



Rapport :
TP : Express.JS

Filière : Génie Informatique (3ème année)

Professeur : Amal Ourdou

Réalisé par :
ETTALBI Mouna
ELMECHHOURI Hajar

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

Express.js est un framework web minimaliste et rapide pour Node.js qui simplifie le développement d'applications web et d'APIs. Il offre un ensemble robuste de fonctionnalités pour les applications web et mobiles, notamment :

- **Routage** : Simplifie la gestion des requêtes HTTP et des routes URL.
- **Support des middlewares** : Permet d'utiliser des fonctions middleware qui peuvent exécuter du code, modifier les objets de requête et de réponse, terminer le cycle de requête-réponse et appeler la prochaine fonction middleware dans la pile.
- **Moteurs de templates** : S'intègre avec divers moteurs de templates pour rendre des pages HTML dynamiques.
- **API RESTful** : Permet de configurer facilement des API RESTful pour les applications.
- **Serveur de fichiers statiques** : Permet de servir facilement des fichiers statiques tels que des images, des fichiers CSS et JavaScript.

Avec Express.js, vous pouvez créer des applications web simples ou des APIs RESTful complexes.

2. Qu'est-ce qu'un middleware dans Express.js ?

Un **middleware** dans Express.js fait référence à des fonctions qui s'exécutent au cours du cycle de vie d'une requête au serveur. Elles peuvent modifier l'objet de requête, l'objet de réponse, terminer le cycle de requête-réponse et appeler le middleware suivant dans la pile. Les middlewares sont essentiels pour diverses fonctionnalités telles que l'authentification, la journalisation, l'analyse des corps de requêtes et la gestion des erreurs.

Exemples de middlewares :

1. **Middleware de journalisation** : Journalise les détails de chaque requête entrante.

```
const express = require('express');
const app = express();
// Middleware de journalisation
app.use((req, res, next) => {
  console.log(`${req.method} requête pour '${req.url}'`);
  next(); // Appelle le prochain middleware
});
```

2. **Middleware d'analyse du corps** : Analyse les corps de requêtes entrantes dans un middleware avant vos gestionnaires.

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
// Middleware d'analyse du corps
app.use(bodyParser.json()); // Pour analyser application/json
app.use(bodyParser.urlencoded({ extended: true })); // Pour analyser application/x-www-form-urlencoded
```

Création d'une application CRUD simple

1. Créer un répertoire de projet

D'abord, créez un nouveau répertoire pour votre projet et naviguez à l'intérieur.

2. Initialiser un projet Node.js

Exécutez la commande suivante pour créer un fichier package.json pour votre projet.

```
npm init -y
```

3. Installer Express

Installez Express.js en utilisant npm.

```
npm install express
```

4. Configurer Express

Créez un fichier app.js pour configurer le serveur.

```
JS app.js > ...
You, 5 minutes ago | 2 authors (You and one other)
1  const express = require('express');
2  const app = express();
3  const port = 3000;
4
5  app.use(express.json()); // Middleware pour parser les données JSON
6
7  // Route de base pour vérifier que le serveur fonctionne
8  app.get('/', (req, res) => {
9    res.send('Serveur Express est opérationnel !');
10 });
11
12 // Lancer le serveur
13 app.listen(port, () => {
14   console.log(`Serveur en écoute sur http://localhost:${port}`);
15 });
16
17 let items = []; // Variable locale pour stocker les éléments
18
```

5. Créer un endpoint POST

Cet endpoint nous permet d'ajouter des éléments à une variable locale.

```
18
19 // Endpoint POST pour ajouter des éléments
20 app.post('/items', (req, res) => {
21   const item = req.body; // Récupérer l'élément depuis le corps de la requête
22   items.push(item); // Ajouter l'élément à l'array local
23   res.status(201).json(item); // Répondre avec l'élément créé
24 });
25
```

6. Créer un endpoint GET

Cet endpoint nous permet de récupérer tous les éléments

```
27 // Endpoint GET pour récupérer tous les éléments
28 app.get('/items', (req, res) => {
29   res.json(items); // Répondre avec le tableau des éléments
30 });
31
```

7. Créer un endpoint GET par ID

Cet endpoint nous permet de récupérer un élément par id.

```
32 // Endpoint GET pour récupérer un élément par ID
33 app.get('/items/:id', (req, res) => {
34   const id = parseInt(req.params.id);
35   const item = items.find(item => item.id === id);
36
37   if (!item) {
38     return res.status(404).send('Item non trouvé');
39   }
40
41   res.json(item);
42 });
43
```

8. Créer un endpoint PUT

Cet endpoint nous permet de mettre à jour un élément .

```
44 // Endpoint PUT pour mettre à jour un élément
45 app.put('/items/:id', (req, res) => {
46   const id = parseInt(req.params.id);
47   const index = items.findIndex(item => item.id === id);
48
49   if (index === -1) {
50     return res.status(404).send('Item non trouvé');
51   }
52
53   items[index] = req.body; // Remplace l'élément par les nouvelles données
54   res.send(items[index]);
55 });
56
```

9. Créer un endpoint DELETE

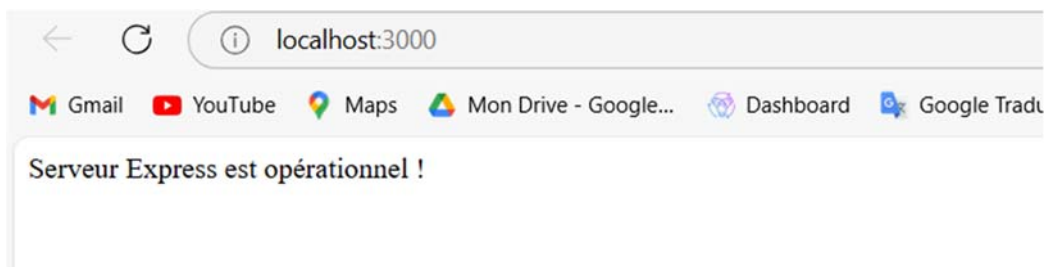
Cet endpoint nous permet de supprimer un élément .

```
57
58 // Endpoint DELETE pour supprimer un élément
59 app.delete('/items/:id', (req, res) => {
60   const id = parseInt(req.params.id, 10);
61   if (items[id]) {
62     items.splice(id, 1); // Retirer l'élément du tableau
63     res.status(204).send(); // Répondre sans contenu
64   } else {
65     res.status(404).json({ message: 'Élément non trouvé' });
66   }
67 });
68
```

10. Démarrer le serveur

Dans votre terminal, assurez-vous d'être dans le répertoire du projet et exécutez le serveur.

```
PS C:\Users\LENOVO\Desktop\GI3\JS\tp_express> node app.js
Serveur en écoute sur http://localhost:3000
```



11. Tester les endpoints avec Postman

1. **POST** /items : Ajouter un nouvel élément.

HTTP <http://localhost:3000/items>

POST <http://localhost:3000/items>

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▾

```
1 {
2   "id": 1,
3   "name": "Item1"
4 }
5
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON ▾

```
1 {
2   "id": 1,
3   "name": "Item1"
4 }
```

2. **GET** /items : Récupérer tous les éléments.

GET <http://localhost:3000/items>

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▾

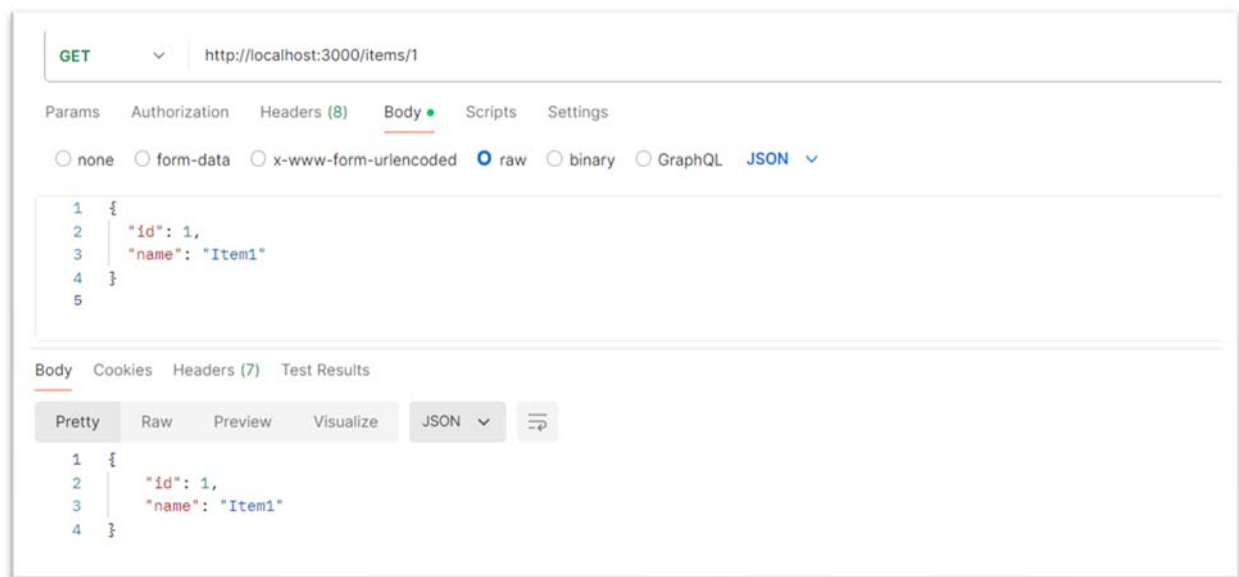
```
1 {
2   "id": 1,
3   "name": "Item1"
4 }
5
```

Body Cookies Headers (7) Test Results

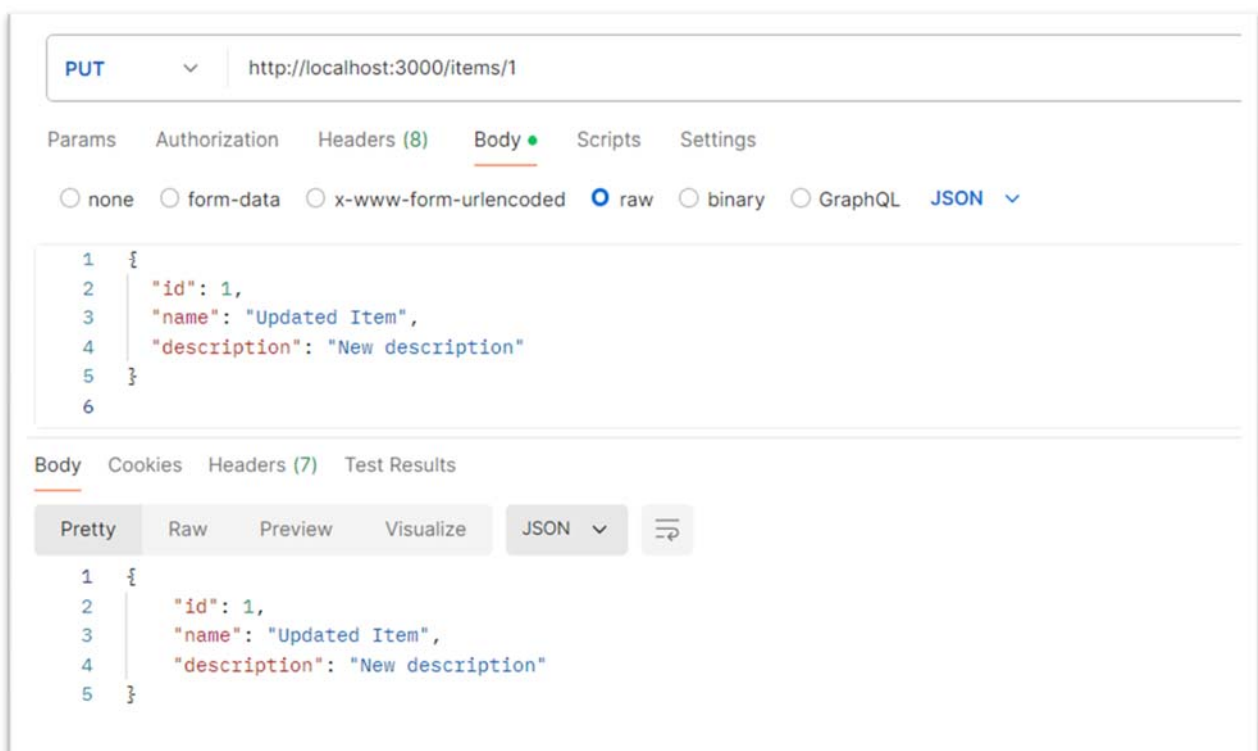
Pretty Raw Preview Visualize JSON ▾

```
1 [
2   {
3     "id": 1,
4     "name": "Item1"
5   }
6 ]
```

3. **GET** /items/1 : Récupérer le premier élément.



4. **PUT** /items/1 : Mettre à jour le premier élément.



5. **DELETE** /items/1 : Supprimer le premier élément.


DELETE  http://localhost:3000/items/1

Params Authorization Headers (8) **Body**  Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON 

```
1  {  
2    "id": 1,  
3    "name": "Updated Item",  
4    "description": "New description"  
5  }  
6
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON  

```
1  {  
2    "message": "Élément non trouvé"  
3  }
```