localhost:3000/items/1`



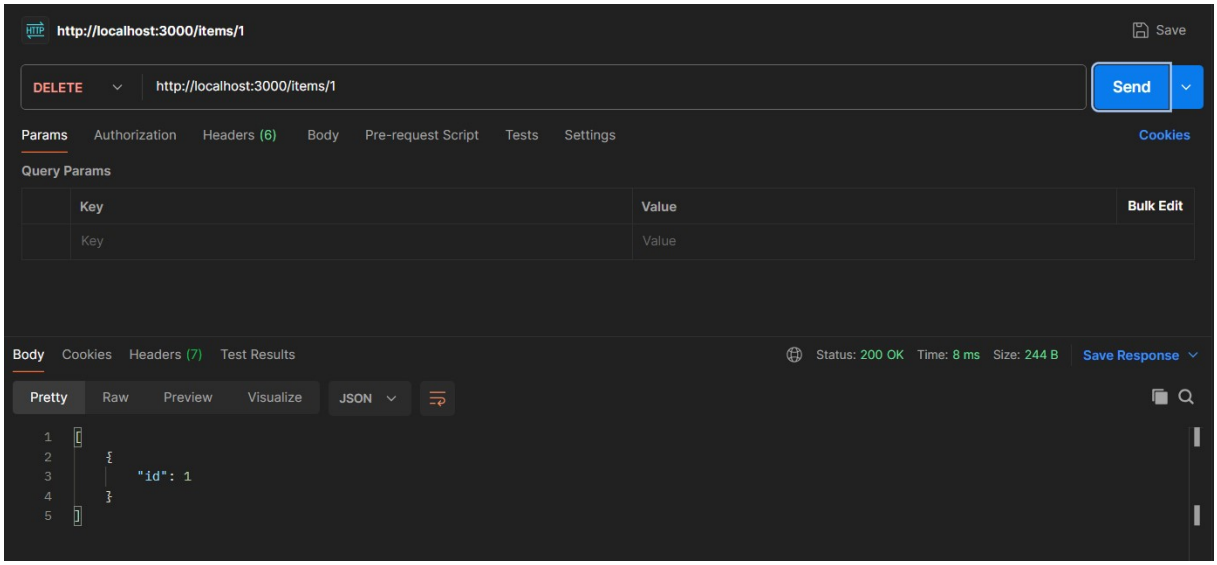
4. Route PUT (Mettre à jour un élément)

Méthode : PUT
URL : `http://localhost:3000/items/1`



5. Route DELETE (Supprimer un élément)

Méthode : DELETE
URL : `http://localhost:3000/items/1`


http://localhost:3000/items/1
Save
DELETE
http://localhost:3000/items/1
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Bulk Edit
Key
Value
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 8 ms
Size: 244 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 [
2 {
3 "id": 1
4 }
5]

Conclusion

Ce TP m'a permis de comprendre les concepts fondamentaux d'Express.js, notamment la gestion des routes et des middlewares. J'ai également appris à mettre en œuvre une application CRUD basique, tout en testant les différentes requêtes avec Postman.