UNIVERSITÉ SULTAN MOULAY SLIMANE

École Nationale des Sciences Appliquées de Khouribga

Département Mathématiques et Informatique

Filière : Génie informatique

Rapport TP : Express.JS

Hamza KERBOUB

Sous la direction de : **Pr OURDOU Amal**

ENSAK 2024/2025

Table des matières

Table des figures

Chapitre 1

Introduction

Dans ce rapport, nous allons expliquer comment créer une application CRUD simple avec Express.js. Nous passerons en revue chaque étape, de la création du projet à la mise en place des endpoints pour chaque opération CRUD (Create, Read, Update, Delete). Des extraits de code seront présentés à chaque étape pour illustrer les concepts.

Chapitre 2

Express.js : Qu'est-ce que c'est ?

2.1 Définition

Express.js est un framework minimaliste et flexible pour Node.js, conçu pour faciliter le développement d'applications web et d'API. Il simplifie la gestion des requêtes HTTP, des réponses et des middlewares.

2.2 Que peut-on faire avec Express.js ?

Avec Express.js, nous pouvons :

- Créer des applications web et des API.
- Gérer les routes pour différents endpoints.
- Utiliser des middlewares pour le traitement des requêtes.
- Intégrer des bases de données pour stocker et récupérer des données.

Chapitre 3

Les middlewares dans Express.js

3.1 Définition des middlewares

Un middleware est une fonction qui a accès à l'objet de requête, l'objet de réponse et à la fonction suivante dans le cycle de requête-réponse. Il peut exécuter du code, modifier les objets de requête/réponse, terminer la requête ou appeler le middleware suivant.

3.2 Exemples d'utilisation

- **Middleware pour analyser les corps de requête JSON :**

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

  Ce middleware permet à Express de traiter les requêtes avec des corps de type JSON.

- **Middleware pour journaliser les requêtes :**

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

  Ce middleware journalise chaque requête avec sa méthode et son URL.

Chapitre 4

Création de l'application CRUD

4.1 Étape 1 : Créer un répertoire de projet

Dans cette étape, nous créons le répertoire de projet et initialisons un projet Node.js :

```
mkdir express-tp
cd express-tp
npm init -y
```

4.2 Étape 2 : Installer Express

Nous installons le framework Express via npm :

```
npm install express
```

4.3 Étape 3 : Configurer le serveur Express

Nous configurons un serveur Express de base qui écoute sur le port 3000.

```
const express = require('express');
const app = express();
const port = 3000;

app.use(express.json());

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

FIGURE 4.1 – Serveur en cours d'exécution dans le terminal

4.4 Étape 4 : Création d'un endpoint POST

Cet endpoint permet d'ajouter un élément à une liste locale :

```
let items = [];

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

4.5 Étape 5 : Création d'un endpoint GET

Cet endpoint permet de récupérer tous les éléments :

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

4.6 Étape 6 : Création d'un endpoint GET par ID

Cet endpoint permet de 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.json(item);
  } else {
    res.status(404).send('Item not found');
  }
});
```

4.7 Étape 7 : Création d'un endpoint PUT

Cet endpoint permet de mettre à jour un élément existant :

```
app.put('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const index = items.findIndex(i => i.id === id);
  if (index !== -1) {
    items[index] = { ...items[index], ...req.body };
    res.send('Item updated');
  } else {
    res.status(404).send('Item not found');
```

```
  }
});
```

4.8 Étape 8 : Création d'un endpoint DELETE

Cet endpoint permet de supprimer un élément par son ID :

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

4.9 Étape 9 : Tester avec Postman

Nous avons testé chaque endpoint à l'aide de Postman pour vérifier le bon fonctionnement de notre application CRUD. Ci-dessous, vous trouverez les résultats des tests pour chaque type de requête.

4.9.1 1. Test du POST Endpoint

Ce test permet d'ajouter un nouvel élément à la liste des items via une requête POST.

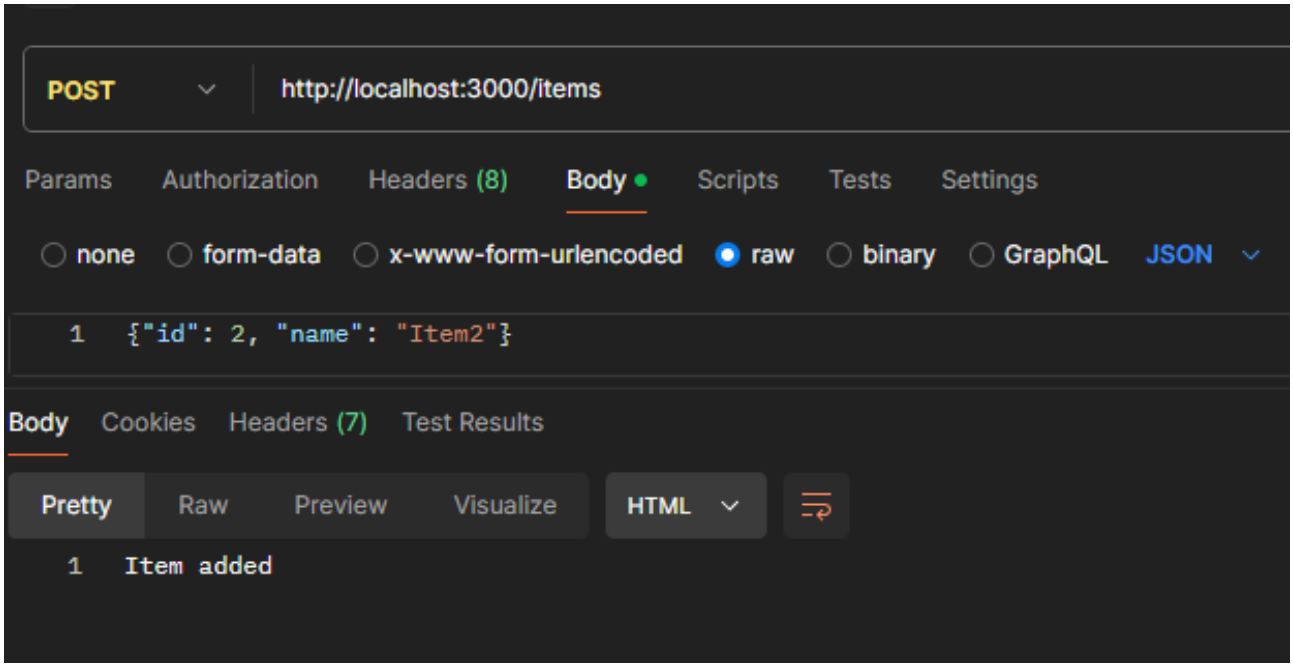

FIGURE 4.2 – Ajout d'un élément via une requête POST dans Postman

4.9.2 2. Test du GET (tous les éléments)

Ce test permet de récupérer tous les éléments via une requête GET.

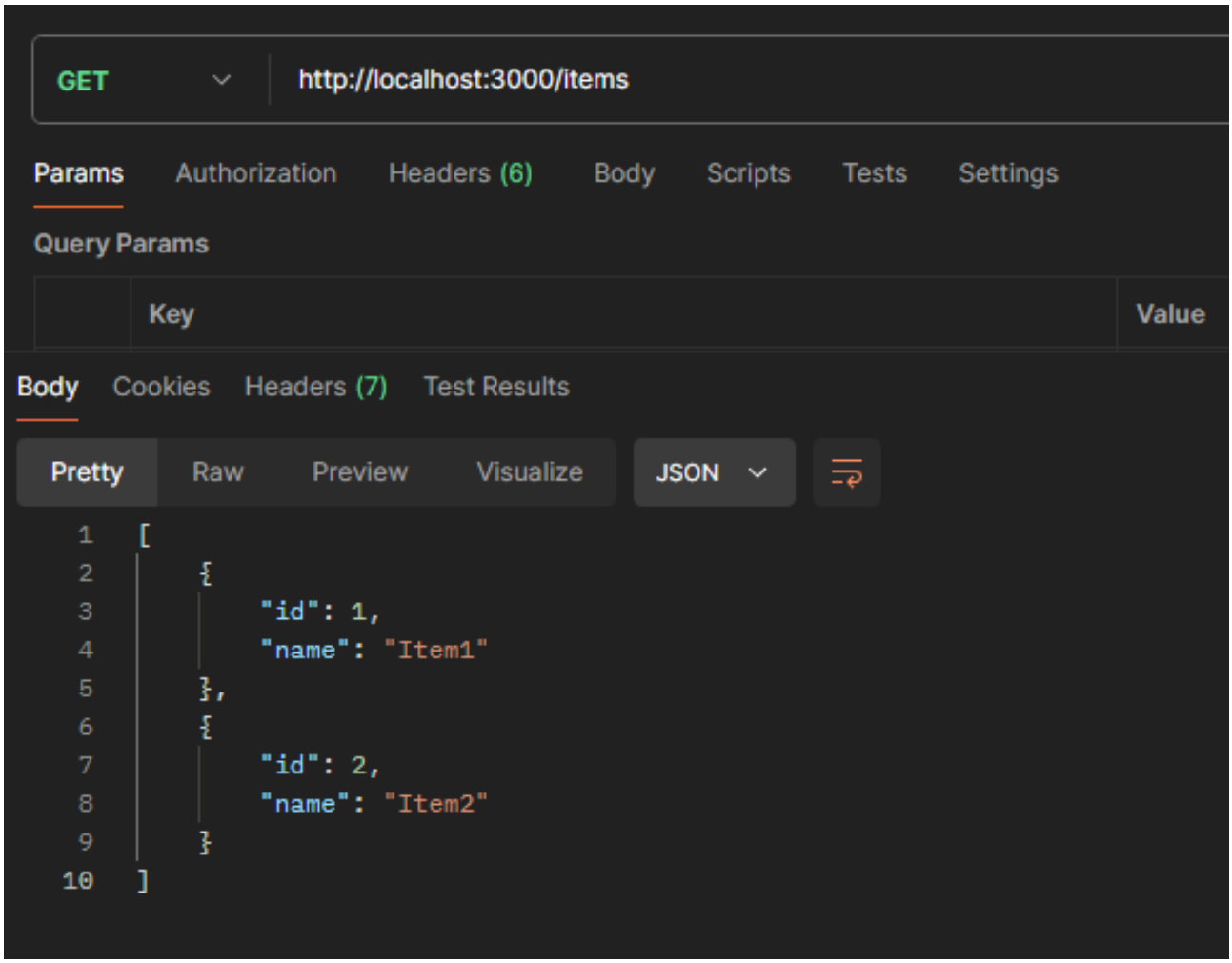

FIGURE 4.3 – Récupération de tous les éléments via une requête GET dans Postman

4.9.3 3. Test du GET (par ID)

Ce test permet de récupérer un élément spécifique en utilisant son ID via une requête GET.

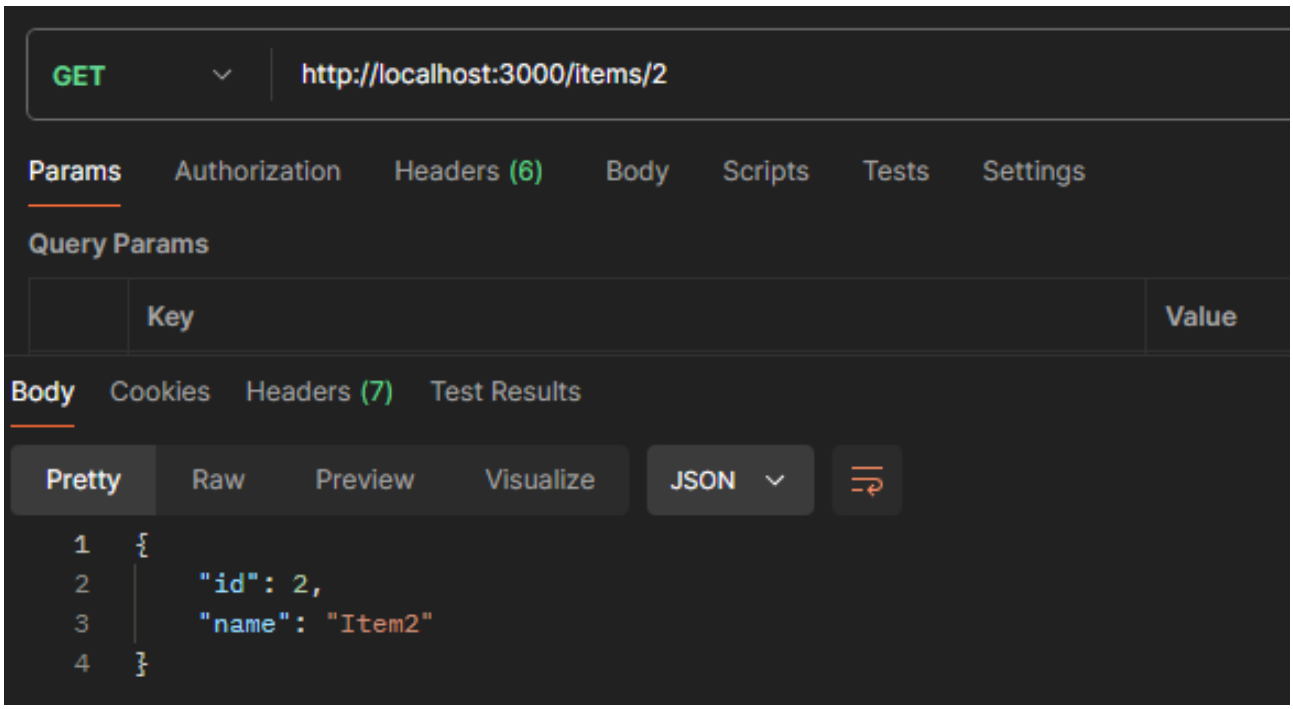

FIGURE 4.4 – Récupération d'un élément par ID via une requête GET dans Postman

4.9.4 4. Test du DELETE Endpoint

Ce test permet de supprimer un élément spécifique en utilisant son ID via une requête DELETE.

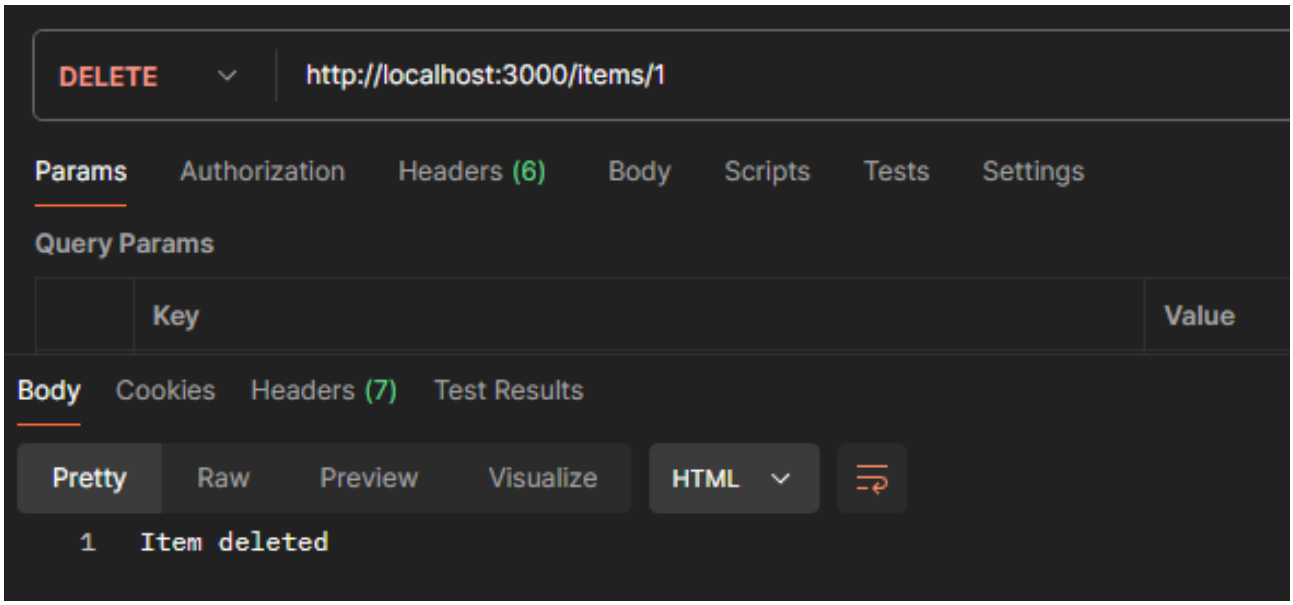

FIGURE 4.5 – Suppression d'un élément via une requête DELETE dans Postman

Chapitre 5

Conclusion

Cette application CRUD basique montre comment utiliser Express.js pour gérer les opérations de création, lecture, mise à jour et suppression sur des données locales. Bien que les données soient stockées dans une variable locale dans cet exemple, l'application pourrait être étendue pour intégrer une base de données réelle, comme MongoDB, pour un usage en production.