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:

- Zougarh zaynab

Introduction

Ce TP vise à introduire le développement d'applications web en utilisant Express.js, un framework léger pour Node.js. L'objectif est de créer un serveur capable de gérer des requêtes HTTP et de réaliser des opérations de création, lecture, mise à jour et suppression (CRUD) sur des données. En utilisant des middlewares, tels que body-parser, pour traiter le corps des requêtes, ce TP permet d'enrichir les fonctionnalités de l'application et de faciliter la gestion des données. Pour valider le bon fonctionnement des différentes opérations, Postman a été utilisé pour effectuer des tests des endpoints.

1.Qu'est-ce qu'Express.js et que peut-on en faire ?

Express.js est un framework web minimaliste et flexible pour Node.js, conçu pour simplifier le développement d'applications web et d'API. Il fournit un ensemble de fonctionnalités robustes pour la création de serveurs web, facilitant ainsi la gestion des requêtes et des réponses HTTP. Grâce à sa simplicité et à sa légèreté, Express.js permet aux développeurs de construire rapidement des applications web performantes.

Avec Express.js, vous pouvez :

- Créer des applications web : Que ce soit des sites statiques ou dynamiques, Express.js vous permet de gérer les routages, les templates et les sessions utilisateur.
- Développer des API RESTful : Vous pouvez facilement construire des API pour vos applications front-end, permettant aux clients d'interagir avec des bases de données et d'autres services.
- Gérer des middleware : Express.js permet l'utilisation de middleware, des fonctions intermédiaires qui peuvent traiter les requêtes avant qu'elles n'atteignent les routes définies, ce qui est utile pour la gestion des authentifications, la validation des données, et le traitement des erreurs.
- Intégrer des bases de données : Grâce à des bibliothèques comme Mongoose ou Sequelize, vous pouvez facilement connecter votre application à des bases de données comme MongoDB ou MySQL.

2.Qu'est-ce que les middleware et comment sont-ils utilisés dans Express.js ?

Les middleware dans Express.js sont des fonctions qui interviennent dans le cycle de traitement des requêtes HTTP. Ils sont exécutés dans l'ordre où ils sont définis dans le code, permettant ainsi d'effectuer des opérations sur les requêtes et les réponses, ou d'arrêter le traitement de la requête si nécessaire. Les middleware sont essentiels pour gérer des tâches courantes telles que l'authentification, la gestion des erreurs, le traitement des données,

- Middleware pour le traitement du corps des requêtes (**body-parser)** : Le middleware body-parser est utilisé pour analyser le corps des requêtes HTTP et le rendre accessible via req.body. Cela est particulièrement utile pour les requêtes POST où des données sont envoyées au serveur.

```
// Middleware pour analyser le corps des requêtes en JSON
app.use(bodyParser.json());
```

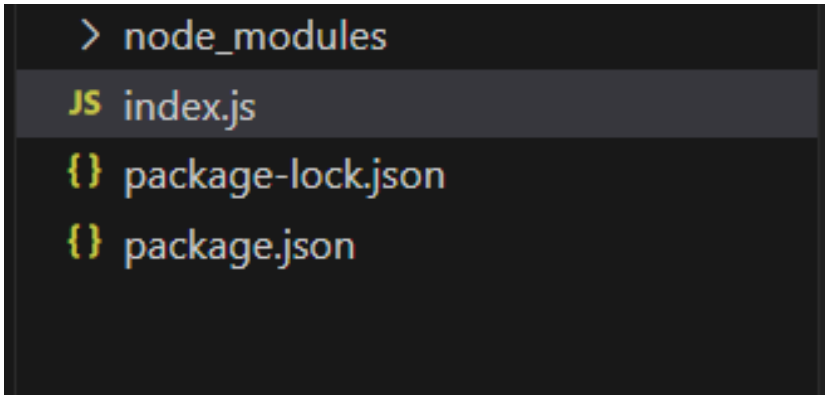
- **Middleware personnalisé pour logger les requêtes**:Ce code définit un middleware personnalisé dans une application Express.js qui enregistre les détails de chaque requête HTTP reçue par le serveur.

```
// Middleware personnalisé pour logger les requêtes
app.use((req, res, next) => {
    console.log(`${req.method} ${req.url}`); // Affiche la méthode et l'URL de la requête
    next(); // Passe à la prochaine fonction middleware
});
```

```
Server is running on http://localhost:3000
GET /items
```

2. Installation d'Express

```
PS C:\Users\vPro\Desktop\TPExpress> npm init -y
>>
Wrote to C:\Users\vPro\Desktop\TPExpress\package.json:

{
  "name": "tpexpress",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}



PS C:\Users\vPro\Desktop\TPExpress> npm i express
>>

added 65 packages, and audited 66 packages in 4s

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

found 0 vulnerabilities
```



3. Créer un Endpoint POST

Dans ce code, nous avons un tableau local items qui sert à stocker les éléments ajoutés. L'endpoint POST /items permet d'ajouter de nouveaux éléments à ce tableau.

```
// Variable pour stocker les éléments
let items = [];

// Endpoint pour ajouter un nouvel élément (POST)
app.post('/items', (req, res) => {
  const newItem = req.body;

  // Vérification de la présence d'un id et d'un name
  if (!newItem || !newItem.id || !newItem.name) {
    return res.status(400).json({ message: 'Bad Request: Missing id or name' }); // 400: Bad Request
  }

  items.push(newItem);
  res.status(201).json(newItem); // 201: Created
});
```

4. Créer un Endpoint GET

L'endpoint GET permet de récupérer tous les éléments stockés dans le tableau items.

```
// Endpoint pour récupérer tous les éléments (GET)
app.get('/items', (req, res) => {
  res.json(items); // Retourner tous les éléments
});

// Endpoint pour récupérer un élément par ID (GET)
app.get('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id, 10); // Convertir l'ID en entier
  const item = items.find(i => i.id === itemId);

  if (!item) {
    return res.status(404).json({ message: 'Item not found' }); // 404: Not Found
  }

  res.json(item); // Retourner l'élément trouvé
});
```

5. Créer un Endpoint GET(id)

Cet endpoint permet de récupérer un élément spécifique à partir de son identifiant.

```
// Endpoint pour récupérer un élément par ID (GET)
app.get('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id, 10); // Convertir l'ID en entier
  const item = items.find(i => i.id === itemId);

  if (!item) {
    return res.status(404).json({ message: 'Item not found' }); // 404: Not Found
  }

  res.json(item); // Retourner l'élément trouvé
});
```

6. Créer un Endpoint PUT

Cet endpoint permet de mettre à jour un élément existant en fonction de son identifiant.

```
// Endpoint pour mettre à jour un élément existant (PUT)
app.put('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id, 10);
  const itemIndex = items.findIndex(i => i.id === itemId);

  if (itemIndex === -1) {
    return res.status(404).json({ message: 'Item not found' }); // 404: Not Found
  }

  const updatedItem = { ...items[itemIndex], ...req.body }; // Fusionner les données existantes avec les nouvelles
  items[itemIndex] = updatedItem; // Mettre à jour l'élément
  res.json(updatedItem); // Retourner l'élément mis à jour
});
```

7. Créer un Endpoint DELETE

Cet endpoint permet de supprimer un élément existant en fonction de son identifiant.

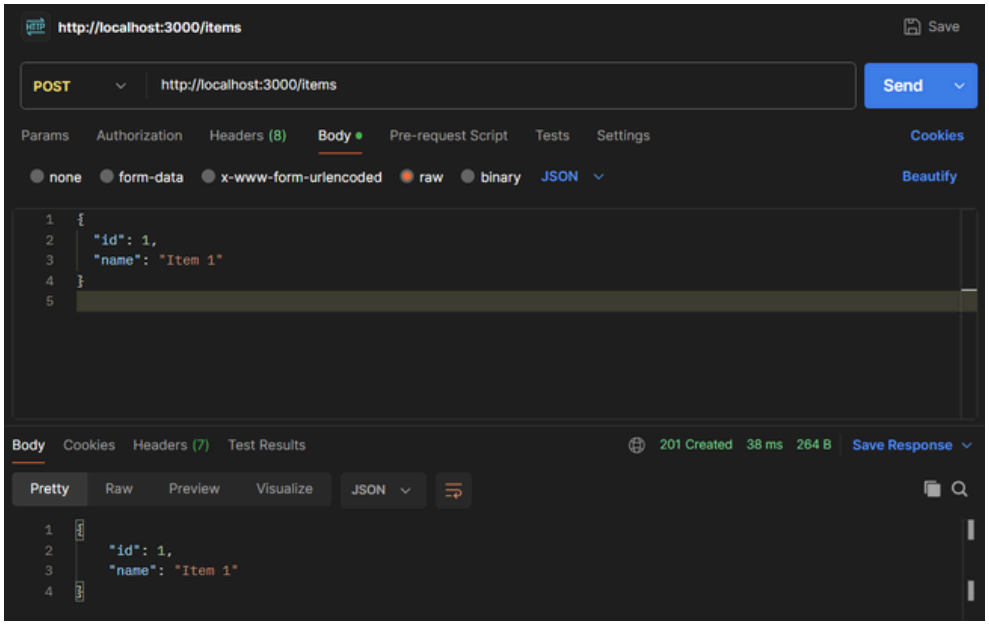
```
// Endpoint pour supprimer un élément (DELETE)
app.delete('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id, 10);
  const itemIndex = items.findIndex(i => i.id === itemId);

  if (itemIndex === -1) {
    return res.status(404).json({ message: 'Item not found' }); // 404: Not Found
  }

  items.splice(itemIndex, 1); // Supprimer l'élément du tableau
  res.status(204).send(); // 204: No Content
});
```

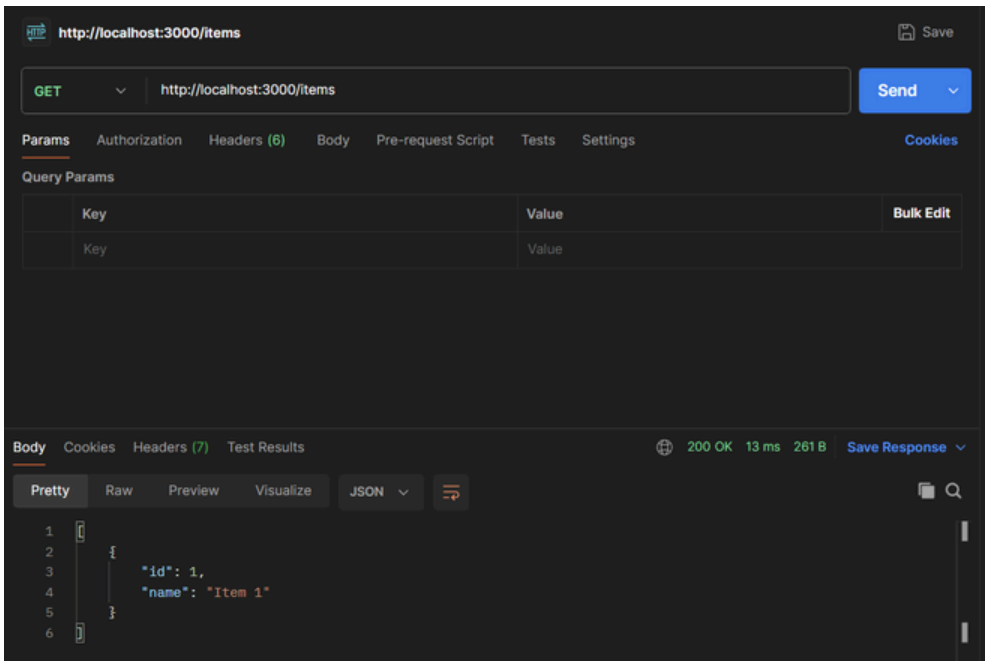
8. Démarrer le Serveur

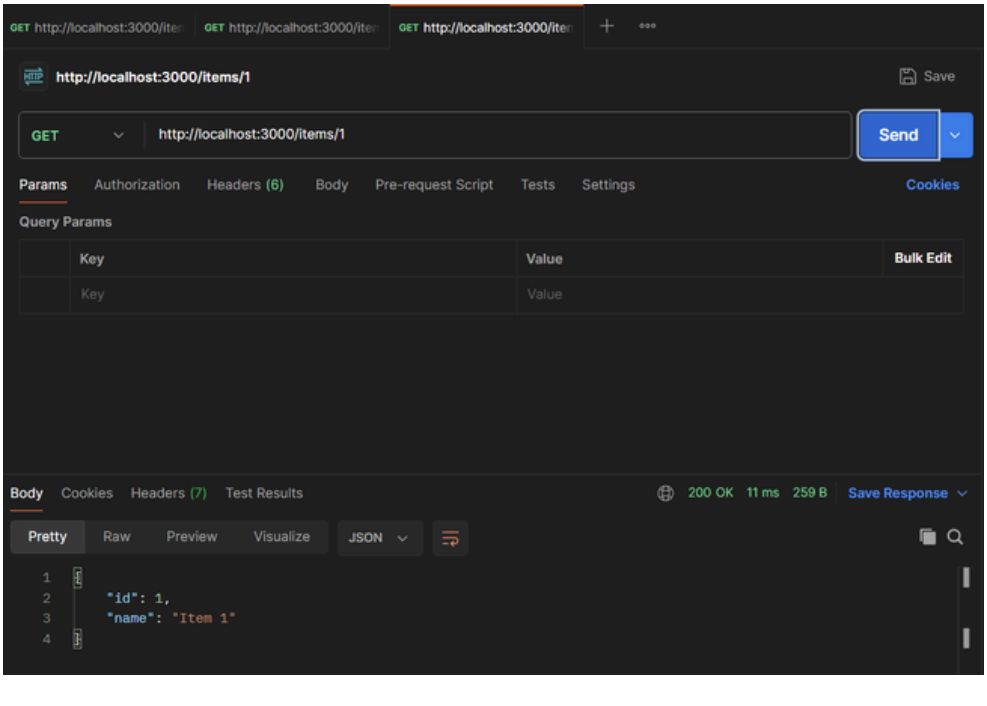
```
// Démarrer le serveur
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

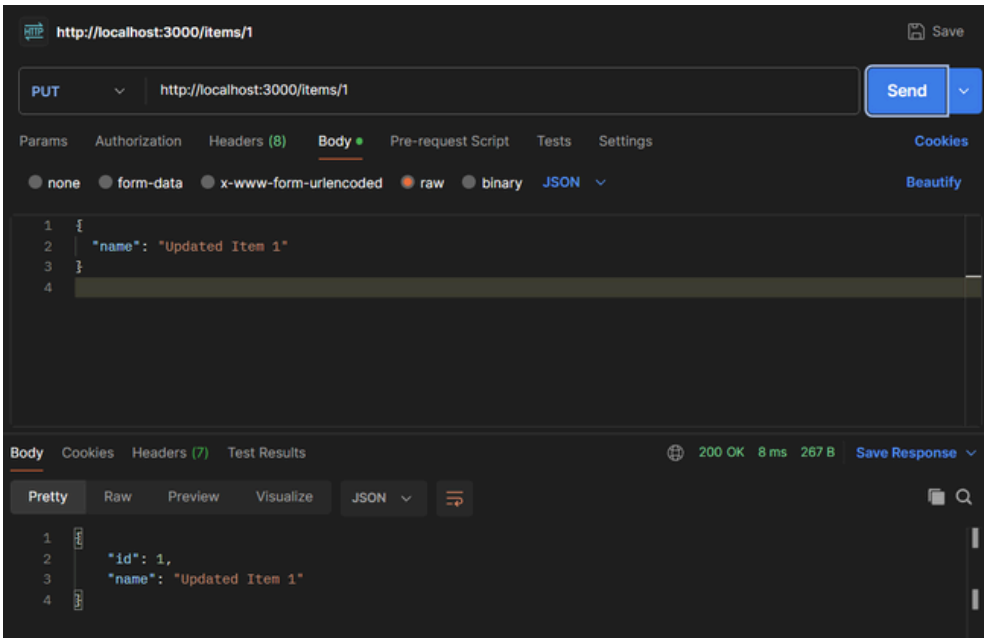
```
PS C:\Users\vPro\Desktop\TPExpress> node index.js
>>
Server is running on http://localhost:3000
```

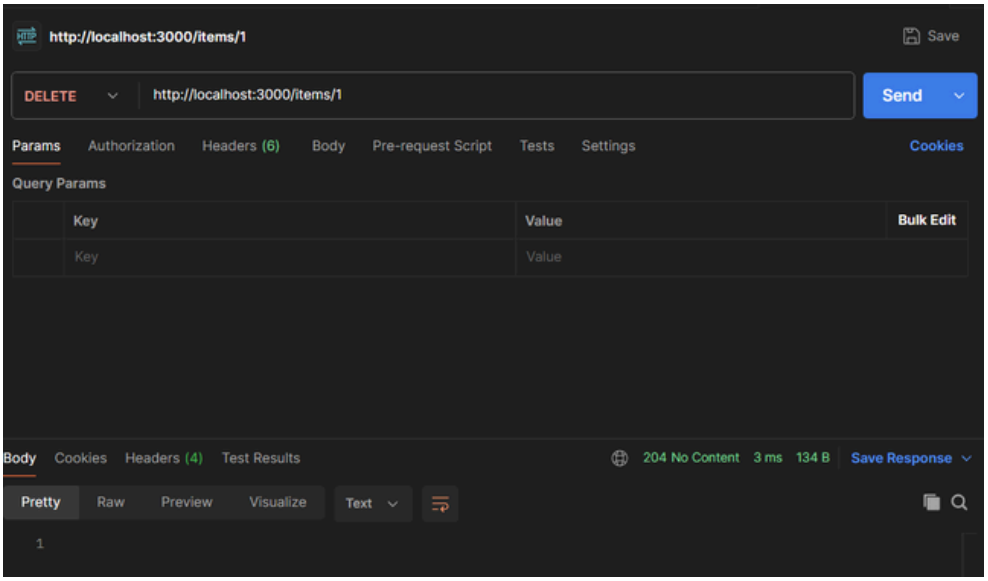
9. Tester les Endpoints avec Postman

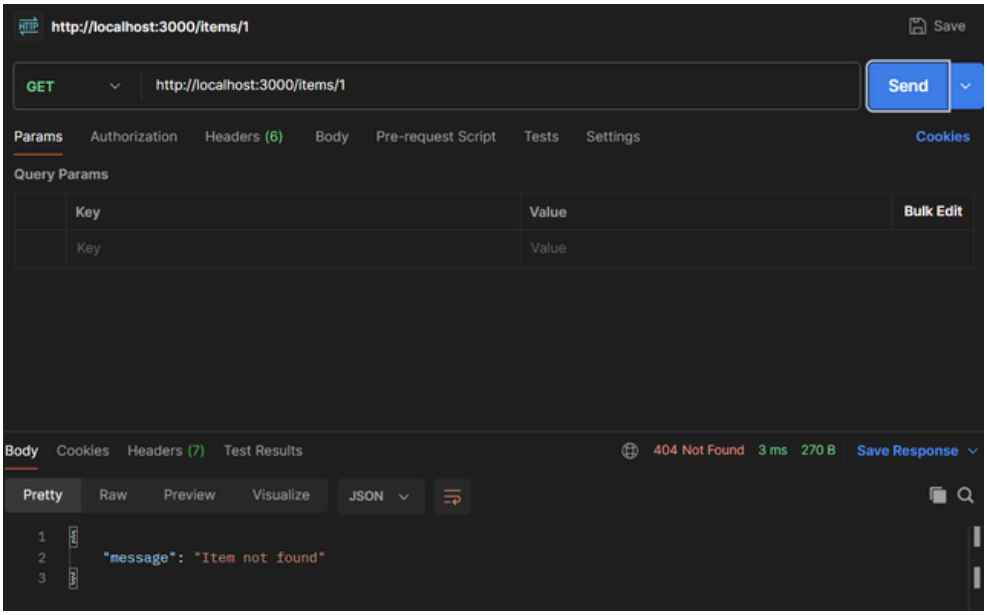

http://localhost:3000/items
Save
POST
http://localhost:3000/items
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
JSON
Beautify
{
"id": 1,
"name": "Item 1"
}
Body
Cookies
Headers (7)
Test Results
201 Created
38 ms
264 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"name": "Item 1"
}


http://localhost:3000/items
Save
GET
http://localhost:3000/items
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Bulk Edit
Key
Value
Body
Cookies
Headers (7)
Test Results
200 OK
13 ms
261 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"name": "Item 1"
}
]


GET http://localhost:3000/iter
GET http://localhost:3000/iter
GET http://localhost:3000/iter
http://localhost:3000/items/1
Save
GET
http://localhost:3000/items/1
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Bulk Edit
Key
Value
Body
Cookies
Headers (7)
Test Results
200 OK
11 ms
259 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"name": "Item 1"
}


http://localhost:3000/items/1
Save
PUT
http://localhost:3000/items/1
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
JSON
Beautify
{
"name": "Updated Item 1"
}
Body
Cookies
Headers (7)
Test Results
200 OK
8 ms
267 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"name": "Updated Item 1"
}


http://localhost:3000/items/1
Save
DELETE
http://localhost:3000/items/1
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Bulk Edit
Key
Value
Body
Cookies
Headers (4)
Test Results
204 No Content
3 ms
134 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1


http://localhost:3000/items/1
Save
GET
http://localhost:3000/items/1
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Bulk Edit
Key
Value
Body
Cookies
Headers (7)
Test Results
404 Not Found
3 ms
270 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Item not found"
}

Conclusion

Ce travail pratique a permis d'explorer le développement d'applications web avec Express.js, en créant une API capable de gérer des opérations CRUD sur des données. Nous avons appris à utiliser des middlewares pour traiter les requêtes et à tester nos endpoints via Postman. À la fin de ce TP, nous avons acquis des compétences pratiques essentielles pour concevoir des applications web modernes, ouvrant la voie à des développements futurs plus avancés.