Rapport du TP

- **Réalisé par :** Naji Aya /Sabbahi Kaoutar.
- **Filière :** Génie informatique 3.
- **Encadré par :** Pr. Ourdou.
- **Année Universitaire :** 2024 /2025.

Table de matière

Introduction

Dans ce travail pratique, nous avons mis en œuvre une application CRUD (Create, Read, Update, Delete) simple en utilisant **Express.js**, un framework minimaliste pour Node.js. L'objectif de ce TP était de comprendre les bases de la création de serveurs backend, le traitement des requêtes HTTP, et l'utilisation des middlewares dans Express.js. En suivant une série d'étapes, nous avons appris à configurer un projet Node.js, créer des routes pour gérer différentes opérations sur des données, et à tester ces fonctionnalités via **Postman**.

1. Qu'est-ce que Express.js et que peut-on faire avec ?

Express.js est un framework web léger pour **Node.js** qui facilite la création de serveurs HTTP et d'API RESTful. Il fournit une série d'outils simples pour gérer les routes, les requêtes et réponses HTTP, ainsi que des middlewares. Express.js est utilisé pour construire des :

- Applications web dynamiques.
- API RESTful.
- Applications Single Page (SPA).
- Serveurs backend pour gérer des requêtes client-serveur, communiquer avec des bases de données et d'autres services.

2. Qu'est-ce qu'un middleware et comment est-il utilisé dans Express.js ?

Un **middleware** dans **Express.js** est une fonction qui peut accéder à la requête (req), la réponse (res), et à la fonction next() qui passe au middleware suivant dans la chaîne. Les middlewares permettent d'intervenir dans le traitement des requêtes/réponses, par exemple pour la journalisation, l'authentification, ou encore la gestion d'erreurs.

Exemple 1 : Middleware de journalisation

Ce middleware enregistre toutes les requêtes effectuées vers le serveur :

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

Exemple 2 : Middleware d'authentification

Ce middleware vérifie si un utilisateur est authentifié avant d'autoriser l'accès aux routes protégées :

```
app.use((req, res, next) => {
  if (!req.headers.authorization) {
    return res.status(403).send('Non autorisé');
  }
  next();
});
```

Réalisation du TP

1. Créer un répertoire de projet

Tout d'abord, nous avons créé un répertoire pour le projet et initialisé un projet Node.js avec la commande :

```
PS C:\Users\kawta\Desktop\ProjetCrud> npm init -y
Wrote to C:\Users\kawta\Desktop\ProjetCrud\package.json:

{
  "name": "projetcrud",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

2. Installation d'Express.js

```
PS C:\Users\kawta\Desktop\ProjetCrud> npm install express

added 65 packages, and audited 66 packages in 8s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\kawta\Desktop\ProjetCrud>
```

3. Configuration du serveur

Nous avons configuré le serveur dans un fichier server.js. Le serveur écoute les requêtes sur le port 3000 :

Server.js

```
const express = require('express');
const app = express();
app.use(express.json());  // Permet de traiter le JSON

const port = 3000;
app.listen(port, () => {
  console.log(`Serveur en écoute sur le port ${port}`);
});
```

4. Création des routes CRUD

- Route POST : Ajouter un élément dans un tableau en mémoire.

```
let items = [];

app.post('/items', (req, res) => {
  const item = req.body;
  items.push(item);
  res.status(201).send('Item ajouté avec succès');
});
```

- Route GET (tous les items) : Récupérer tous les éléments.

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

- Route GET (par ID) : Récupérer un élément spécifique par son ID.

```
app.get('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const item = items.find(i => i.id === id);

  if (item) {
    res.status(200).json(item);
  } else {
    res.status(404).send('Item non trouvé');
  }
});
```

- **Route PUT** : Mettre à jour un élément existant.

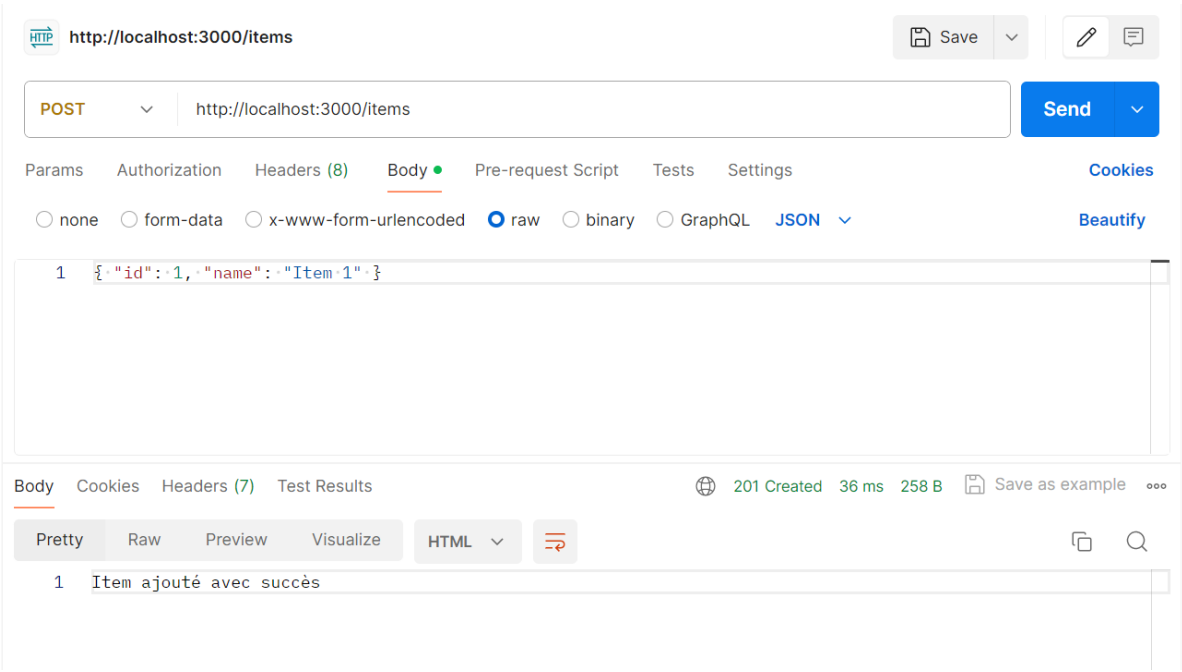
```
35  app.put('/items/:id', (req, res) => {
36    const id = parseInt(req.params.id);
37    const itemIndex = items.findIndex(i => i.id === id);
38
39    if (itemIndex !== -1) {
40      items[itemIndex] = { ...items[itemIndex], ...req.body };
41      res.status(200).send('Item mis à jour');
42    } else {
43      res.status(404).send('Item non trouvé');
44    }
45  });
```

- **Route DELETE** : Supprimer un élément.

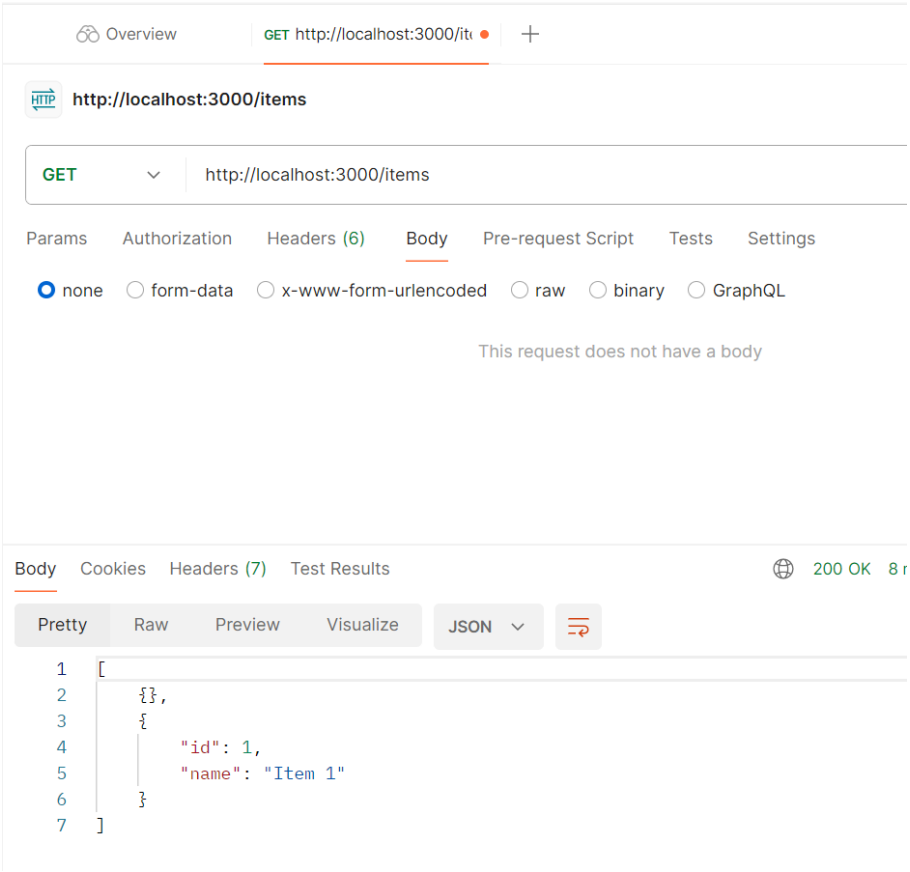
```
47  app.delete('/items/:id', (req, res) => {
48    const id = parseInt(req.params.id);
49    items = items.filter(i => i.id !== id);
50    res.status(200).send('Item supprimé');
51  });
```

5. Test des routes avec Postman

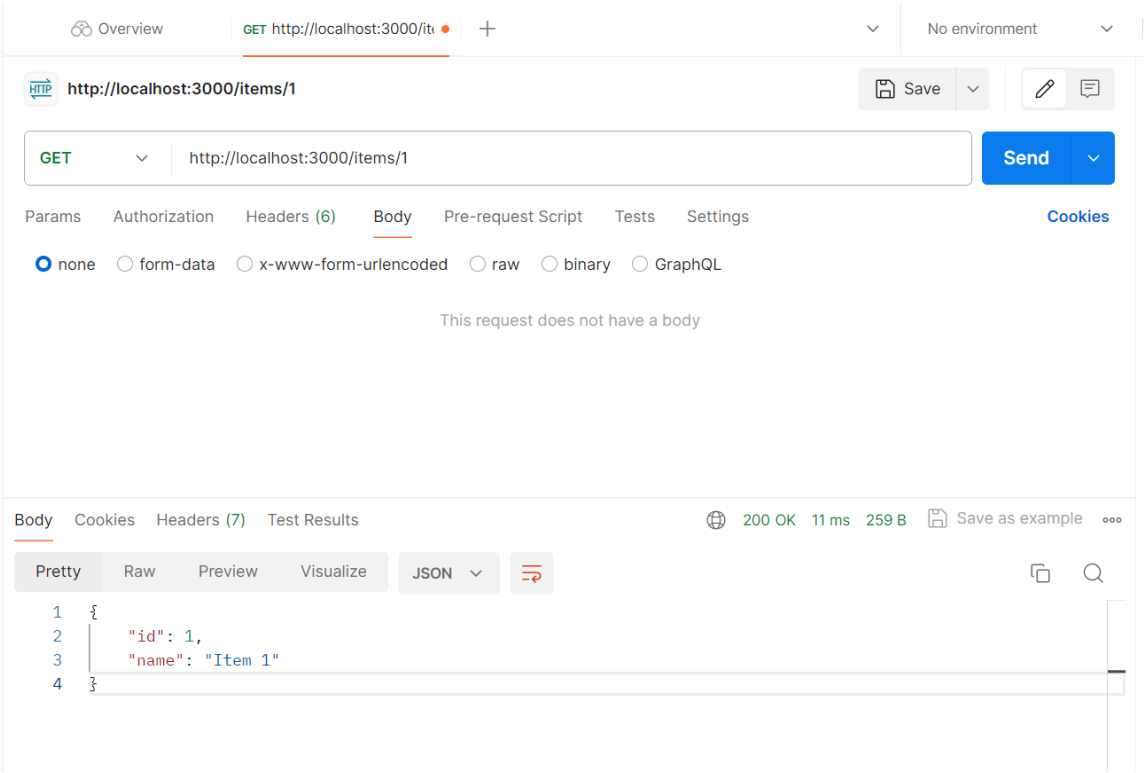
- Route POST



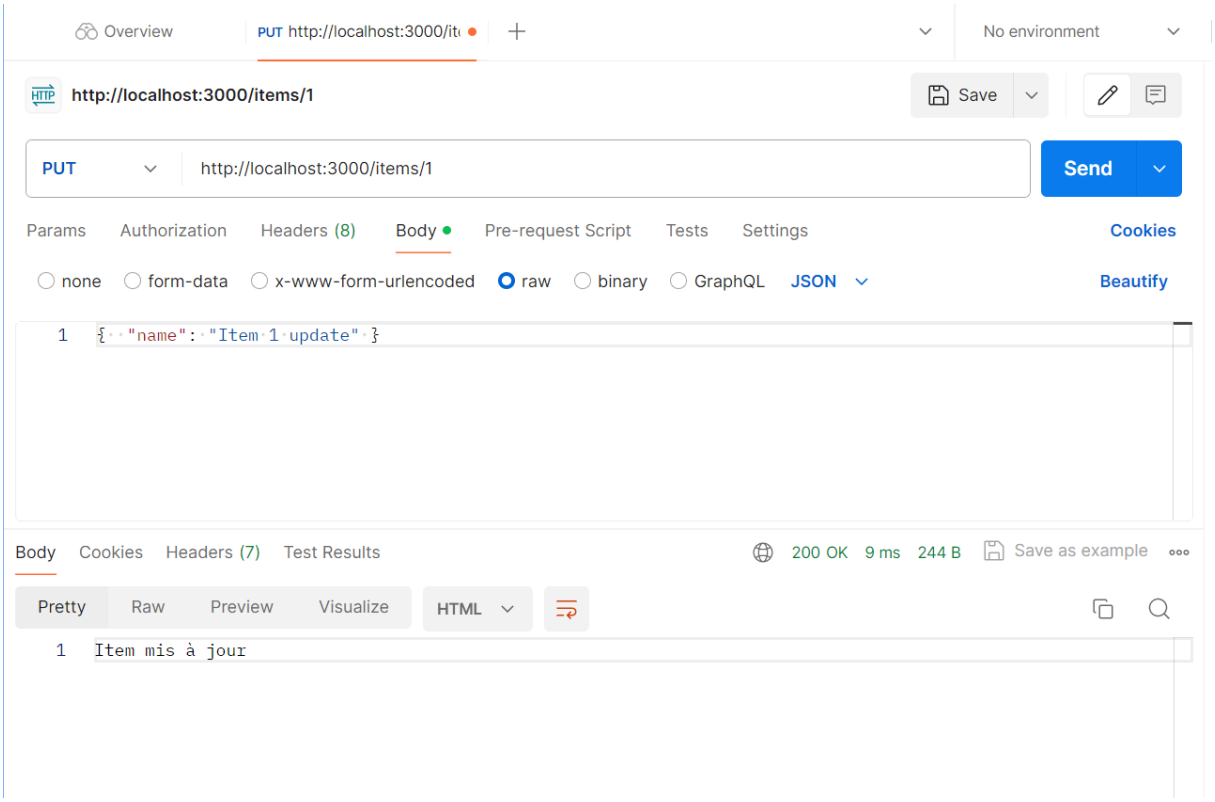
- Route GET



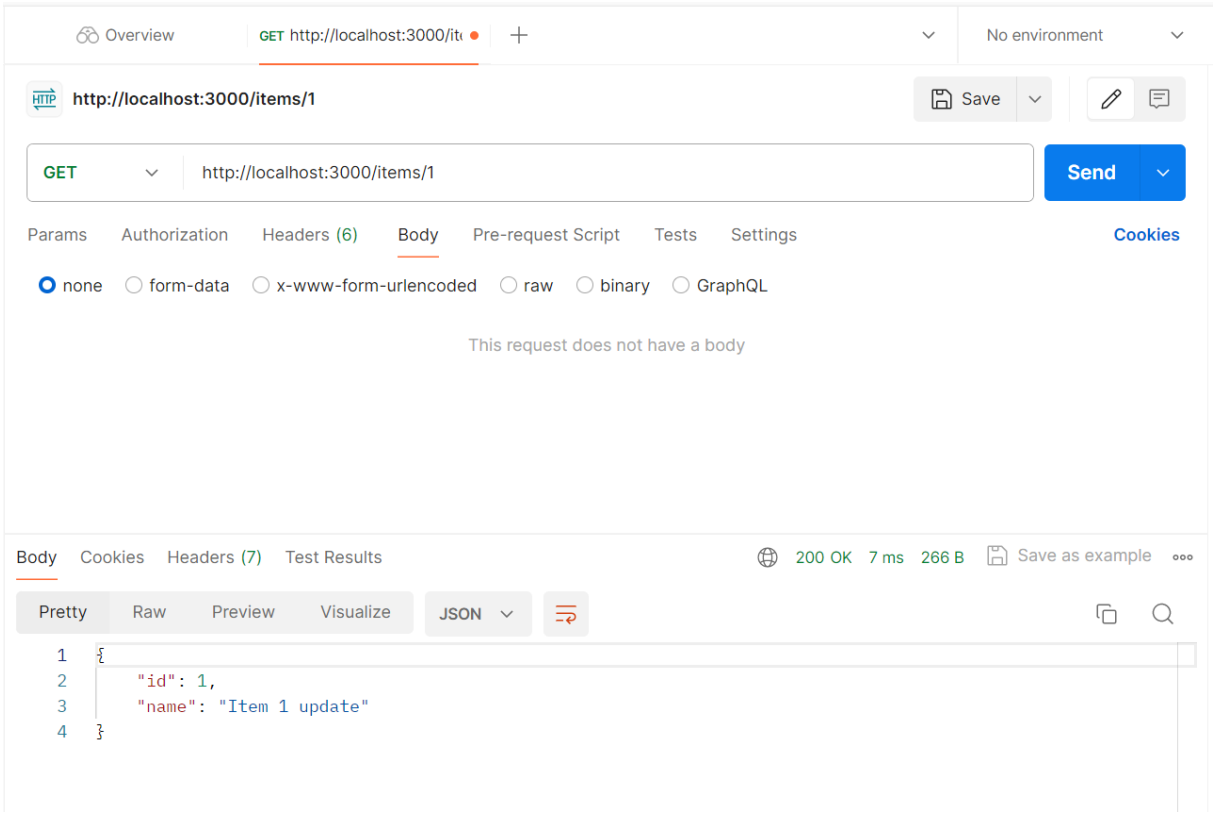
- Route GET (par ID)



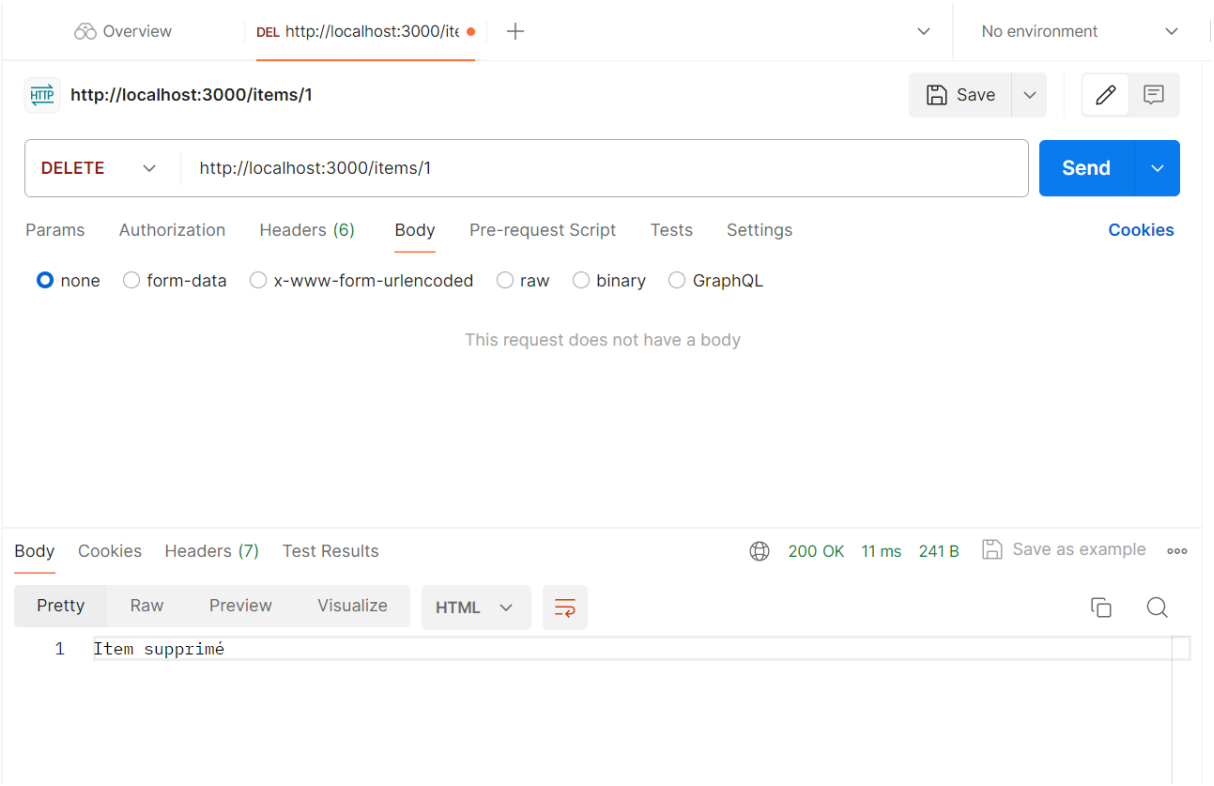
- Route PUT



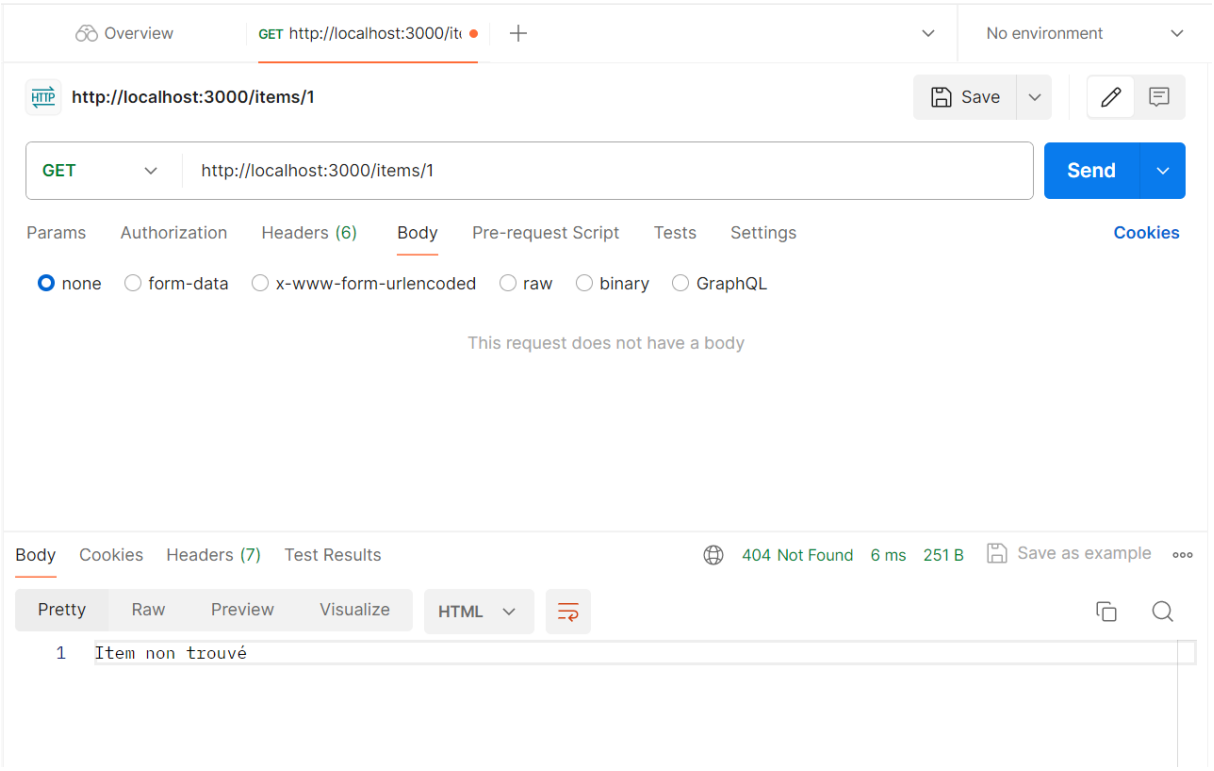
Résultat après mise à jour :



- Route DELETE



Après la suppression :



Conclusion

Ce TP nous a permis de mieux comprendre le fonctionnement d'une application CRUD en utilisant Express.js. Nous avons appris à configurer un serveur web, à créer des routes pour manipuler des données en mémoire, et à utiliser des middlewares pour faciliter le traitement des requêtes. L'utilisation de Postman a facilité la validation des fonctionnalités implémentées. Cette base peut être étendue en ajoutant une base de données ou d'autres fonctionnalités pour des projets plus complexes.