Filière : Génie Informatique

3eme Année du Cycle ingénieur

Présenté par :
LACHHAB MOSAAB

Encadré par :
Pr. OURDOU AMAL

Rapport - TP Express.js

Année universitaire : 2024/2025

1.Introduction

Dans ce TP, nous avons utilisé Express.js, un framework minimaliste pour Node.js, afin de créer une application web basique de type CRUD (Create, Read, Update, Delete). L'objectif était de comprendre les concepts fondamentaux d'Express.js, notamment la création de routes, l'utilisation des middlewares et la gestion d'une application serveur. Nous avons mis en place un serveur HTTP permettant de gérer des items à travers différents points de terminaison (endpoints) RESTful.

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

Express.js est un framework web pour **Node.js** qui facilite la gestion des requêtes HTTP, des routes, et des middlewares. Il permet de développer rapidement des applications web ou des APIs. Avec Express.js, nous pouvons :

- Créer des serveurs web performants.
- Gérer des routes pour des APIs.
- Manipuler des données entrantes (JSON, formulaires).
- Utiliser des middlewares pour ajouter des fonctionnalités comme l'authentification, la gestion des erreurs, etc.

3.Utilisation des Middlewares dans Express.js

Les middlewares sont des fonctions qui ont accès à l'objet request (requête), response (réponse), et à la méthode next () qui permet de passer au middleware suivant. Ils sont utilisés pour :

- Ajouter des fonctionnalités globales (comme l'authentification, la gestion des erreurs, ou le parsing des données).

  Exemples :

  1. Middleware pour parser le JSON :

app.use(express.json());

Ici, ce middleware permet à notre serveur de comprendre et traiter les données au format JSON dans les requêtes entrantes.

2.Middleware pour gérer les routes :

```
app.get('/', (req, res) => {
res.send('Bienvenue sur Express.js');
});
```

Ce middleware est utilisé pour répondre à la requête **GET** à la racine de notre serveur.

4.Création de l'application CRUD

4.1 Importation d'Express.js

```
const express = require('express');// importer express.js
const app = express();
let items = [];
```

4.2 Middleware pour traiter données sous format JSON

```
app.use(express.json()); // Pour parser les requêtes en JSON
```

4.3 Endpoint GET par défaut

```
app.get('/', (req, res) => {
  res.send('Bienvenue sur Express.js');
});
```

4.4 Endpoint POST pour ajouter item

```
app.post('/items', (req, res) => {

  const newItem = { id: items.length + 1, name: req.body.name };

  items.push(newItem);

  res.status(201).json(newItem);

});
```

4.5 Endpoint GET pour récuperer tous les items

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

    res.json(items);

  });
```

4.6 Endpoint GET pour récuperer un item par son ID

```
app.get('/items/:id', (req, res) => {

    const itemId = parseInt(req.params.id);

    const item = items.find(item => item.id === itemId);



    if (item) {

      res.json(item);

    } else {

      res.status(404).json({ message: 'Item non trouvé' });

    }

  });
```

4.7 Endpoint PUT pour modifier un item existant

```
app.put('/items/:id', (req, res) => {

    const itemId = parseInt(req.params.id);

    const itemIndex = items.findIndex(item => item.id === itemId);



    if (itemIndex !== -1) {

      items[itemIndex].name = req.body.name;

      res.json({ message: 'Item mis à jour avec succès', item:
items[itemIndex] });

    } else {

      res.status(404).json({ message: 'Item non trouvé' });

    }

  });
```

4.8 Endpoint DELETE pour supprimer un item existant

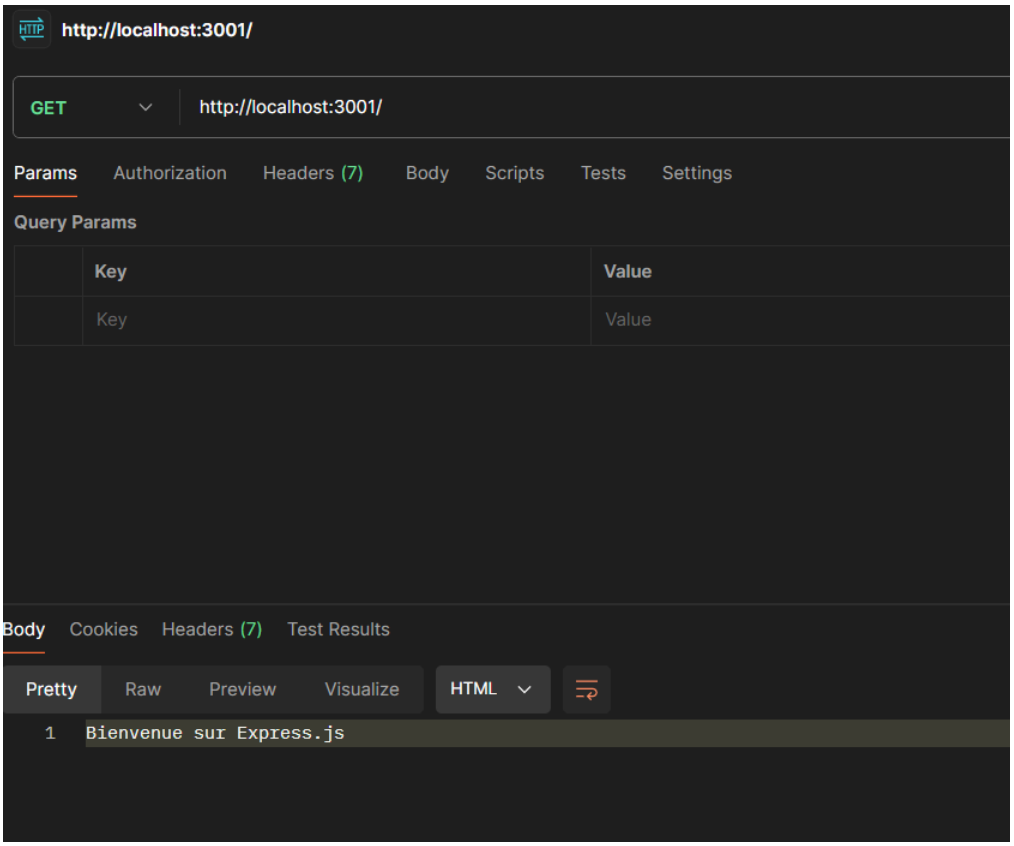
```
app.delete('/items/:id', (req, res) => {

    const itemId = parseInt(req.params.id);

    const itemIndex = items.findIndex(item => item.id === itemId);



    if (itemIndex !== -1) {

      items.splice(itemIndex, 1);

      res.json({ message: 'Item supprimé avec succès' });

    } else {

     res.status(404).json({ message: 'Item non trouvé' });

    }

  });
```

4.9 Lancer le serveur dans le port 3001

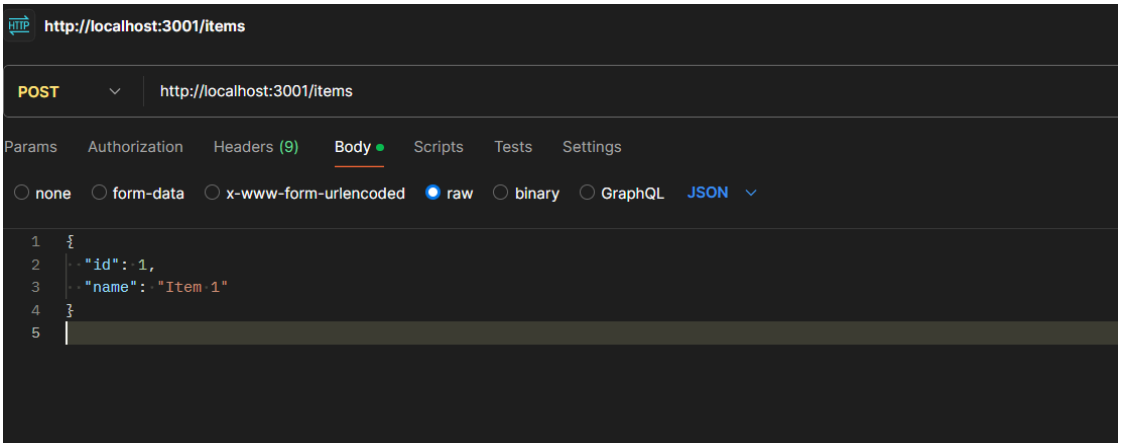
```
app.listen(3001, () => {

    console.log("Serveur démarré sur le port 3001");

  });
```

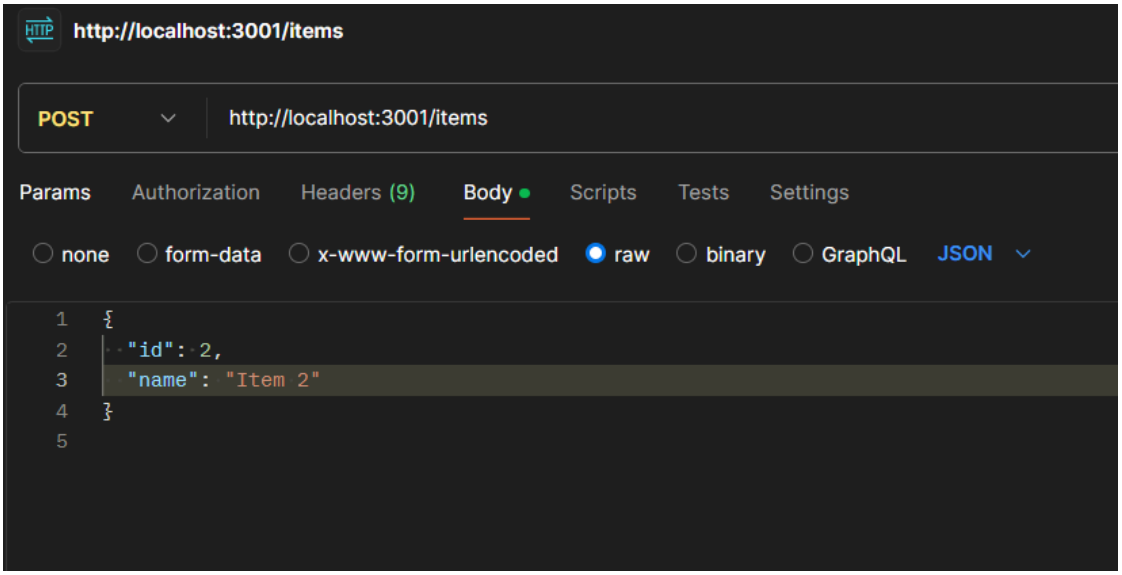
5. Test de l'application sur Postman

5.1 GET par defaut

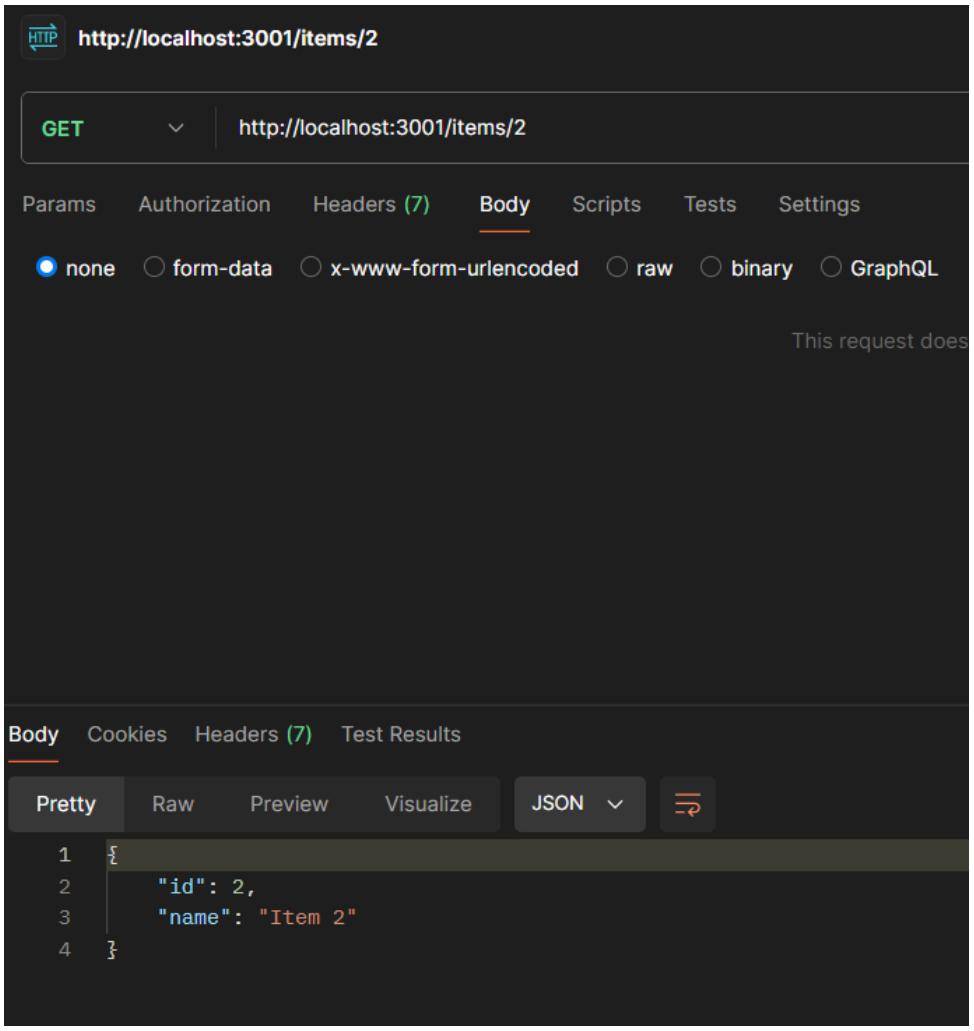


5.2 POST

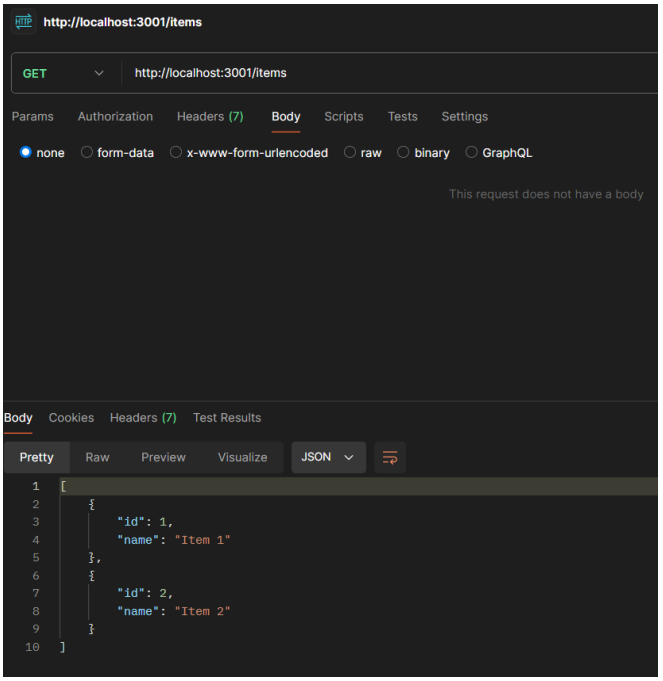

http://localhost:3001/items
POST
http://localhost:3001/items
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"id": 1,
"name": "Item 1"
}


http://localhost:3001/items
POST
http://localhost:3001/items
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"id": 2,
"name": "Item 2"
}

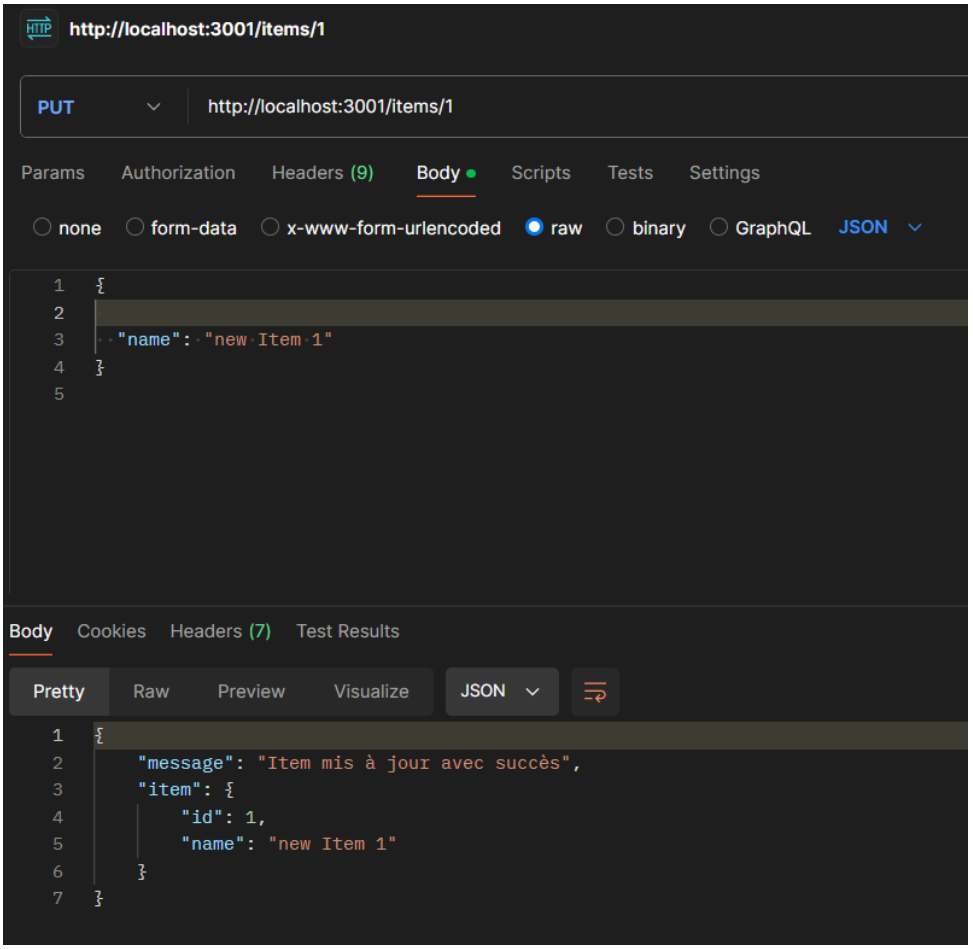
5.3 GET by ID



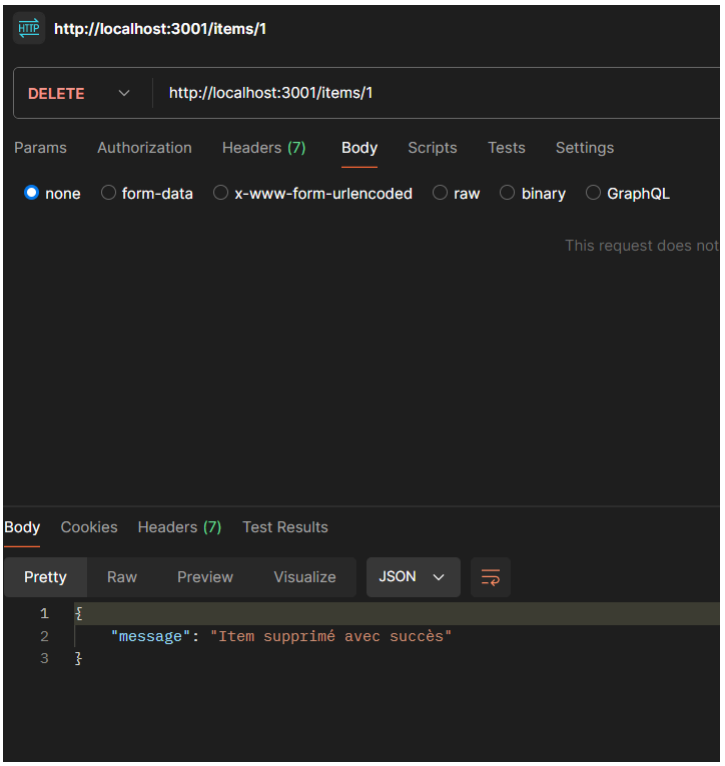
5.4 GET

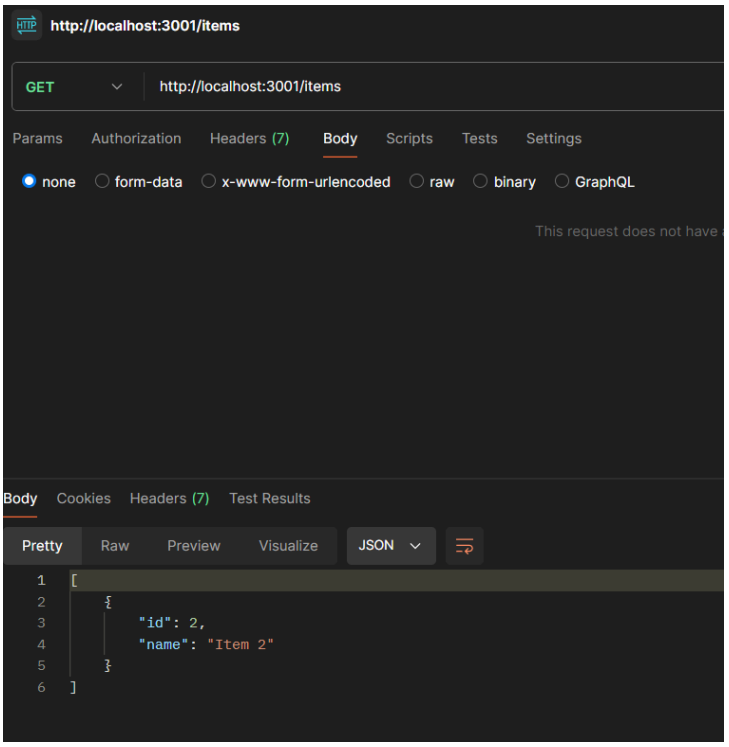


5.5 PUT



5.6 DELETE





6. Conclusion

À travers ce TP, nous avons vu comment utiliser Express.js pour créer une application CRUD basique. Nous avons appris à utiliser des middlewares, à configurer un serveur HTTP, et à manipuler des données via des points de terminaison RESTful. Grâce à Express, la gestion des requêtes HTTP et des routes devient plus simple et rapide, permettant un développement web efficace et modulaire.