Université Sultan Moulay Slimane École Nationale des Sciences Appliquées
-Khouribga-

Rapport de TP (Express.JS)
Filière Informatique et Ingénierie des données

Rapport de Tp Node.JS

Réalisé par : Salma faraj

professeur :
AMAL OURDOU

21 octobre 2024

Table des matières

1 Qu'est-ce que Express.js et ce que nous pouvons faire avec ?

1.1 definition de Express.js :

Express.js est un framework web minimaliste pour Node.js qui simplifie le développement d'applications web et d'API. Il offre des outils simples pour gérer les routes, les requêtes HTTP, et les réponses.

1.2 Utilisation de Express.js :

Express permet de créer des applications serveur rapides et légères, telles que :

- Des applications web dynamiques
- Des API RESTful
- Des applications d'une page (Single Page Applications)
- Des applications en temps réel (avec WebSockets)

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

Un middleware dans Express.js est une fonction qui a accès à l'objet requête (req), l'objet réponse (res) et la prochaine fonction middleware dans le cycle de requête-réponse. Les middlewares permettent d'exécuter du code, de modifier des requêtes ou des réponses, de terminer le cycle de requête-réponse ou d'appeler le prochain middleware.

Un middleware reçoit trois arguments principaux :
req : l'objet requête (request), contenant les informations sur la requête HTTP.
res : l'objet réponse (response), utilisé pour envoyer une réponse au client.
next : une fonction qui permet de passer au middleware suivant dans la chaîne.

2.1 Exemples de middlewares :

2.1.1 Middleware de journalisation

Ce middleware enregistre chaque requête reçue par le serveur. Cela permet de suivre les requêtes effectuées. Dans cet exemple, chaque fois qu'une requête est reçue, le type de méthode (GET, POST, etc.) et l'URL sont affichés dans la console. La fonction next() est appelée pour passer au middleware suivant ou à l'endpoint.

```
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next(); // Passe au middleware suivant
});
```
*

2.1.2 Middleware d'analyse JSON

Express inclut un middleware intégré qui permet d'analyser les corps des requêtes au format JSON. Cela facilite le traitement des données envoyées dans les requêtes POST. Ce middleware convertit le corps de la requête JSON en un objet JavaScript accessible

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

via req.body. Cela permet de traiter facilement les données envoyées par le client lors des requêtes POST.

3 Création d'une application CRUD simple

3.1 Créer un répertoire de projet

Un nouveau répertoire a été créé pour le projet afin d'organiser les fichiers. Cela a permis de regrouper tous les fichiers nécessaires dans un seul emplacement.

3.2 Initialiser un projet Node.js

Le projet a été initialisé à l'aide de la commande npm init -y, ce qui a généré un fichier package.json. Ce fichier contient des métadonnées importantes sur le projet, telles que son nom, sa version et ses dépendances

```
PS C:\Users\SALMA\Desktop\test\tp\CRUD> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
```

3.3 Installer Express

Le framework Express a été installé à l'aide de la commande npm install express. Express facilite la création d'applications web en fournissant des fonctionnalités robustes et flexibles pour gérer les requêtes HTTP.

```
PS C:\Users\SALMA\Desktop\test\tp\CRUD> npm install express
>>

up to date, audited 66 packages in 2s

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

found 0 vulnerabilities
PS C:\Users\SALMA\Desktop\test\tp\CRUD>
```

3.4 Configurer Express et exécuter le serveur

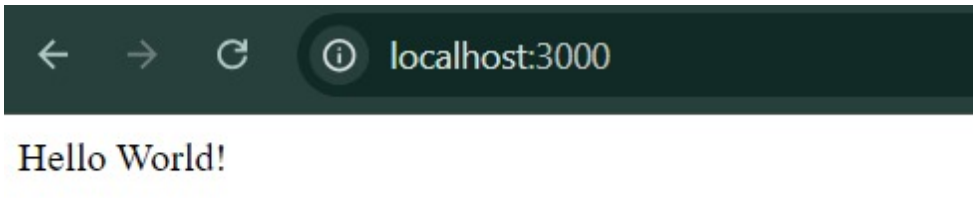
Un fichier, app.js, a été créé. Dans ce fichier, le code suivant a été ajouté pour configurer le serveur Express : Cela a permis de créer une instance de l'application Express et de configurer le serveur pour écouter les requêtes sur le port 3000. Le middleware express.json() a été utilisé pour analyser les corps de requêtes JSON.

```
CRUD > JS app.js > ...
1   const express = require('express');
2   const app = express();
3
4
5   app.get('/', (req, res) => {
6     res.send('Hello World!');
7   });
8
9   // Configurer le serveur pour écouter sur un port
10  const PORT = 3000;
11  app.listen(PORT, () => {
12    console.log(`Server is running on port ${PORT}`);
13  });
14
```

3.4.1 Execution de serveur

```
PS C:\Users\SALMA\Desktop\test\tp\CRUD> node app.js
Server is running on port 3000
```

3.4.2 Resultas



3.5 Créer un Endpoint POST

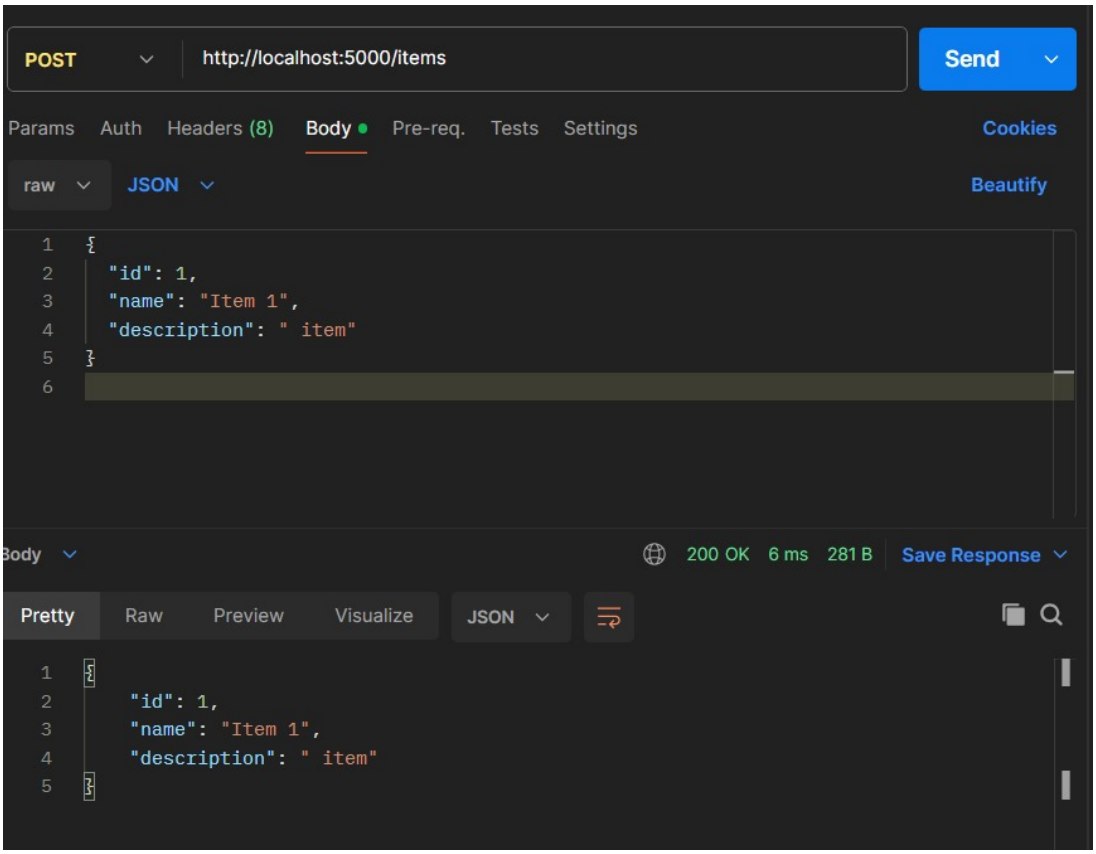
Un endpoint POST a été défini pour permettre l'ajout d'éléments à un tableau en mémoire : Cet endpoint vérifie si un nouvel élément a été fourni. Si l'élément est valide,

```
app.post('/items', (req, res) => {
    const newItem = req.body;
    if (!newItem || Object.keys(newItem).length === 0) {
        return res.status(400).send('Un élément est requis');
    }
    items.push(newItem);
    console.log('Éléments actuels :', items);
    res.status(201).send(items);
});
```

il est ajouté au tableau et une réponse avec le statut 201 est envoyée.

3.5.1 Test dans Postman

Dans Postman, une requête POST a été effectuée vers l'URL http ://localhost :3000/items en envoyant un objet JSON dans le corps de la requête. Le serveur a répondu avec la liste mise à jour des éléments.



3.6 Créer un Endpoint GET

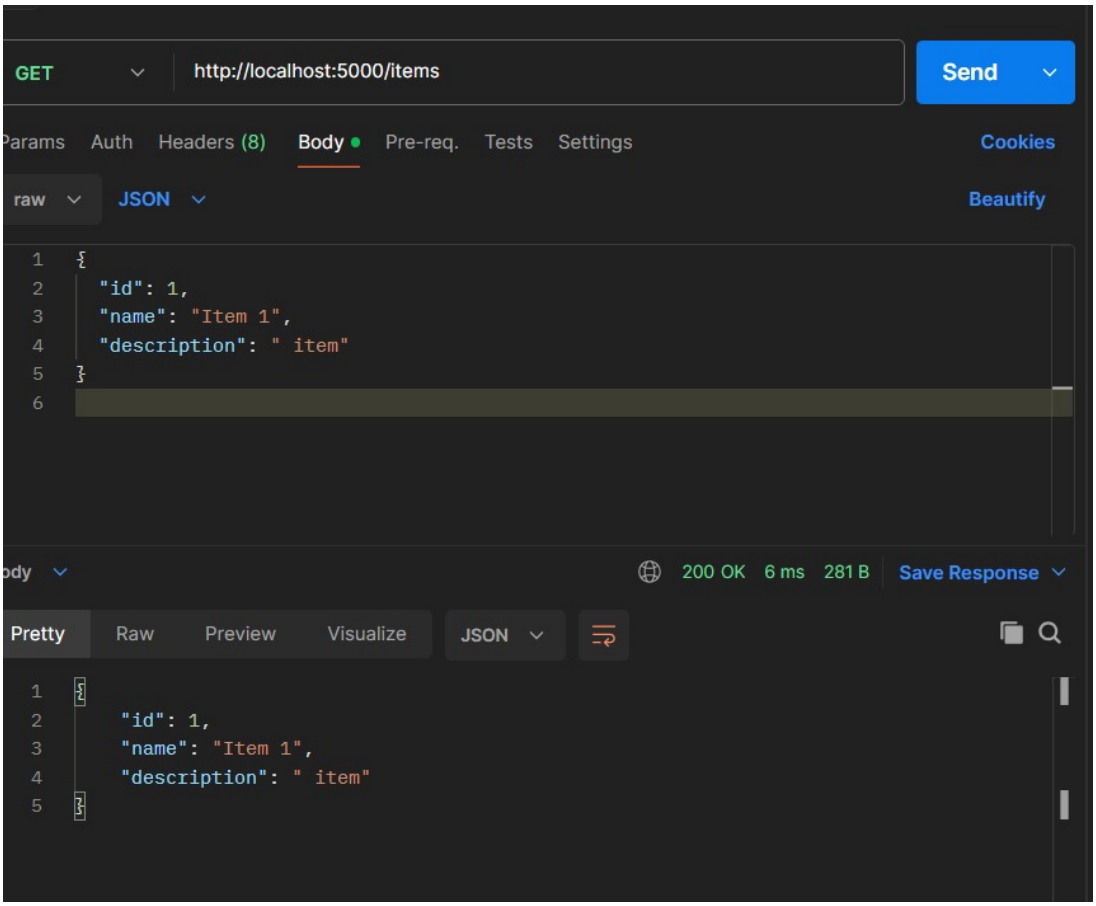
Un endpoint GET a été créé pour retourner tous les éléments stockés :

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

Cet endpoint envoie tous les éléments présents dans le tableau avec un statut 200.

3.6.1 Test dans Postman

Dans Postman, une requête GET a été envoyée à l'URL http ://localhost :3000/items. La réponse renvoyée contenait tous les éléments actuellement dans la liste.



3.7 Créer un Endpoint GET par ID

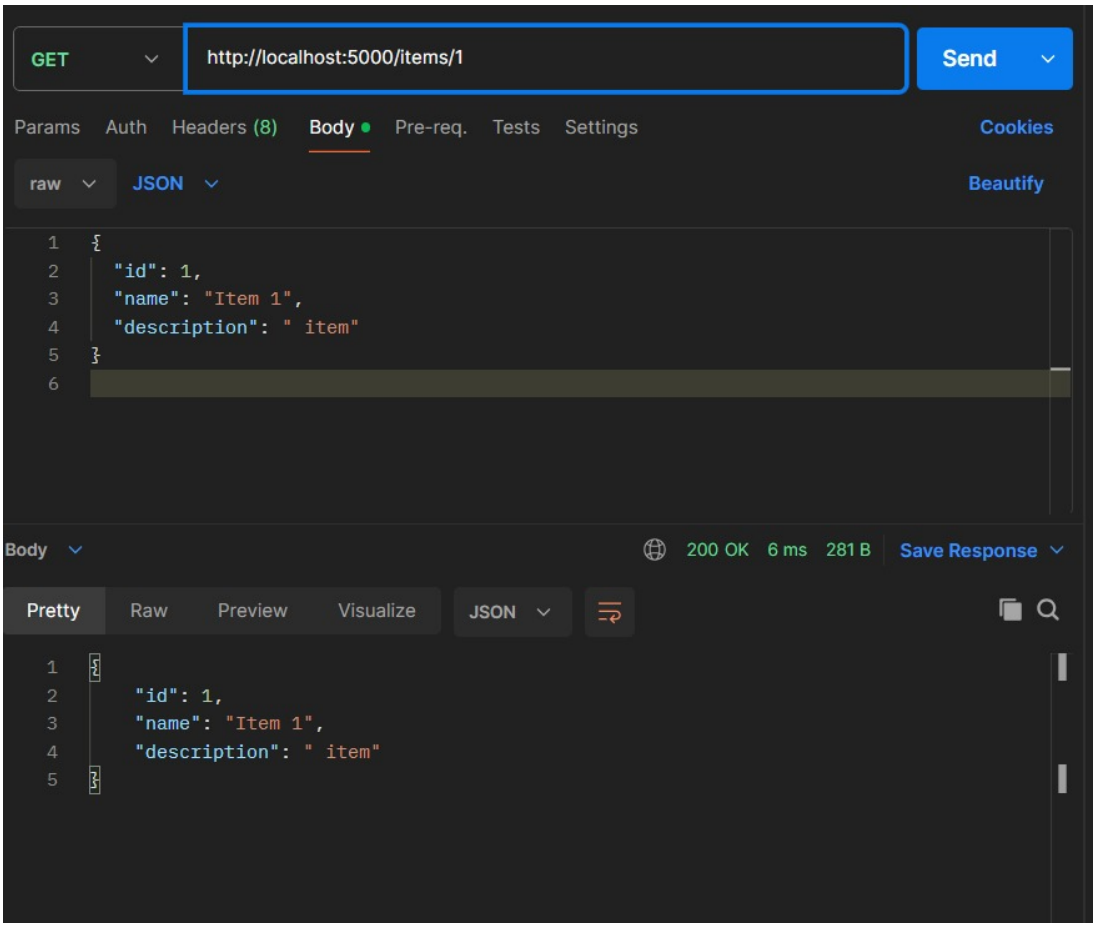
Un endpoint GET a été mis en place pour récupérer un élément spécifique par son ID : Cette méthode recherche l'élément par ID et retourne une réponse appropriée en fonction

```
app.get('/items/:id', (req, res) => {
  const id = parseInt(req.params.id); // Extraire l'ID de l'URL
  const item = items.find(i => i.id === id); // Trouver l'élément par ID

  if (item) {
    res.json(item); // Si trouvé, retourner l'élément
  } else {
    res.status(404).json({ message: 'Élément non trouvé' }); // Retourner une erreur 404 si non trouvé
  }
});
```

de la présence ou de l'absence de l'élément.

3.7.1 Test dans Postman

Dans Postman, une requête GET a été envoyée à l'URL http ://localhost :3000/items/1 (en supposant que l'élément avec l'ID 1 existe). Si l'élément existe, il a été renvoyé dans la réponse.



3.8 8. Créer un Endpoint PUT

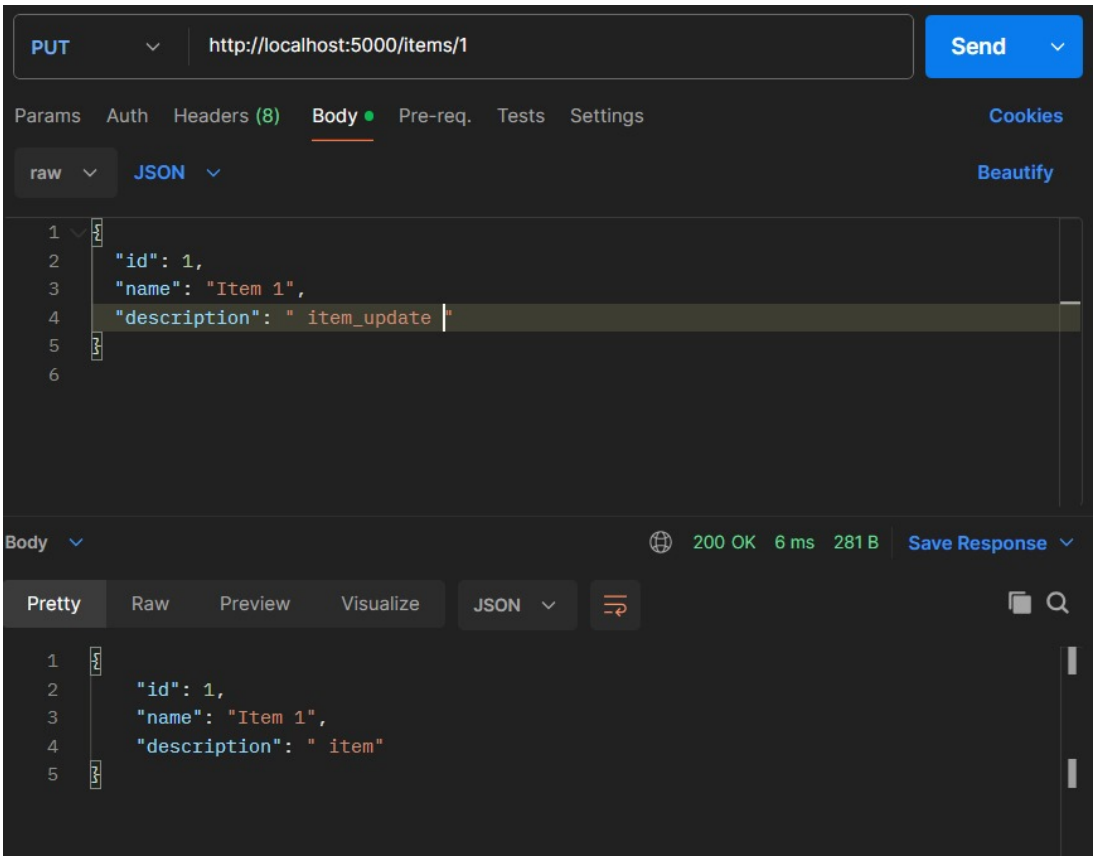
Un endpoint PUT a été configuré pour permettre la mise à jour d'un élément existant

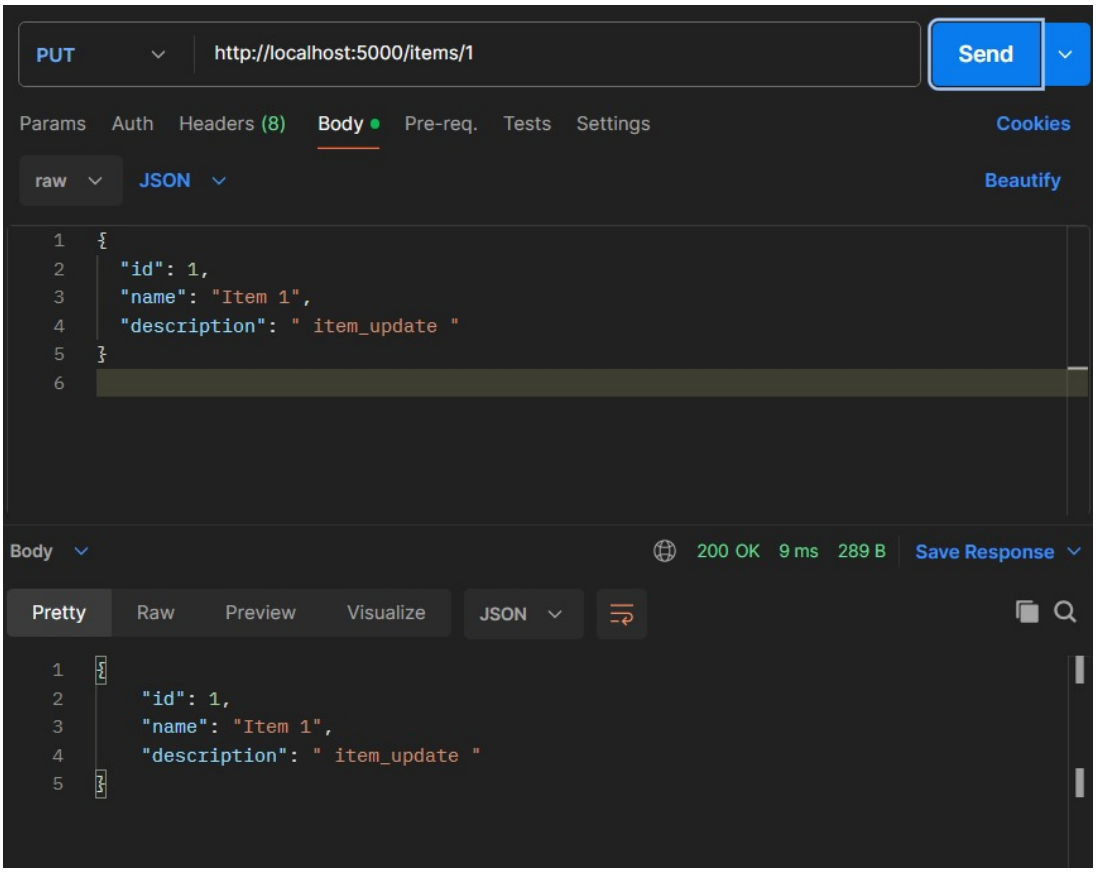
```
app.put('/items/:id', (req, res) => {
  const id = parseInt(req.params.id); // Extraire l'ID
  const index = items.findIndex(i => i.id === id); // Trouver l'index de l'élément

  if (index !== -1) {
      items[index] = { ...items[index], ...req.body }; // Mettre à jour l'élément
      res.status(200).json(items[index]); // Retourner l'élément mis à jour
  } else {
      res.status(404).json({ message: 'Élément non trouvé' }); // Retourner une erreur 404 si non trouvé
  }
});
```

Cet endpoint recherche l'élément par ID, met à jour les données si l'élément est trouvé et renvoie l'élément mis à jour.

3.8.1 Test dans Postman

Une requête PUT a été envoyée à l'URL http ://localhost :3000/items/1 avec un corps contenant les données mises à jour. Si l'élément avec l'ID spécifié existe, il a été mis à jour et renvoyé dans la réponse.




PUT
http://localhost:5000/items/1
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"id": 1,
"name": "Item 1",
"description": " item_update "
}
Body
200 OK
9 ms
289 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"name": "Item 1",
"description": " item_update "
}

3.9 9. Créer un Endpoint DELETE

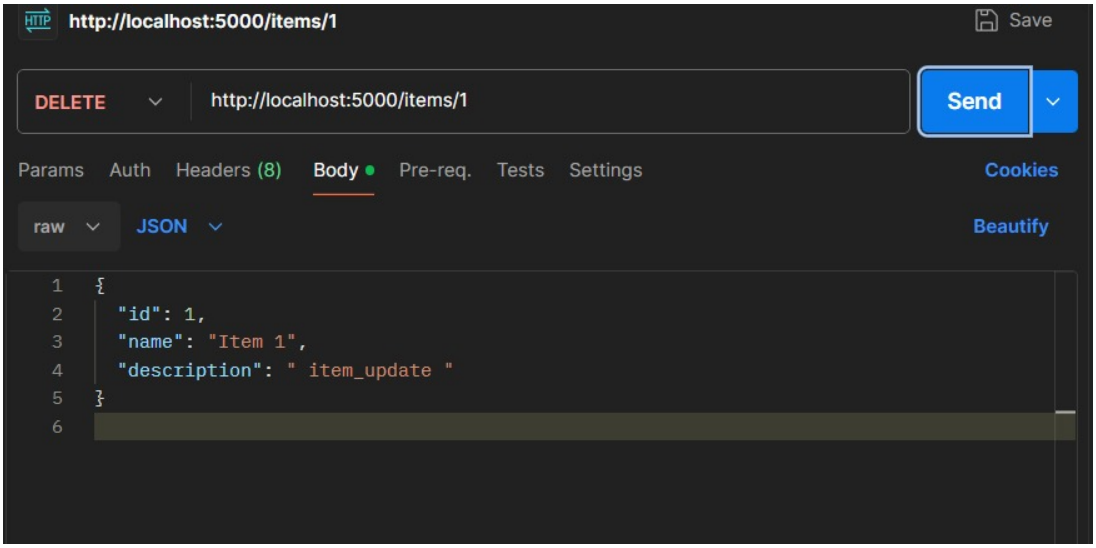
Un endpoint DELETE a été mis en place pour supprimer un élément par son ID :

```
app.delete('/items/:id', (req, res) => {
  const id = parseInt(req.params.id); // Extraire l'ID
  const index = items.findIndex(i => i.id === id); // Trouver l'index de l'élément

  if (index !== -1) {
      const deletedItem = items.splice(index, 1); // Retirer l'élément du tableau
      res.status(200).json(deletedItem); // Retourner l'élément supprimé
  } else {
      res.status(404).json({ message: 'Élément non trouvé' }); // Retourner une erreur 404 si non trouvé
  }
});
```

Ce code permet de supprimer un élément du tableau en fonction de son ID, retournant une réponse appropriée selon que l'élément a été trouvé ou non.

3.9.1 Test dans Postman

Une requête DELETE a été envoyée à l'URL http ://localhost :3000/items/1. Si l'élément avec l'ID spécifié existait, il a été supprimé et la réponse a contenu l'élément supprimé.



3.10 Démarrage du Serveur

Le serveur a été démarré avec le code suivant :

```
const PORT = 3000;
app.listen(PORT, () => {
    console.log("Server is running ");
});
```

4 Conclusion

Express.js est un framework web minimaliste mais puissant qui simplifie le développement d'applications web et d'API avec Node.js. Grâce à sa flexibilité et à son système de middleware, il permet de gérer facilement les requêtes, les réponses et de modifier le comportement d'une application en ajoutant des fonctionnalités comme la journalisation, la gestion des erreurs, l'authentification, et bien plus encore.