ENSA
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES
KHOURIBGA

Université Sultan Moulay Slimane

ENSA KHOURIBGA ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

Compte rendu TP (Express.js)

Réalisé par:

- Rachidi aya

Introduction

Ce projet a pour objectif de développer une application CRUD (Create, Read, Update, Delete) en utilisant Express.js, un framework minimaliste pour Node.js. Cette application permet de manipuler une collection d'éléments via des points de terminaison API. Pour tester les différentes fonctionnalités de l'application, j'utilise Postman.

1. Explication d'Express.js

Express.js est un framework web pour Node.js qui simplifie le développement d'applications web et d'API REST. Il offre une série de méthodes HTTP et de middlewares qui facilitent la gestion des requêtes, la définition des routes, et la manipulation des réponses envoyées aux clients. Grâce à sa structure minimaliste, il est rapide à mettre en place et permet une grande flexibilité.
Avec Express.js, on peut :

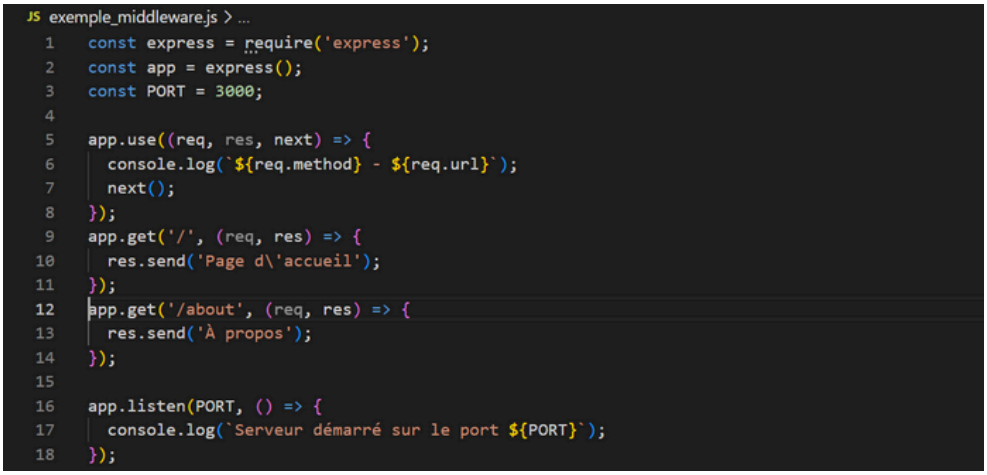
- Créer des API RESTful
- Gérer des sessions, des cookies, des formulaires
- Créer des applications web dynamiques

2. Explication des middlewares

Les middlewares dans Express.js sont des fonctions exécutées lors du traitement des requêtes. Ils permettent d'intercepter les requêtes pour exécuter une logique avant de passer à la suite du traitement.

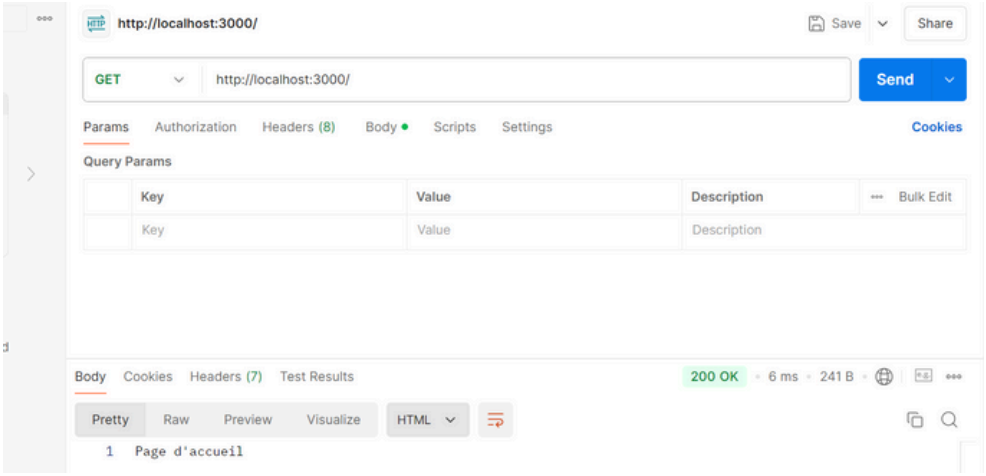
- **Exemple 1 :**

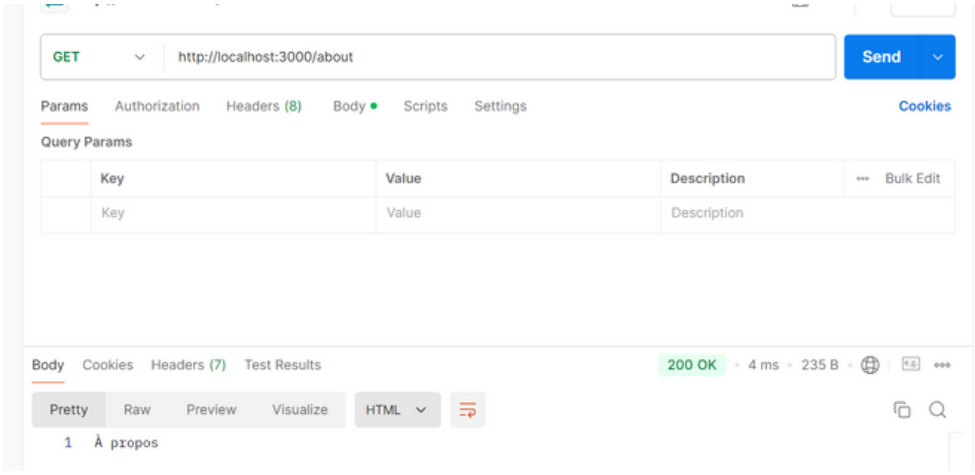
  Un middleware pour enregistrer les informations dechaque requête (méthode HTTP et URL) dans la console.



  Ici, chaque requête est loguée avant d'être transmise à la prochaine fonction middleware.

=>Execution






```
PS C:\Users\DELL\Desktop\tp_express> node exemple_middleware.js
Serveur démarré sur le port 3000
GET - /
GET - /about
```

- **Exemple 2 :**

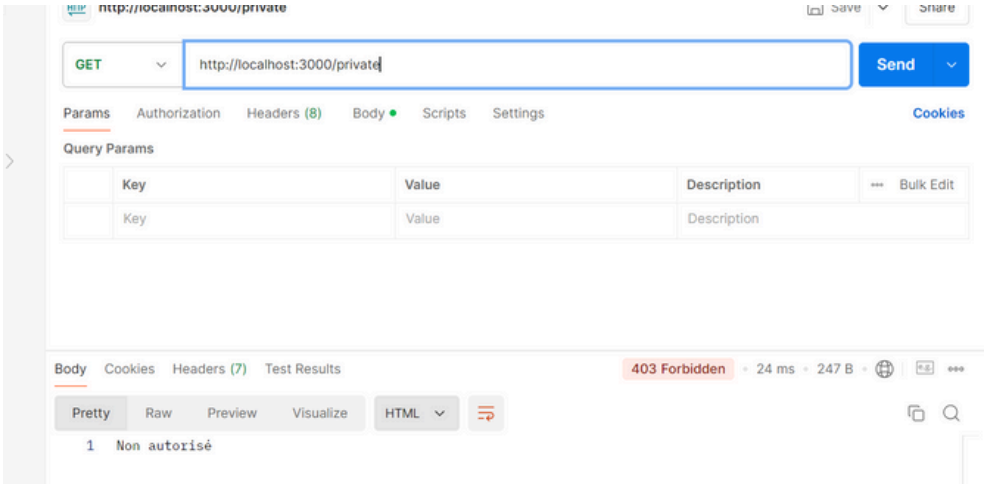
  Un middleware pour vérifier si l'utilisateur est authentifié avant d'accéder à certaines routes privées.

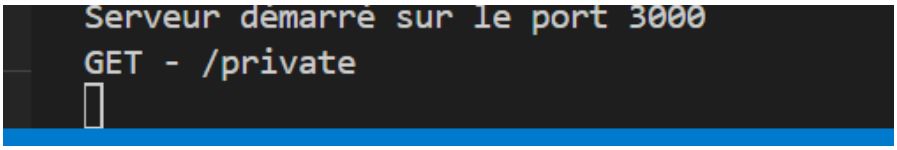
```
app.use((req, res, next) => {
    req.isAuthenticated = () => {
      return false;
    };
    next();
  });
  app.use((req, res, next) => {
    if (!req.isAuthenticated()) {
      return res.status(403).send('Non autorisé');
    }
    next();
  });

  app.get('/private', (req, res) => {
    res.send('Page privée accessible uniquement aux utilisateurs authentifiés');
  });

app.listen(PORT, () => {
  console.log(`Serveur démarré sur le port ${PORT}`);
});
```

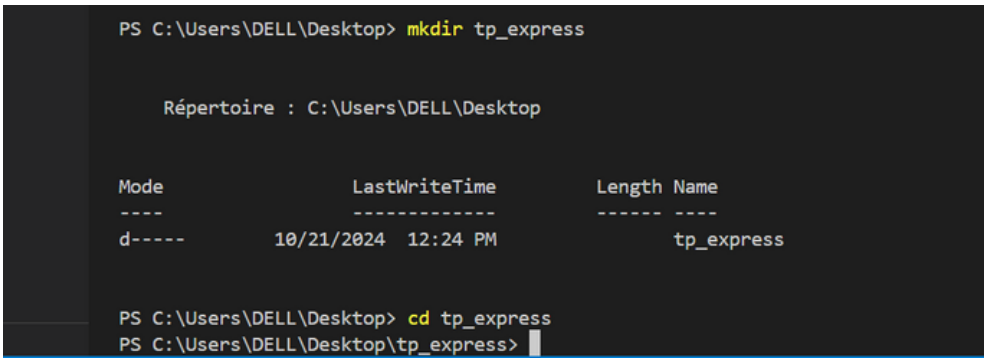
=>Execution



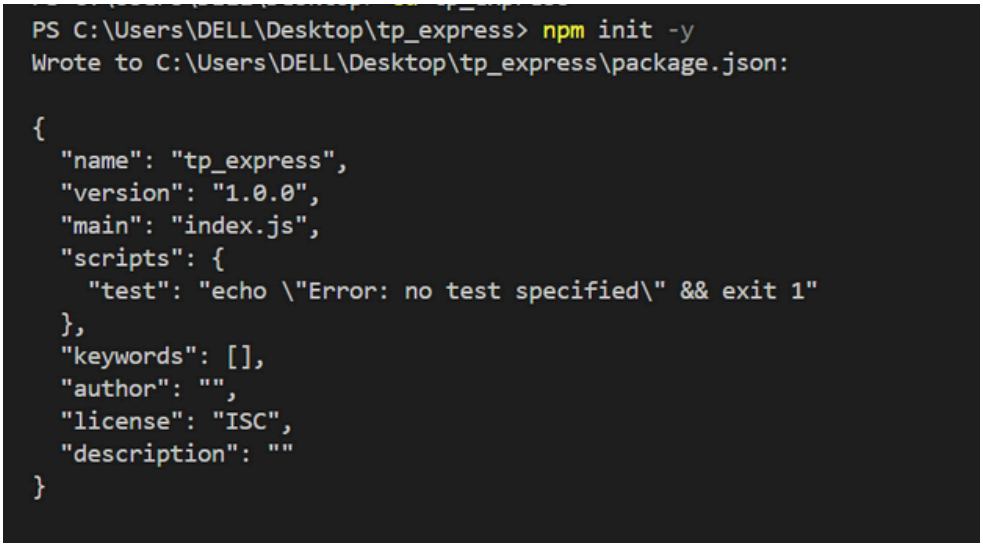


Création d'une application CRUD

1. Création du répertoire de projet



2. Initialisation du projet Node.js



3. Installation d'Express

```
PS C:\Users\DELL\Desktop\tp_express> npm install express

added 65 packages, and audited 66 packages in 6s

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

found 0 vulnerabilities
PS C:\Users\DELL\Desktop\tp_express>
```

4. Configuration d'Express et lancement du serveur

Pour configurer le serveur, j'ai créé un fichier index.js où j'ai importé Express et configuré le serveur pour écouter sur le port 3000

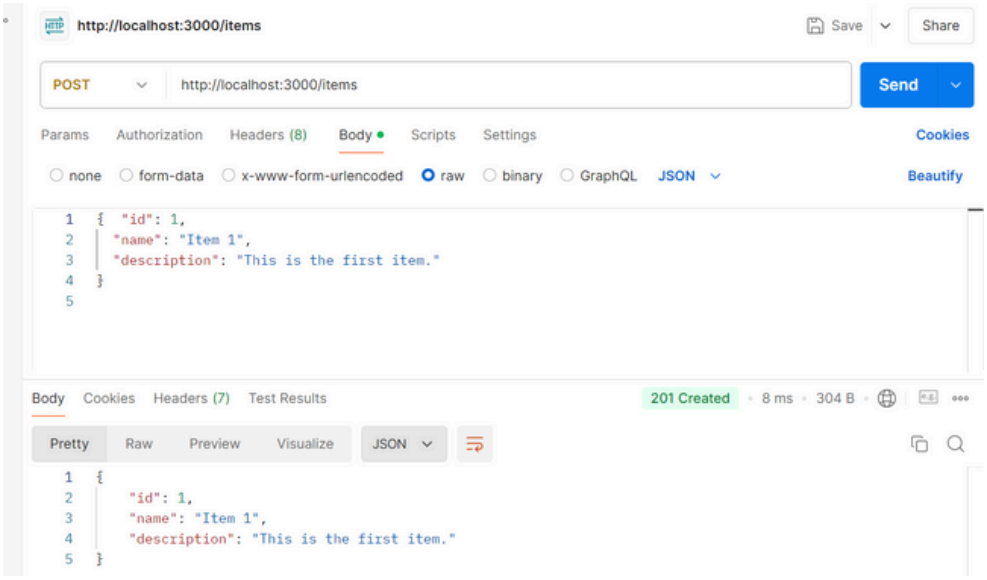
5. Création d'un point de terminaison POST

Ce point de terminaison permet d'ajouter un élément à une variable locale items. Il reçoit les données en format JSON via le corps de la requête et les stocke dans un tableau.
Le tableau items sert de base de données temporaire. Chaque fois qu'un élément est ajouté, il est sauvegardé dans ce tableau, puis la réponse avec le statut 201 (créé) est renvoyée.

```
let items = [];

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

=>Execution

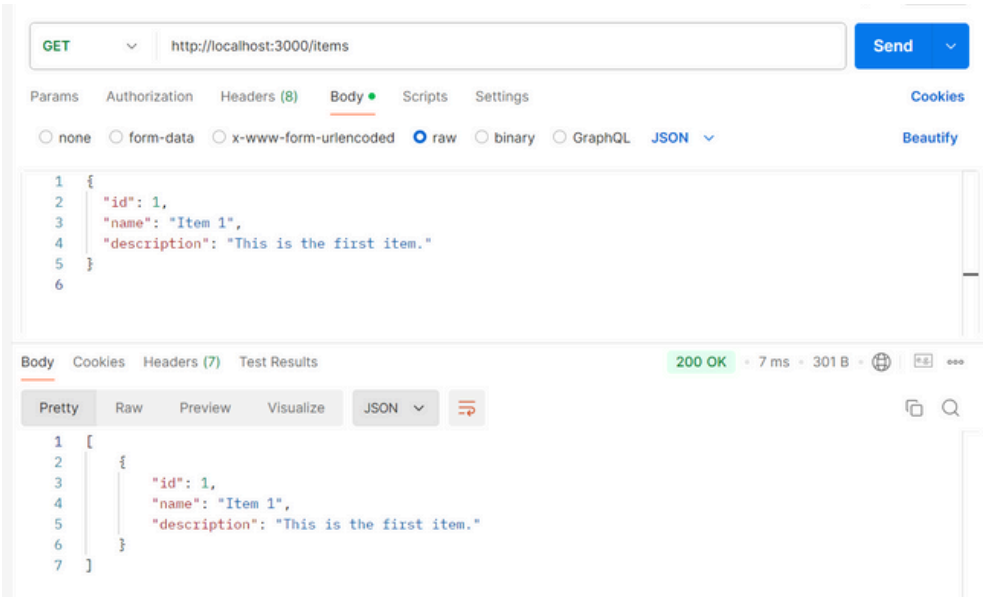


6. Création d'un point de terminaison GET (Tous les éléments)

Lorsque cette route est appelée, le tableau items est renvoyé en tant que réponse.

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

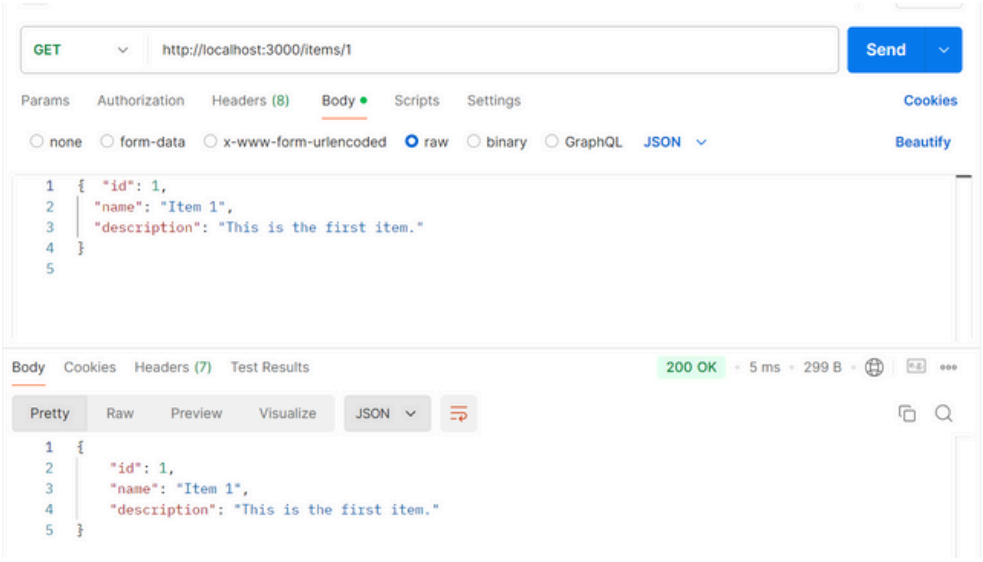
=>Execution



7. Création d'un point de terminaison GET par 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 non trouvé');
  res.send(item);
});
```

=>Execution



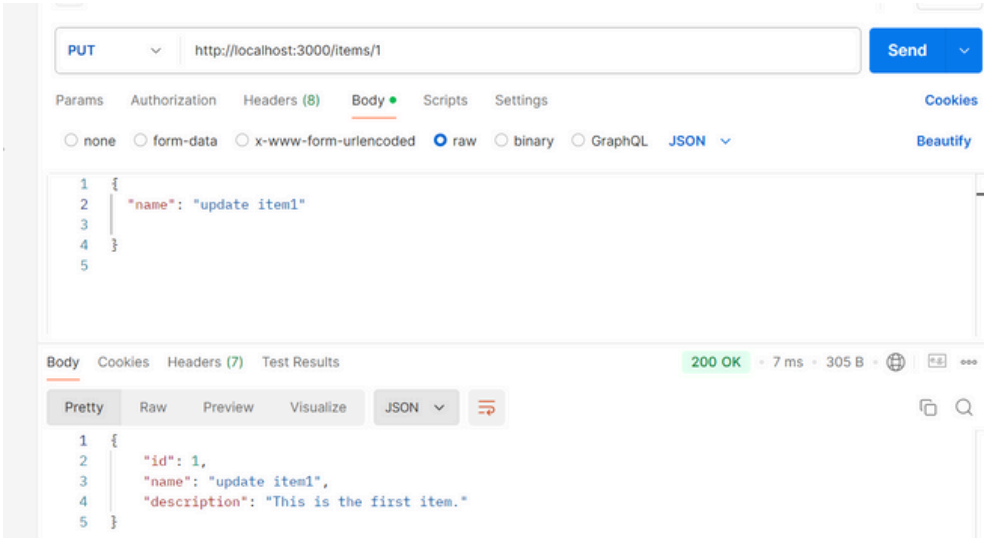
8. Création d'un point de terminaison PUT

Après avoir récupéré l'élément par ID, ses propriétés sont mises à jour avec les nouvelles données fournies dans le corps de la requête.

```
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 non trouvé');

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

=>Execution



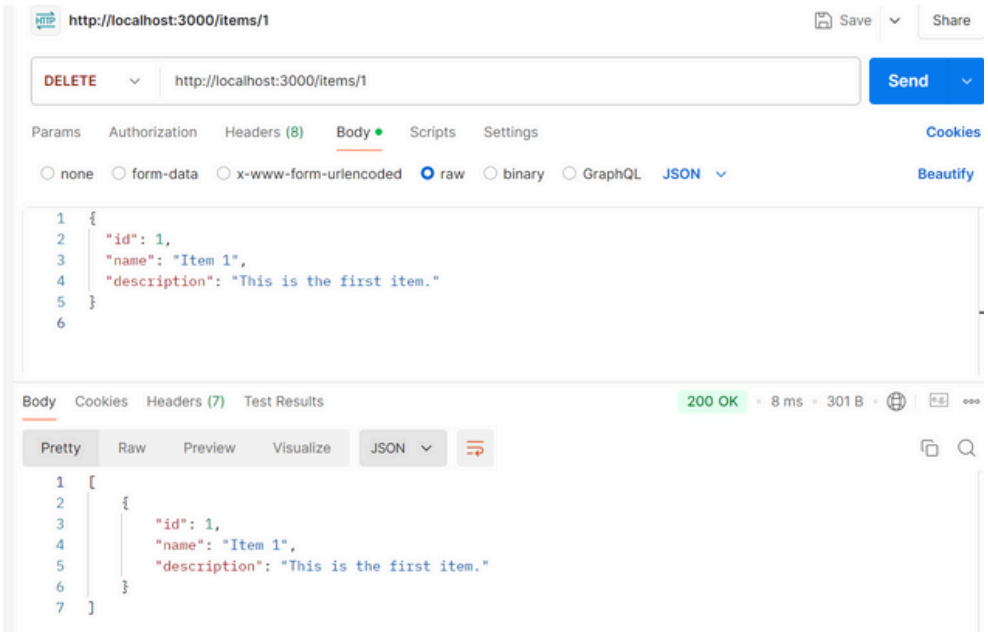
9. Création d'un point de terminaison DELETE

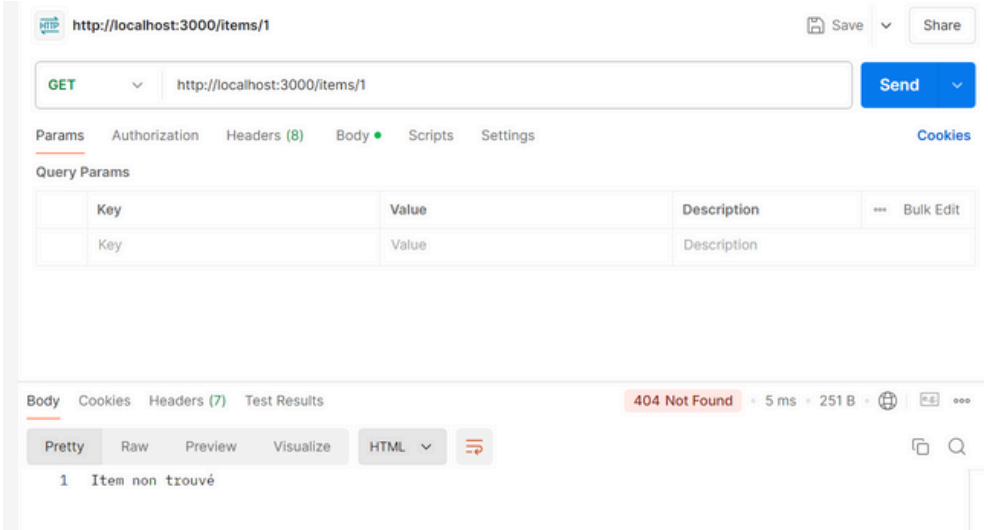
Cette fonction recherche l'élément dans le tableau par son ID et le supprime si trouvé.

```
app.delete('/items/:id', (req, res) => {
  const index = items.findIndex(i => i.id === parseInt(req.params.id));
  if (index === -1) return res.status(404).send('Item non trouvé');

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

=>Execution





10. Démarrage du serveur

Pour démarrer le serveur, j'ai utilisé:

```
PS C:\Users\DELL\Desktop\tp_express> node index.js
Serveur lancé sur le port 3000
```

11. Test des points de terminaison avec Postman

Enfin, j'ai testé les différentes routes (POST, GET, PUT, DELETE) à l'aide de l'outil Postman pour vérifier que l'application CRUD fonctionne correctement. Postman permet de simuler des requêtes HTTP et de voir les réponses retournées par l'API.

Conclusion

Ce TP m'a permis de comprendre comment utiliser Express.js pour créer une application CRUD simple, avec la gestion des routes et des middlewares. Le framework Express facilite le développement d'applications web, en offrant une structure claire et une syntaxe simple pour manipuler les requêtes HTTP.