Université Sultane Moulay Slimane
École Nationale des Sciences Appliquées de Khouribga

Rapport de TP :
TP (Express.JS)

Réalisé par :

KHIAR Siham
BENWADI Afrae

Encadré par : OURDOU Amal

Année Universitaire : 2024-2025

1 Introduction

Ce rapport décrit le développement d'une application CRUD simple utilisant **Express.js**. L'application permet de gérer des items à travers une API REST en utilisant les méthodes HTTP (GET, POST, PUT, DELETE).

2 1. Express.js et ses applications

Express.js est un framework pour Node.js permettant de créer des applications web et des API de manière simple et efficace. Avec Express.js, nous pouvons :

- Créer des serveurs web pour des applications frontend (Angular, React).
- Construire des API REST pour la gestion de données.
- Gérer les sessions et les authentifications.
- Développer des applications temps réel comme les chats.

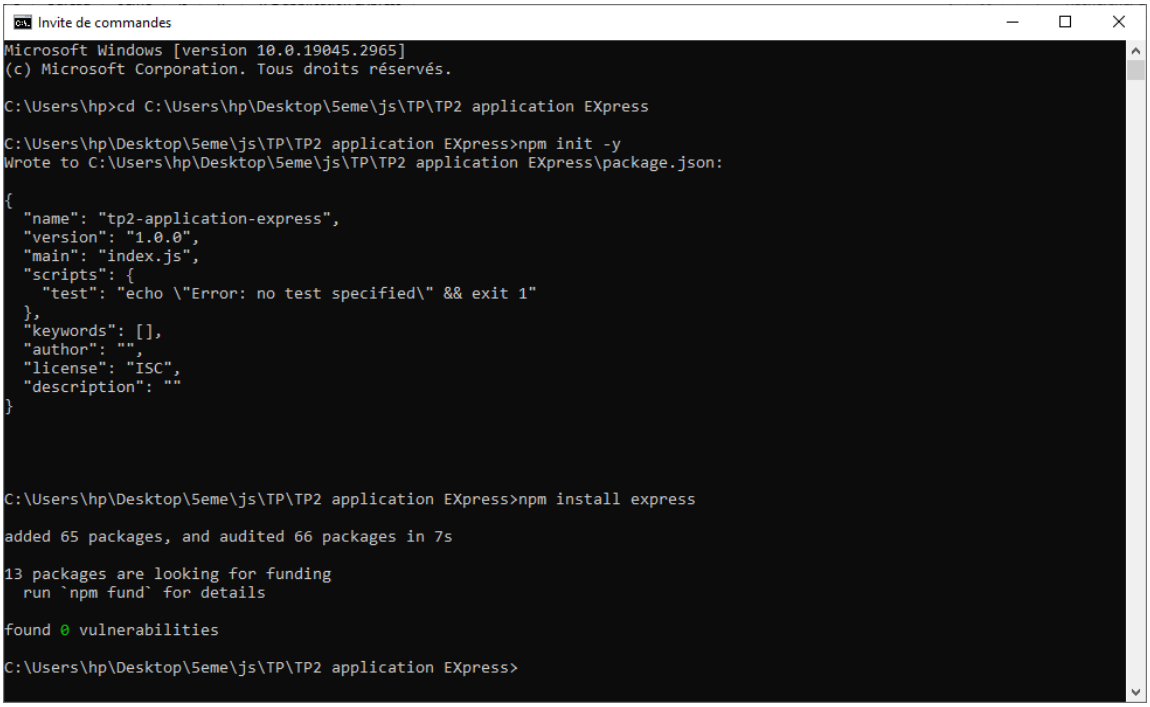
3 2. Les Middlewares dans Express.js

Les **middlewares** dans Express.js sont des fonctions qui s'exécutent pendant le traitement d'une requête avant qu'elle n'atteigne le gestionnaire final. Ils permettent de :

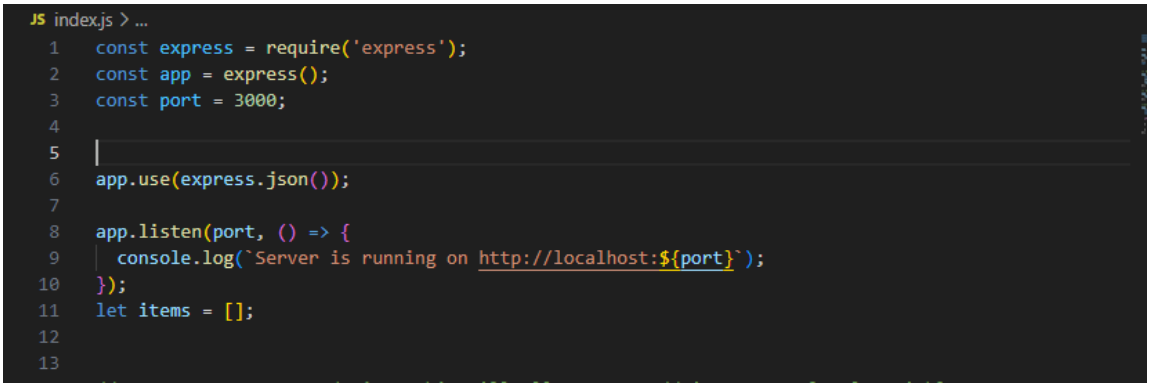
- Modifier les objets `req` et `res`.
- Gérer les erreurs.
- Terminer le cycle de requête/réponse.

4 Création d'une Application CRUD

4.1 1. Créer un répertoire de projet // 2. Initialiser un projet Node.js //3. Installer Express



4.2 4. Configurer le serveur Express



4.3 5. Créer un endpoint POST pour ajouter des items

```
// 5. Create a POST Endpoint: This will allow us to add items to a local variable.
app.post('/items', (req, res) => {
  const item = req.body;
  items.push(item);
  res.status(201).send(item);
});
```

Explication : Ce code définit un endpoint POST à /items pour ajouter un nouvel item à la liste items.

4.4 6. Créer un endpoint GET pour récupérer tous les items

```
// Create a GET Endpoint:  This will allow us to retrieve all items.
app.get('/items', (req, res) => {
  res.send(items);
});
```

Explication : Ce code définit un endpoint GET à /items pour renvoyer la liste complète des items.

4.5 7. Créer un endpoint GET par ID pour récupérer un item

```
//Create a GET Endpoint by ID: This will allow us to get a specific item.
app.get('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const item = items.find(i => i.id === id);

  if (item) {
    res.send(item);
  } else {
    res.status(404).send('Item not found');
  }
});
```

Explication : Cet endpoint récupère un item spécifique à partir de son id. S'il n'est pas trouvé, il retourne une erreur 404

4.6 8. Créer un endpoint PUT pour mettre à jour un item

```
//Create a PUT Endpoint: This will allow us to update an existing item.
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] = req.body;
    res.send(items[index]);
  } else {
    res.status(404).send('Item not found');
  }
});
```

Explication : Cet endpoint met à jour un item existant en remplaçant ses données par celles fournies dans la requête.

4.7 9. Créer un endpoint DELETE pour supprimer un item

```
//Create a DELETE Endpoint: This will allow us to delete an item.
app.delete('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const index = items.findIndex(i => i.id === id);

  if (index !== -1) {
    const deletedItem = items.splice(index, 1);
    res.send(deletedItem);
  } else {
    res.status(404).send('Item not found');
  }
});
```

Explication : Cet endpoint supprime un item en fonction de son id. S'il est trouvé, il est retiré de la liste items.

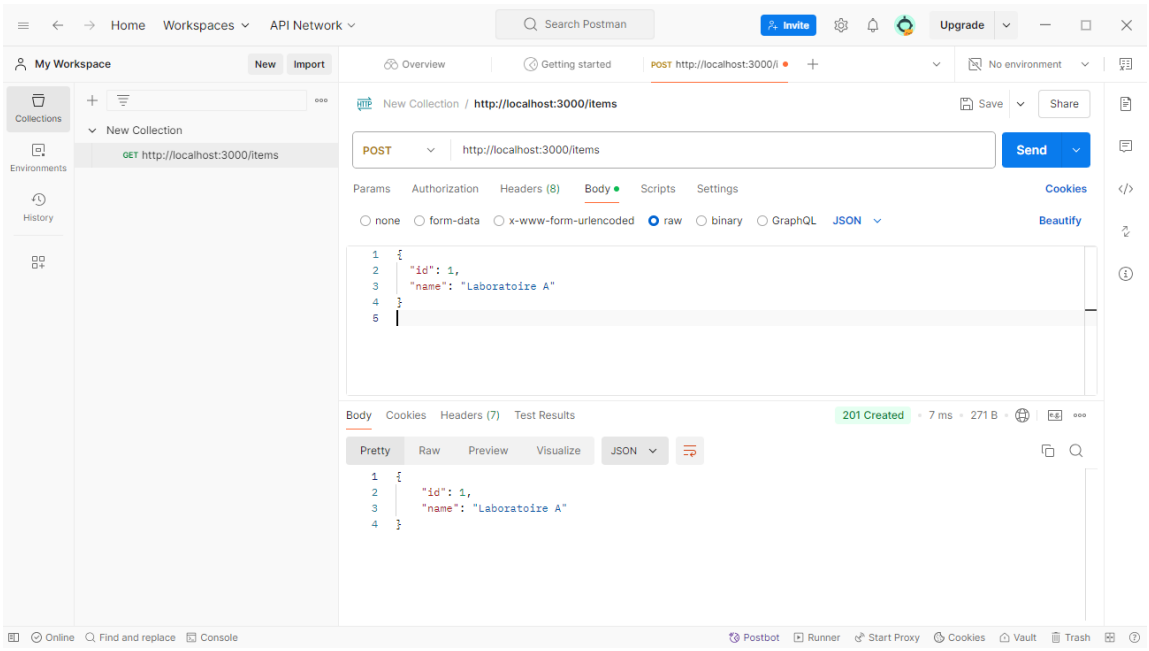
4.8 10. Démarrer le serveur

```
C:\Users\hp\Desktop\5eme\js\pokemon-battle-game\TP2 application EXpress>node index.js
Server is running on http://localhost:3000
```

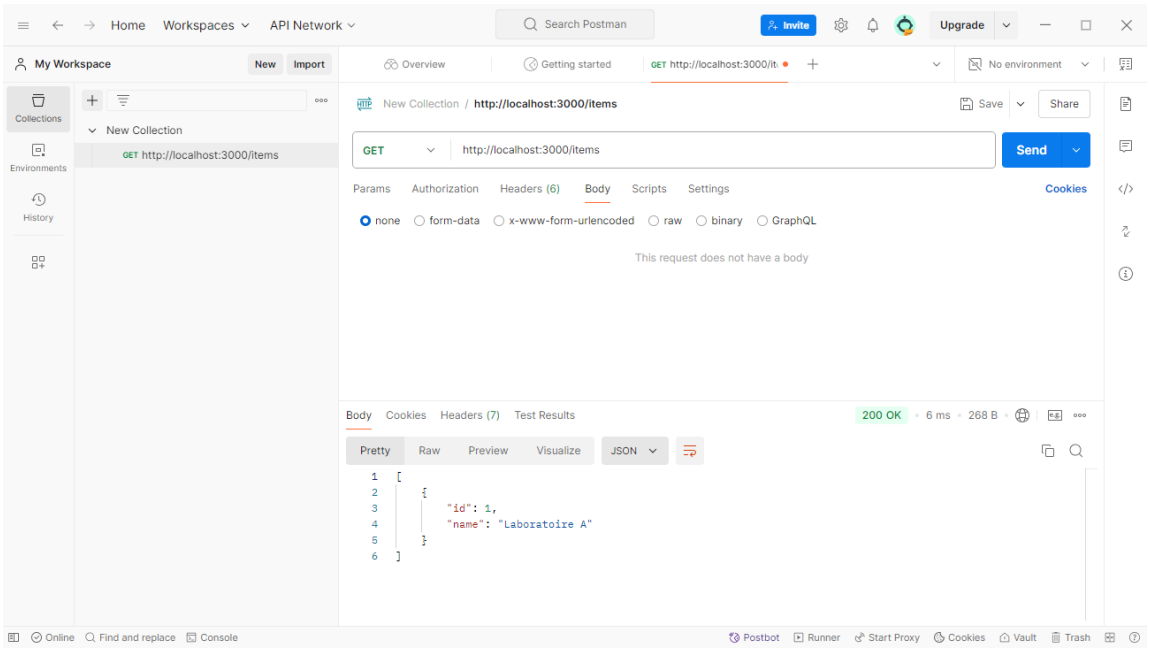
4.9 11. Tester les endpoints avec Postman

Utilisez **Postman** pour tester les différentes méthodes HTTP :

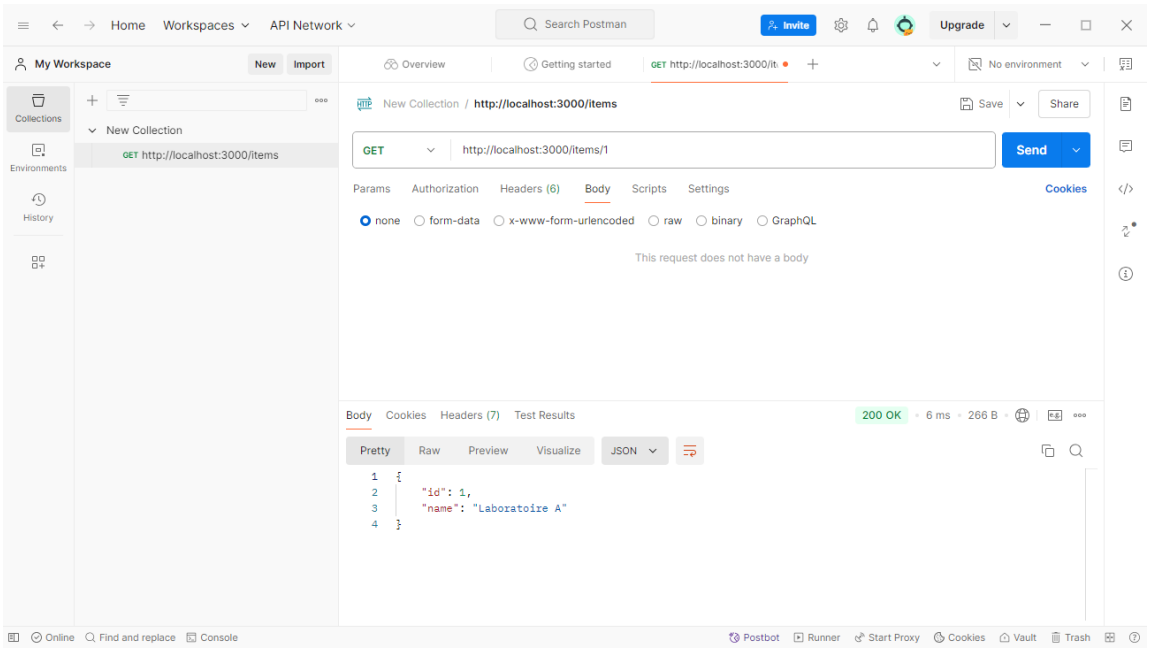
- **POST** : Ajoutez un item avec un corps JSON à `http://localhost:3000/items`.



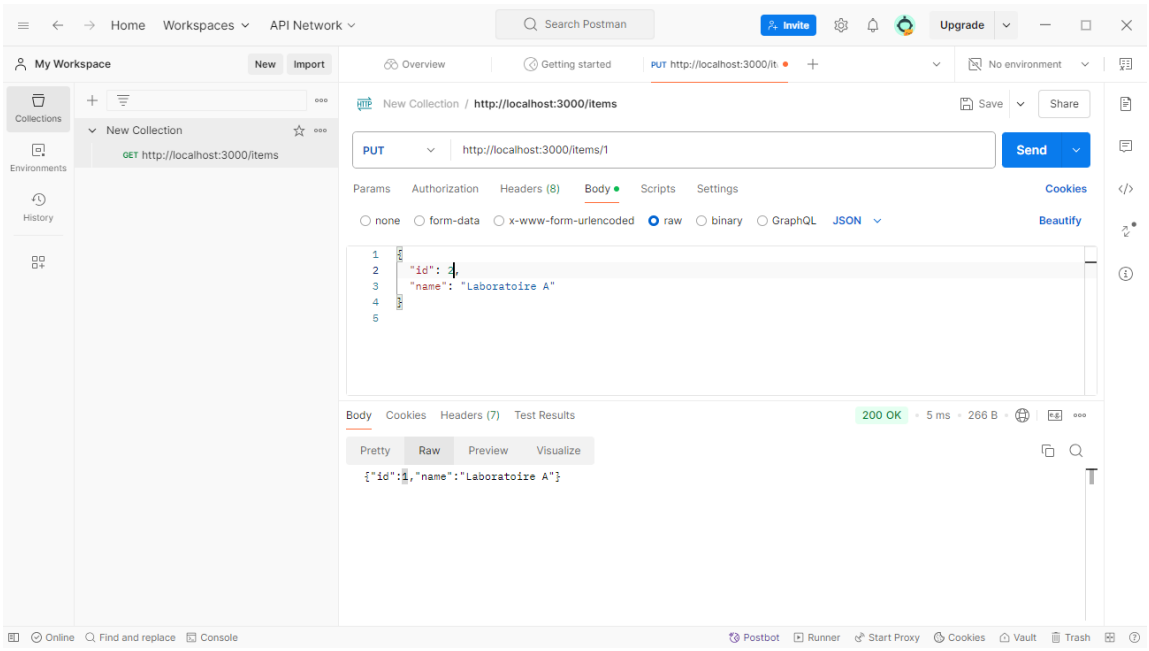
- **GET** : Récupérez tous les items à `http://localhost:3000/items`.

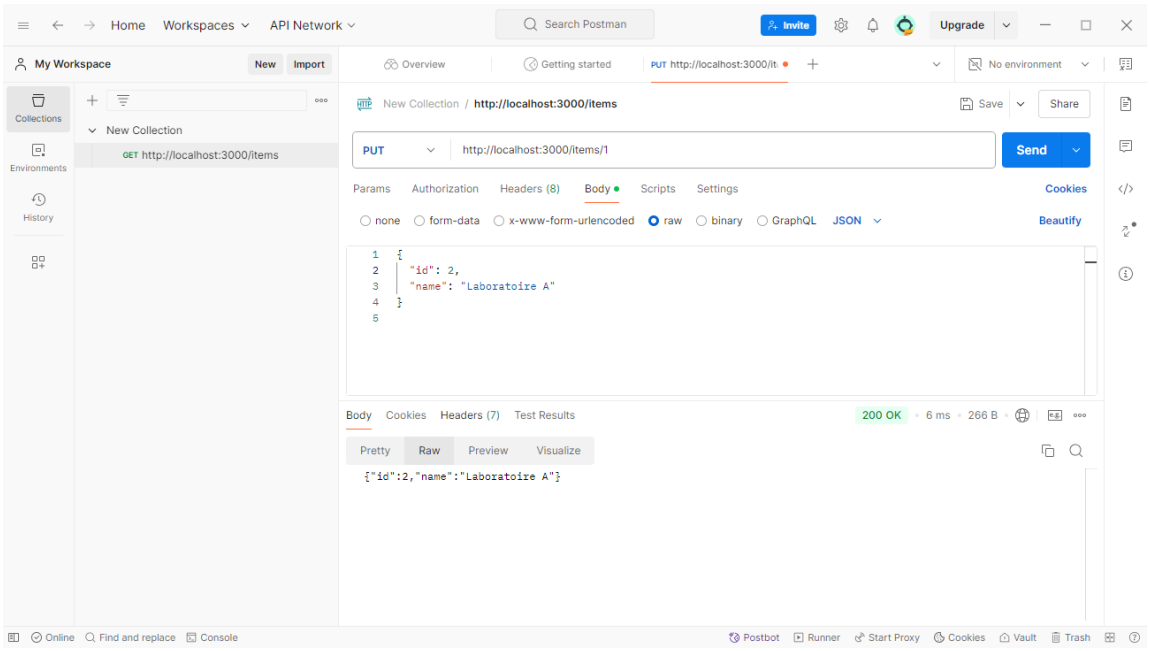


- **GET (par ID)** : Récupérez un item spécifique à `http://localhost:3000/items/1`.

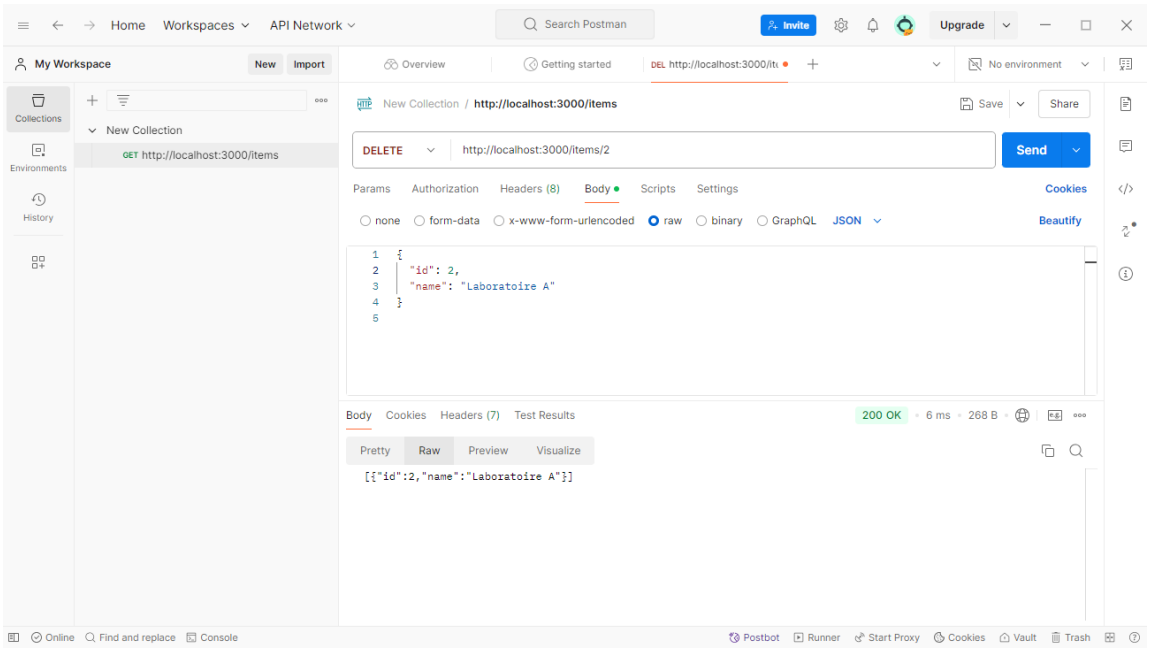


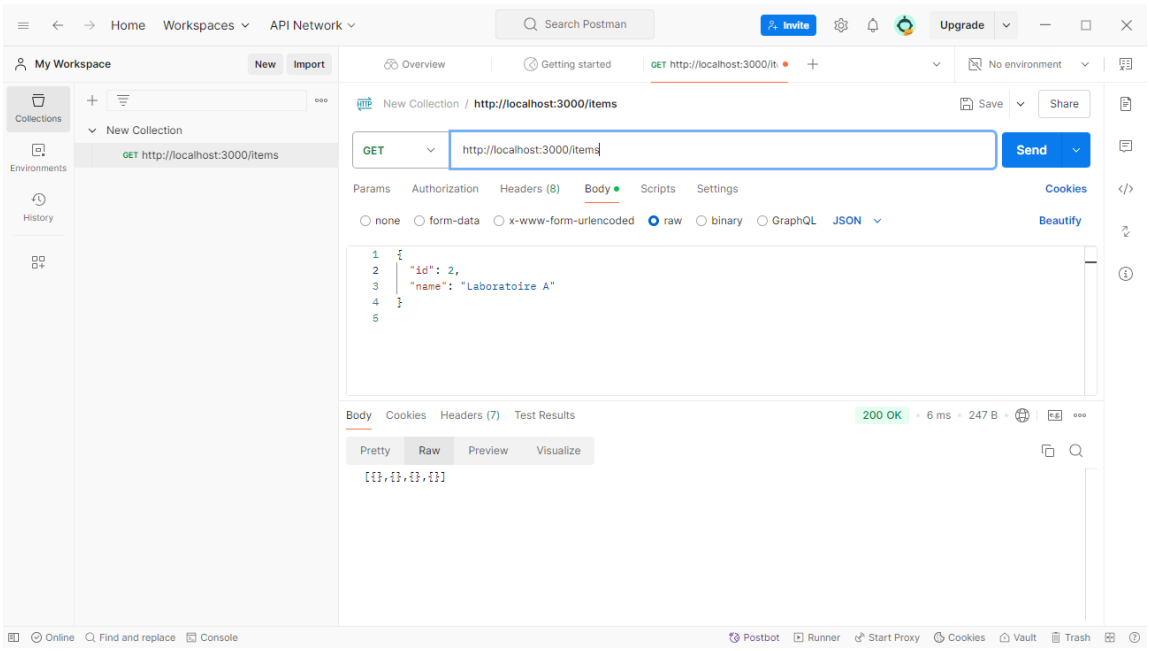
- **PUT** : Mettez à jour un item à `http://localhost:3000/items/1`.





- **DELETE** : Supprimez un item à `http://localhost:3000/items/1`.


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
Getting started
DEL http://localhost:3000/it
No environment
Collections
Environments
History
New Collection
GET http://localhost:3000/items
New Collection / http://localhost:3000/items
Save
Share
DELETE
http://localhost:3000/items/2
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id": 2,
"name": "Laboratoire A"
}
Body
Cookies
Headers (7)
Test Results
200 OK
6 ms
268 B
Pretty
Raw
Preview
Visualize
[{"id":2,"name":"Laboratoire A"}]
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
Getting started
GET http://localhost:3000/it
No environment
Collections
Environments
History
New Collection
GET http://localhost:3000/items
New Collection / http://localhost:3000/items
Save
Share
GET
http://localhost:3000/items
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id": 2,
"name": "Laboratoire A"
}
Body
Cookies
Headers (7)
Test Results
200 OK
6 ms
247 B
Pretty
Raw
Preview
Visualize
[{},{},{},{}]
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

5 Conclusion

Ce TP a permis de découvrir les bases d'Express.js à travers la création d'une application CRUD. Nous avons appris à configurer un serveur, à gérer les requêtes HTTP et à manipuler des données de manière dynamique. Les middlewares ont facilité la gestion des flux de requêtes et le traitement des erreurs.