TP 2: Création d'un serveur Web avec NodeJS

Dr Mad généré par une IA

Objectifs

- Premiers pas avec ExpressJS
- Gérer les requêtes HTTP des clients

Préparation du projet

Pour ce TP2, nous continuerons à travailler sur le même répertoire que celui que vous avez utilisé pour votre TP1. Assurez-vous de conserver la structure existante du projet. Dans ce TP, nous allons intégrer un serveur Web à notre application NodeJS existante. Le serveur Web sera responsable de la gestion des requêtes HTTP et répondra en conséquence.

La structure du répertoire devrait ressembler à ceci :

```
[nom_du_projet]/
├── data/
├── inputs/
├── models/
├── drmad.js
└── package.json
```

Avant de commencer, assurez-vous que votre environnement de développement est bien configuré et que toutes les dépendances nécessaires sont installées. Si nécessaire, exécutez la commande ***npm install*** pour installer les dépendances manquantes.

Installer express :

npm install express

créer un fichier **server.js** :

touch server.js

Création du Serveur Express

Dans cette étape du TP, vous êtes chargés de mettre en place un serveur avec Express qui sera en écoute sur le port 3000.

Étapes à suivre :

- Ouvrez le fichier server.js
    - Vous avez déjà créé un fichier server.js. Ouvrez ce fichier pour y ajouter du code.
- Importez le module Express
    - Assurez-vous que le module Express est bien installé. Si ce n'est pas le cas, utilisez npm pour l'installer.
- Créez une instance d’Express
    - Initiez votre application Express et assignez-la à une variable.
- Définissez le port d'écoute
    - Assurez-vous que votre serveur écoute sur le port 3000. Le numéro du port doit être stocké dans une variable pour une meilleure gestion.

- Créez une route test
    - Créez une route de base qui renverra une réponse simple. Cette étape est cruciale pour tester si votre serveur fonctionne correctement.
- Démarrez le serveur
    - Utilisez la méthode appropriée pour faire écouter votre serveur sur le port que vous avez défini précédemment. Assurez-vous également d'afficher un message dans la console pour confirmer que le serveur est opérationnel.
- Testez votre serveur
    - Utilisez un navigateur ou un outil de test d'API pour vérifier que votre serveur est en cours d'exécution et renvoie la réponse attendue lorsque vous accédez à la route test.

Utilisation du fichier models/items.js pour Récupérer les Données

Dans la continuité de votre progression, il est impératif de mettre l'accent sur l'organisation et la structuration de votre code. Vous avez déjà travaillé sur le fichier **models/items.js** durant le TP1. Pour ce TP2, nous allons intégrer ce module dans la création de la nouvelle route de votre serveur Express.

Étapes à suivre :

- Réutilisation du Module items.js :
    - Revisitez le fichier **models/items.js** que vous avez élaboré pendant le TP1. Ce module, qui contient déjà des fonctions pour lire et manipuler les données de **data.json**, sera essentiel dans cette étape.
- Intégration avec Express :
    - Intégrez les fonctions du module **models/items.js** dans votre serveur Express. Assurez-vous que la nouvelle route utilise ces fonctions pour récupérer les données de **data.json**.
- Création de la Route :
    - La nouvelle route que vous créerez devrait faire appel aux fonctions contenues dans **models/items.js** pour récupérer et formater les données avant de les envoyer en tant que réponse JSON.
- Test de l'intégration :
    - Avec l'intégration complète, testez la route pour vous assurer que la récupération et l'envoi des données sont effectués efficacement et que le format JSON de la réponse est conforme aux attentes.

Quel point de terminaison de routage (routing endpoint) avez-vous choisi ? Quelle méthode HTTP cette route gère-t-elle ?


```
[
  {
    "_id": "646cee044f07a537bf388bcb",
    "links": [
      "https://fr.wikipedia.org/wiki/Variole"
    ],
    "name": "variole",
    "description": "variole",
    "stock": 0,
    "wait": {
      "$date": "2023-05-23T16:47:00.354Z"
    },
    "sold": false,
    "price": 75000,
    "promotion": [],
    "object": {
      "code": "ACBCBBACADDCADDBCDDCABBBCADDCADDBCD"
    }
  },
  {
    "_id": "646cee044f07a537bf388bc9",
    "links": [
      "https://fr.wikipedia.org/wiki/Prion_(prot%C3%A9ine)"
    ],
    "name": "prion",
    "description": "prion",
    "stock": 1,
    "wait": {
      "$date": "2023-05-23T16:47:00.353Z"
    },
    "sold": true,
    "price": 10000,
    "promotion": [
      {
        "_id": "646cee044f07a537bf388bca",
        "discount": 10,
        "amount": 2
      }
    ],
    "object": {
      "code": "ABCACACABCAB"
    }
  },
  { … }, // 10 items
  { … }, // 10 items
  { … }, // 10 items
  { … } // 10 items
]
```


Un exemple de ce que la route API créée doit envoyer en réponse

Ajout d'une Fonctionnalité pour Récupérer le Nom et le Code des Virus

Méthode async dans models/items.js :

- Implémentez une nouvelle méthode async qui lira le contenu du fichier **data/data.json**.

- Utilisez await pour attendre la lecture du fichier et la conversion des données en objet JavaScript.
- Filtrez cet objet pour ne récupérer que le nom et le code de chaque virus.
- Gérez les erreurs efficacement.

Route dans server.js :

- Intégrez une nouvelle route pour servir le nom et le code des virus.
- Faites appel à la méthode async que vous avez définie dans **models/items.js.**
- Assurez une gestion d'erreur efficace pour envoyer des réponses adaptées selon les situations de succès ou d'échec.

Ajout d'une Route pour Enregistrer un Nouveau Virus

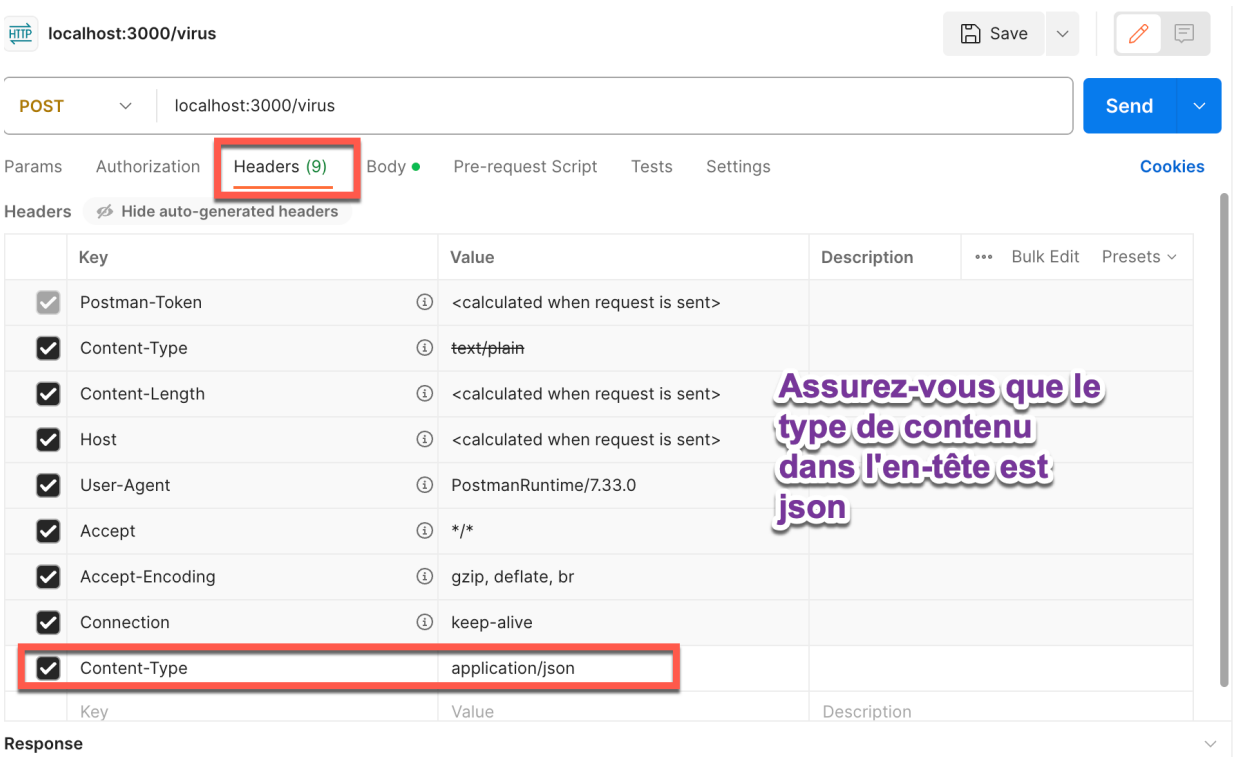
Considérez l'exemple d'objet virus ci-dessous. L'objectif est de créer une route dans votre serveur **express** qui reçoit cet objet dans le corps (**body**) d'une requête HTTP et de l'ajouter à **data/data.json**. Assurez-vous que la fonction dans **models/items.js** qui fait cela est écrite en utilisant la syntaxe **async/await**.

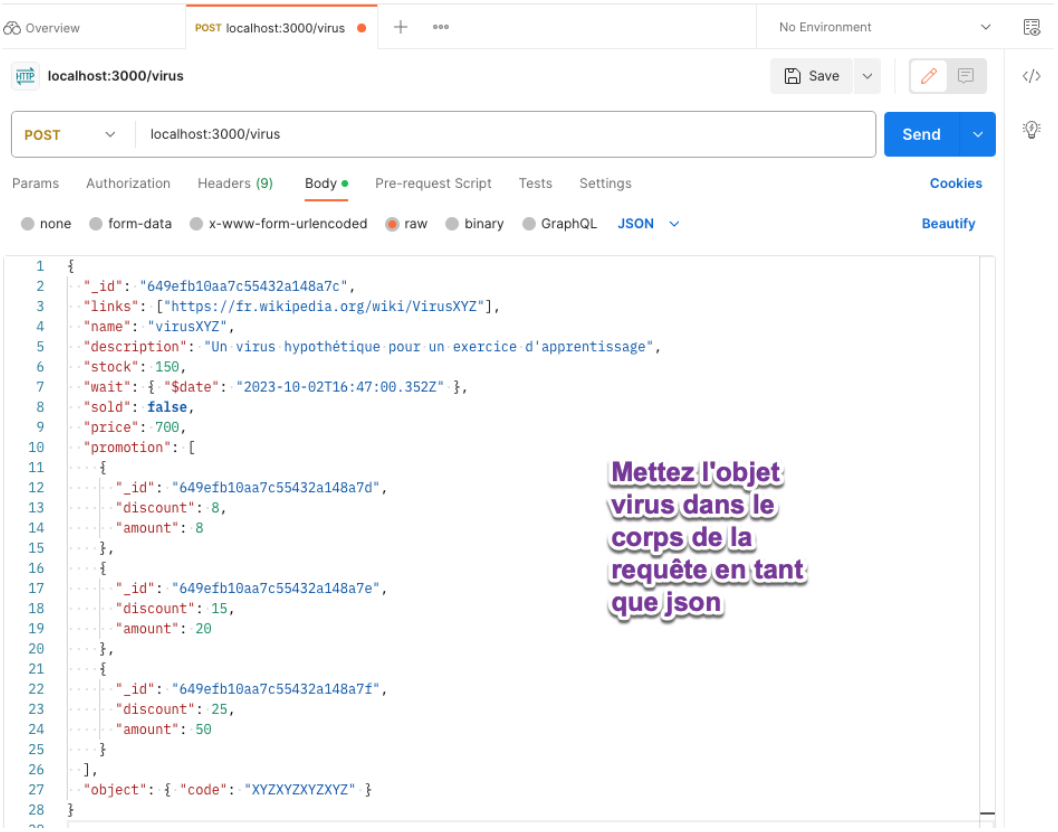
```
{
  "_id": "649efb10aa7c55432a148a7c",
  "links": ["https://fr.wikipedia.org/wiki/VirusXYZ"],
  "name": "virusXYZ",
  "description": "Un virus hypothétique pour un exercice d'apprentissage",
  "stock": 150,
  "wait": { "$date": "2023-10-02T16:47:00.352Z" },
  "sold": false,
  "price": 700,
  "promotion": [
    {
      "_id": "649efb10aa7c55432a148a7d",
      "discount": 8,
      "amount": 8
    },
    {
      "_id": "649efb10aa7c55432a148a7e",
      "discount": 15,
      "amount": 20
    },
    {
      "_id": "649efb10aa7c55432a148a7f",
      "discount": 25,
      "amount": 50
    }
  ],
  "object": { "code": "XYZXYZXYZXYZ" }
}
```

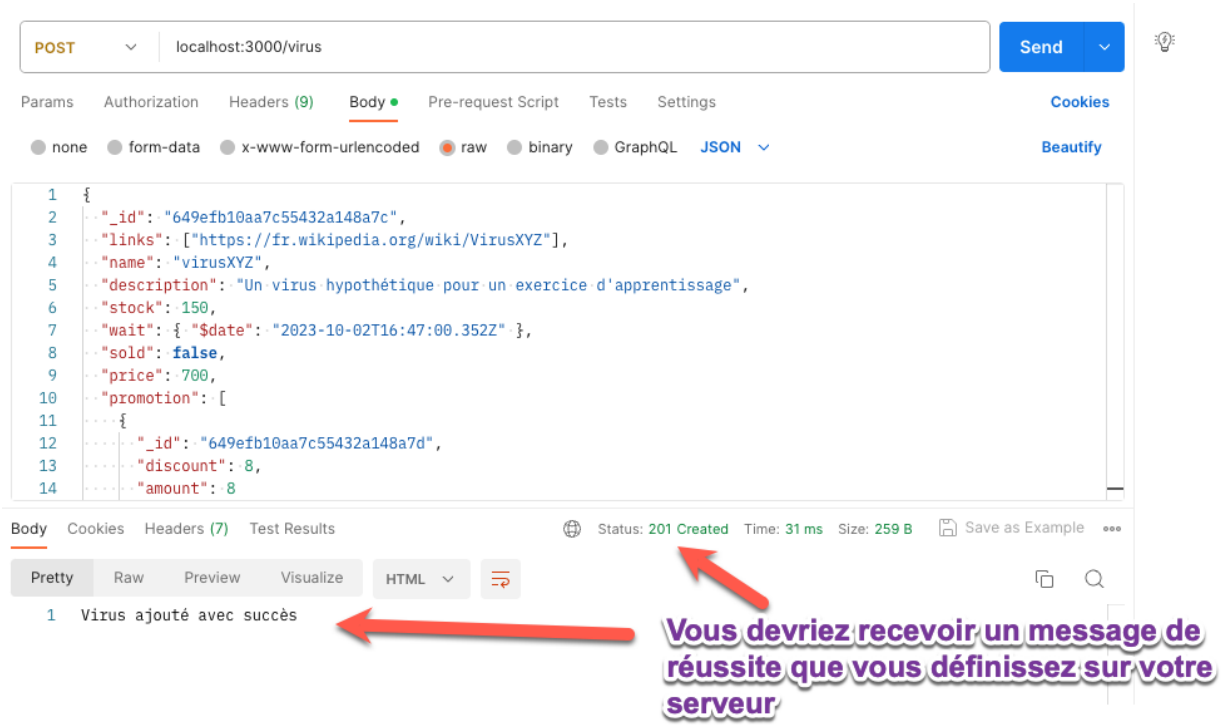

Instructions :

- Middleware pour Parser le JSON :
    - Intégrez le middleware **app.use(express.json());** dans votre fichier **server.js**. Ce middleware est nécessaire pour parser le corps des requêtes JSON reçues par le serveur.
- Création de la Route POST :
    - Créez une nouvelle route POST dans **server.js**.
    - Cette route doit être capable de recevoir un objet JSON contenant les détails du nouveau virus et de l'enregistrer dans le fichier **data/data.json**.
    - Utilisez la syntaxe async/await pour gérer les opérations asynchrones de manière efficace et propre.
- Implémentation dans **models/items.js** :

    - Développez une méthode async qui reçoit l'objet du virus, lit le fichier **data/data.json**, ajoute le nouveau virus à la liste et sauvegarde les modifications dans le fichier.
- Test avec Postman :
    - Postman est un outil de collaboration pour le développement d'API qui vous permet de construire, tester et modifier des API de manière rapide et facile. Il offre une interface intuitive où vous pouvez envoyer des requêtes HTTP, visualiser les réponses et même créer des tests automatisés pour vos API.
    - https://www.postman.com/downloads/postman-agent/
    - https://welovedevs.com/fr/articles/postman/




Overview
POST localhost:3000/virus
No Environment
localhost:3000/virus
Save
POST
localhost:3000/virus
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"_id": "649efb10aa7c55432a148a7c",
"links": ["https://fr.wikipedia.org/wiki/VirusXYZ"],
"name": "virusXYZ",
"description": "Un virus hypothétique pour un exercice d'apprentissage",
"stock": 150,
"wait": { "$date": "2023-10-02T16:47:00.352Z" },
"sold": false,
"price": 700,
"promotion": [
{
"_id": "649efb10aa7c55432a148a7d",
"discount": 8,
"amount": 8
},
{
"_id": "649efb10aa7c55432a148a7e",
"discount": 15,
"amount": 20
},
{
"_id": "649efb10aa7c55432a148a7f",
"discount": 25,
"amount": 50
}
],
"object": { "code": "XYZXYZXYZXYZ" }
}
Mettez l'objet virus dans le corps de la requête en tant que json


POST
localhost:3000/virus
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"_id": "649efb10aa7c55432a148a7c",
"links": ["https://fr.wikipedia.org/wiki/VirusXYZ"],
"name": "virusXYZ",
"description": "Un virus hypothétique pour un exercice d'apprentissage",
"stock": 150,
"wait": { "$date": "2023-10-02T16:47:00.352Z" },
"sold": false,
"price": 700,
"promotion": [
{
"_id": "649efb10aa7c55432a148a7d",
"discount": 8,
"amount": 8
Body
Cookies
Headers (7)
Test Results
Status: 201 Created
Time: 31 ms
Size: 259 B
Save as Example
Pretty
Raw
Preview
Visualize
HTML
Virus ajouté avec succès
Vous devriez recevoir un message de réussite que vous définissez sur votre serveur