

Compte Rendu – TP MERN Semaine 2

Cours : Structuration de l'API : Routes et Contrôleurs

Objectif du TP

L'objectif de ce TP est de transformer une application Express simple, basée sur un seul fichier server.js, en une architecture d'API structurée et professionnelle.

On apprend ici à :

- Appliquer le principe de la **Séparation des Préoccupations (SoC)**.
- Créer des **routes** et **contrôleurs** indépendants.
- Organiser le code pour le rendre plus **clair, maintenable et évolutif**.
- Tester l'API avec **Postman**.

Partie 1 – Concepts architecturaux

-Le problème : le fichier monolithique

Le code initial contenait toutes les routes dans server.js, ce qui le rendait :

- Illisible avec de nombreuses lignes.
- Difficile à maintenir et à modifier.
- Impraticable pour le travail collaboratif.

Ce type de code est souvent appelé “**code spaghetti**”.

-La solution : la séparation des préoccupations (SoC)

Pour éviter cela, on sépare les responsabilités en trois niveaux :

1. **Serveur (server.js)** : le chef d'orchestre — il configure et connecte les modules.
 2. **Routeur (/routes)** : l'aiguilleur — il dirige les requêtes HTTP vers le bon contrôleur.
 3. **Contrôleur (/controllers)** : l'ouvrier — il contient la logique métier (traitement, création, réponse).
- L'outil clé : **express.Router()**

C'est un mini-routeur fourni par Express permettant de regrouper les routes d'un même type dans un seul fichier modulaire.

Chaque routeur est ensuite connecté au serveur principal via `app.use()`.

3. Partie 2 – Atelier pratique : refactorisation du projet

Nous avons restructuré le projet **mon-api-blog** en suivant 5 étapes principales.

Étape 1 – Préparation de l'architecture

Dans le dossier du projet, création des deux répertoires nécessaires :

`mkdir routes`

`mkdir controllers`

Le but : séparer le code selon son rôle.

Étape 2 – Création du contrôleur d'articles

Fichier : `controllers/articleController.js`

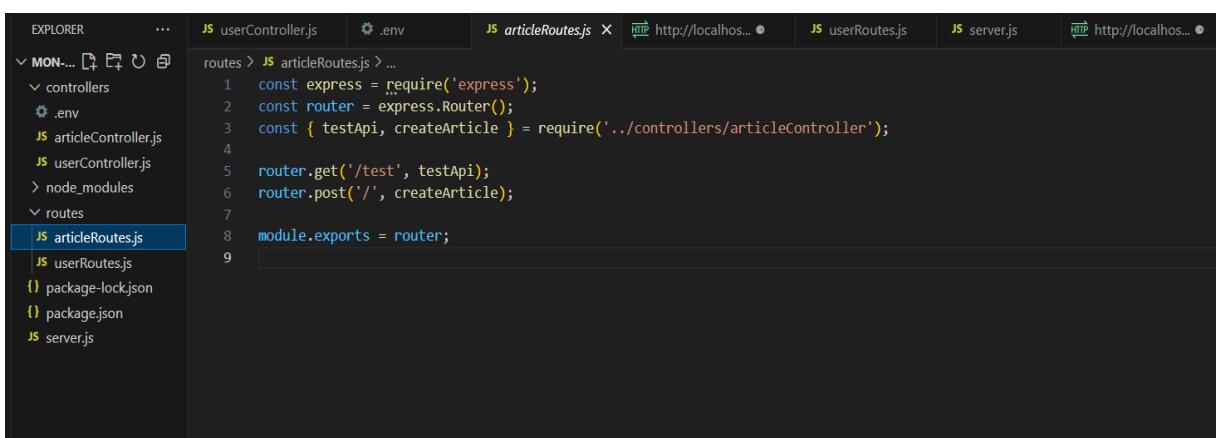


```
EXPLORER ... JS UserController.js .env JS articleController.js X HTTP http://localhost:3001 UserController.js JS userRoutes.js JS server.js HTTP http://localhost:3001

controllers > JS articleController.js > createArticle
1 const testApi = (req, res) => {
2   res.status(200).json({ message: 'Le test depuis le contrôleur a fonctionné !', success: true });
3 }
4
5 const createArticle = (req, res) => {
6   const articleData = req.body;
7   console.log('Données reçues :', articleData);
8   res.status(201).json({
9     message: 'Article créé avec succès via le contrôleur !',
10    article: { id: Date.now(), ...articleData }
11  });
12 }
13
14 module.exports = { testApi, createArticle };
15
```

Étape 3 – Création du routeur d'articles

Fichier : `routes/articleRoutes.js`

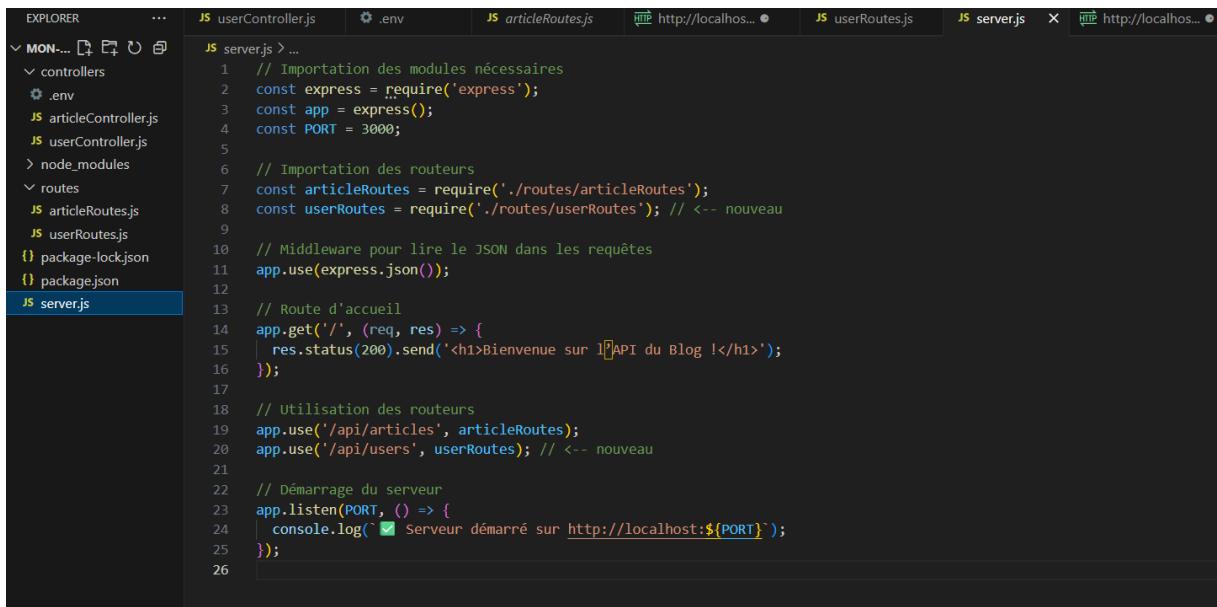


```
EXPLORER ... JS UserController.js .env JS articleRoutes.js X HTTP http://localhost:3001 UserController.js JS userRoutes.js JS server.js HTTP http://localhost:3001

routes > JS articleRoutes.js > ...
1 const express = require('express');
2 const router = express.Router();
3 const { testApi, createArticle } = require('../controllers/articleController');
4
5 router.get('/test', testApi);
6 router.post('/', createArticle);
7
8 module.exports = router;
9
```

Étape 4 – Mise à jour du serveur principal

Fichier : server.js



```
EXPLORER ... JS userController.js | .env | JS articleRoutes.js | HTTP http://localhost:3000 JS userRoutes.js | JS server.js X | HTTP http://localhost:3000

JS server.js > ...
1 // Importation des modules nécessaires
2 const express = require('express');
3 const app = express();
4 const PORT = 3000;
5
6 // Importation des routeurs
7 const articleRoutes = require('./routes/articleRoutes');
8 const userRoutes = require('./routes/userRoutes'); // <-- nouveau
9
10 // Middleware pour lire le JSON dans les requêtes
11 app.use(express.json());
12
13 // Route d'accueil
14 app.get('/', (req, res) => {
15   res.status(200).send('<h1>Bienvenue sur l'API du Blog !</h1>');
16 });
17
18 // Utilisation des routeurs
19 app.use('/api/articles', articleRoutes);
20 app.use('/api/users', userRoutes); // <-- nouveau
21
22 // Démarrage du serveur
23 app.listen(PORT, () => {
24   console.log(`Serveur démarré sur http://localhost:${PORT}`);
25 });
26
```

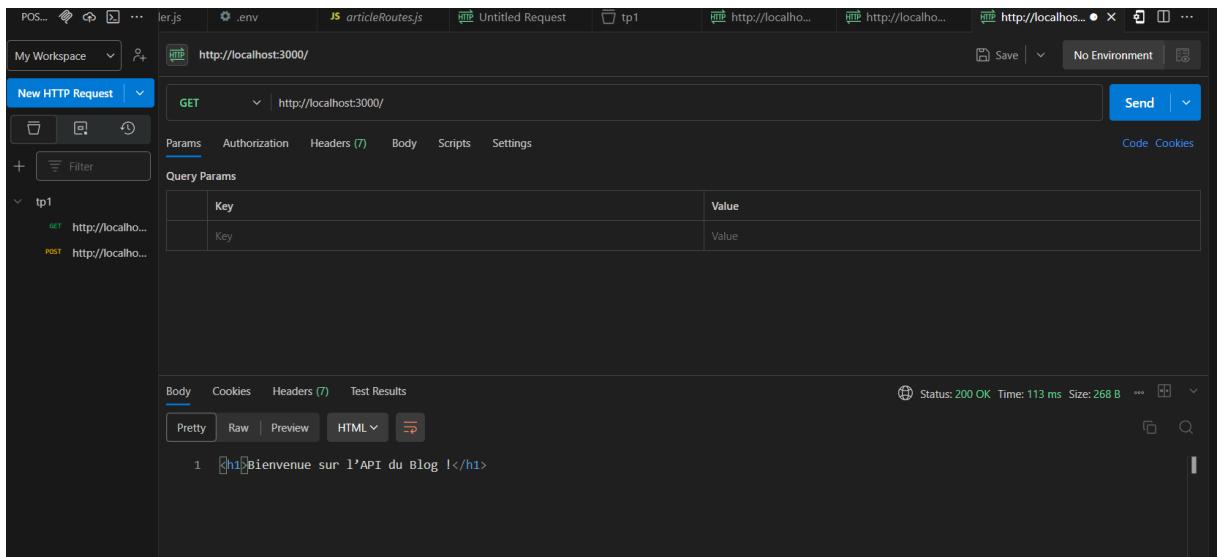
Étape 5 – Validation avec Postman

-Test 1 – Route d'accueil

GET http://localhost:3000/

→ Affiche :

<h1>Bienvenue sur l'API du Blog !</h1>



POST ... JS userController.js | .env | JS articleRoutes.js | Untitled Request | tp1 | HTTP http://localhost:3000 | ...

New HTTP Request | My Workspace | Save | No Environment | Send |

GET http://localhost:3000/

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview HTML

Status: 200 OK Time: 113 ms Size: 268 B

1 <h1> Bienvenue sur l'API du Blog !</h1>

-Test 2 – Test API d'article

GET http://localhost:3000/api/articles/test

→ Résultat :

{

"message": "Le test depuis le contrôleur a fonctionné !",

```
"success": true
```

```
}
```

The screenshot shows the Postman application interface. A new HTTP request is being created for the URL `http://localhost:3000/api/articles/test`. The method is set to GET. In the Params tab, there is a single query parameter named "Key". The response body is displayed in JSON format, showing a message and a success status. The status bar at the bottom indicates a 200 OK status with a time of 10 ms.

```
1  [{"message": "Le test depuis le contrôleur a fonctionné !", "success": true}]
```

Test 3 – Création d'un article

POST `http://localhost:3000/api/articles`

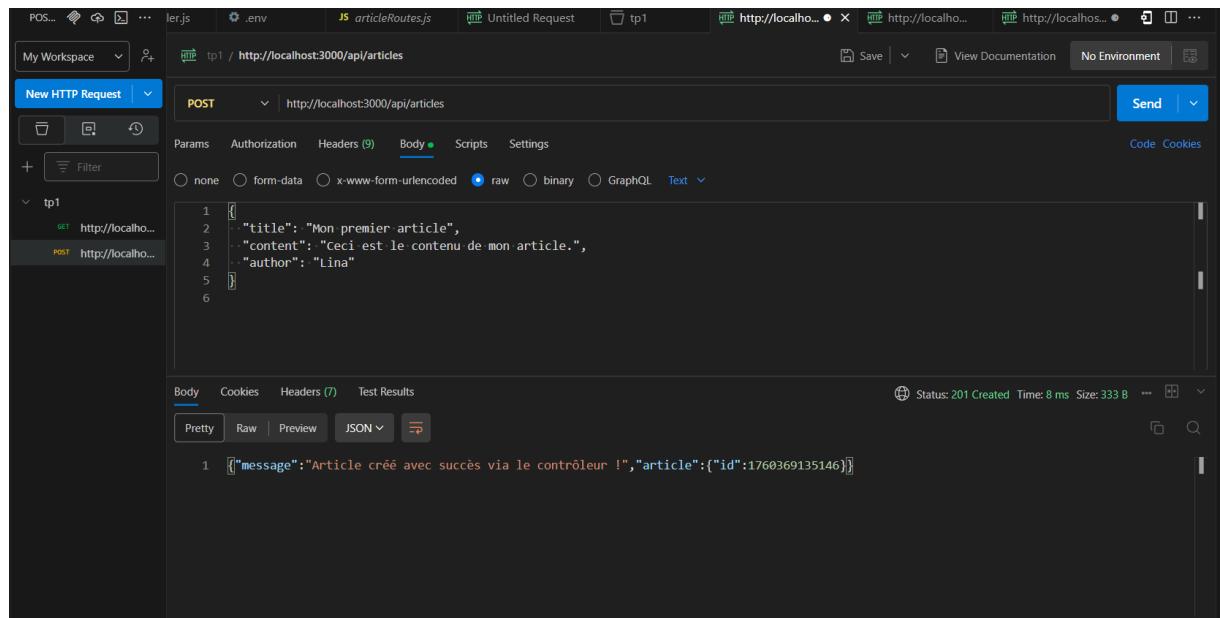
Body (JSON) :

```
{  
  "titre": "Premier article",  
  "contenu": "Exemple d'article créé via Postman"
```

```
}
```

→ Résultat :

```
{  
  "message": "Article créé avec succès via le contrôleur !",  
  "article": { "id": 1728293492000, "titre": "Premier article", "contenu": "Exemple  
d'article créé via Postman" }  
}
```



4. Travail pratique complémentaire – Gestion des utilisateurs

Nous avons ajouté une **nouvelle ressource** : “**users**”, en appliquant la même logique de structuration.

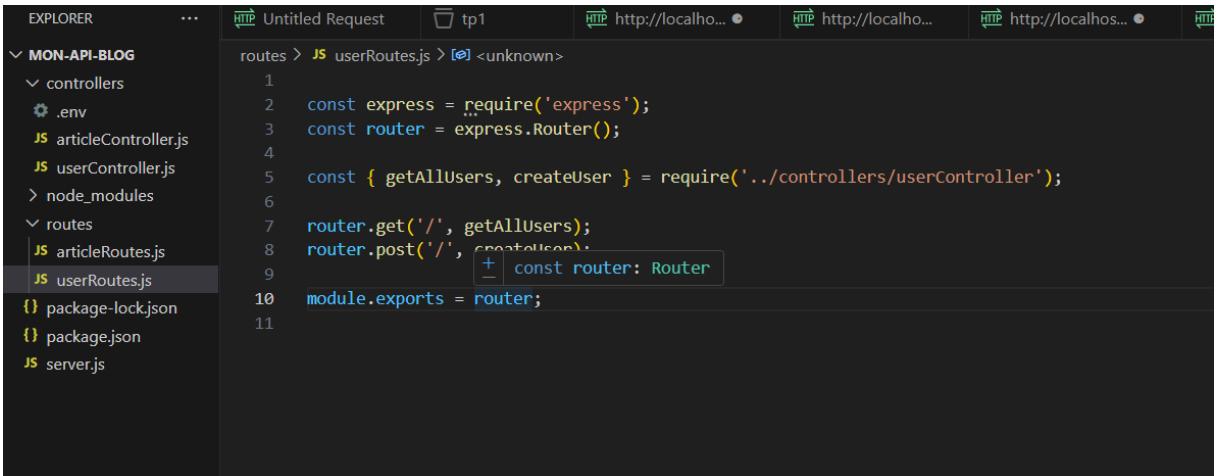
-Fichier : controllers/userController.js

```

1  const getAllUsers = (req, res) => {
2    const users = [
3      { id: 1, nom: 'Alice', email: 'alice@example.com' },
4      { id: 2, nom: 'Bob', email: 'bob@example.com' },
5      { id: 3, nom: 'Charlie', email: 'charlie@example.com' }
6    ];
7
8    res.status(200).json({
9      message: 'Liste des utilisateurs récupérée avec succès',
10     users
11   });
12 };
13
14
15
16 const createUser = (req, res) => {
17   const userData = req.body;
18   console.log('Données reçues pour création :', userData);
19
20   res.status(201).json({
21     message: 'Utilisateur créé avec succès',
22     user: { id: Date.now(), ...userData }
23   });
24 };
25
26
27 module.exports = {
28   getAllUsers,
29   createUser
30 };
31

```

Fichier : routes/userRoutes.js



```
routes > JS userRoutes.js > [o] <unknown>
1 2  const express = require('express');
3  const router = express.Router();
4
5  const { getAllUsers, createUser } = require('../controllers/userController');
6
7  router.get('/', getAllUsers);
8  router.post('/', createUser);
9
10 module.exports = router;
11
```

Test 3 – Création d'un article

Méthode : POST

URL : <http://localhost:3000/api/articles>

Body (JSON) :

```
{  
  "titre": "Premier article",  
  "contenu": "Exemple d'article créé via Postman"  
}
```

→ **Résultat :**

```
{  
  "message": "Article créé avec succès via le contrôleur !",  
  "article": {  
    "id": 1728293492000,  
    "titre": "Premier article",  
    "contenu": "Exemple d'article créé via Postman"  
  }  
}
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' and a 'New HTTP Request' button. The main area has a header 'http://localhost:3000/api/articles'. Below it, a 'POST' request is selected with the URL 'http://localhost:3000/api/articles'. The 'Body' tab is active, showing raw JSON input:

```
1 [ {  
2   ... "titre": "Premier article",  
3   ... "contenu": "Exemple d'article créé via Postman"  
4 } ]
```

Below the body, the response status is shown as 'Status: 201 Created Time: 9 ms Size: 411 B'. The response body is displayed in JSON format:

```
1 [ {"message": "Article créé avec succès via le contrôleur !", "article": {"id": 1760370630406, "titre": "Premier article", "contenu": "Exemple d'article créé via Postman"} } ]
```