TP NodeJS - API TCG

Première partie

Présentation

L'objectif est de créer une API RESTful en Node.js pour une application de jeu de cartes à collectionner (TCG).

Le TP se déroulera en plusieurs parties, cette première partie s'oriente sur l'initialisation du projet, la gestion des utilisateurs et les premières gestion des cartes.

API REST

Une API (Application Programming Interface) est un ensemble de règles et d'outils qui permet à différents logiciels de communiquer entre eux.

Elle définit comment des fonctionnalités ou des données peuvent être utilisées par d'autres programmes, sans exposer leur fonctionnement interne.

REST (Representational State Transfer) est un style d'architecture pour concevoir des API web.
Il repose sur des principes simples :

- utilisation du protocole HTTP
- accès aux ressources via des URL
- opérations standards (GET, POST, PUT, DELETE)
- communication sans état (stateless), chaque appel est indépendant

TCG

TCG pour Trading Card Game, en français : jeu de cartes à collectionner.

Il s’agit d’un type de jeu où les joueurs construisent des decks personnalisés à partir de cartes qu'ils collectionnent, puis s'affrontent selon des règles spécifiques.

Les cartes peuvent représenter des personnages, des sorts, des objets ou des effets, et elles ont souvent différentes raretés.

Aspects techniques

Express sera utilisé pour gérer les endpoints et le routing de l’API (voir TP précédent)

Les données seront stockées sous forme de fichiers JSON (pas de base de données).

Postman sera utilisé pour tester les endpoints et valider l’avancement du TP.

Initialisation du projet

Créer un dossier “tcg-api” avec un fichier principal **`app.js`** et installer Express comme pour le précédent TP.

Utiliser Express et ajouter une première route à la racine de l’application renvoyant un message :

```
app.get("/", (req, res) => {
  res.json(
    {
      message : "Bienvenue sur l'API TCG",
      data : {}
    }
  );
});
```

On gardera ce format de réponse pour tout les appels à l’API :

- Message : Affiche un message de réponse selon l’appel, sert également pour les erreurs
- Data : Contient éventuellement les données renvoyées selon l’endpoint appelé

Ajouter cette ligne dans **`app.js`** :

```
app.use(express.urlencoded({ extended: true }));
```

Cela nous permet de gérer les données des formulaires correctement pour la suite.

Ajouter un fichier **.gitignore** et ignorer le dossier **node_modules**, partager le projet sur Github à lucas-bierne.

On ignore ce dossier car il contient toutes les dépendances et alourdirait énormément les commits inutilement, on ne souhaite garder que le code réalisé.

Gestion des utilisateurs

Créer un fichier à **data/users.json** contenant :

```
[
  {
    "id": 1,
    "username": "Exemple",
    "password": "exemple",
    "collection": []
  }
]
```

Il s'agit du fichier de stockage de nos utilisateurs, on peut y voir un utilisateur servant d'exemple pour présenter la structure.

Créer un module **`users.js`** (voir le précédent TP pour les modules), qui contiendra toute la gestion des utilisateurs.

Création des utilisateurs

On va créer un premier endpoint à l'API : la création d'utilisateur via un envoi en POST.

Dans notre module users, ajouter une nouvelle fonction vide **`RegisterUser(req, res)`** et l'exporter.

Ensuite, l'utiliser dans le routing d'express de cette manière (sans oublier le require) :

```
app.post("/register", users.RegisterUser);
```

Maintenant que la route est créée vers la méthode de création d'utilisateur, on peut la remplir.

Pour récupérer les informations envoyées par POST, on peut les récupérer dans `req.body` , par exemple pour récupérer le username on pourra utiliser `req.body.username`

Exemple de la fonction qui récupère le username :

```
function RegisterUser(req, res)
{
    if(!req.body)
    {
        res.status(400).json({"message": "Erreur : Aucune données"});
        return;
    }

    let username = req.body.username;

    // TODO

    res.json({"message": "OK"});
}
```

On remarquera qu'un message d'erreur est envoyé si le formulaire est vide.

Note :
Les mots de passe sont en clair dans le fichier JSON. Dans le cadre du TP çe n'est pas un problème, mais pour les plus courageux, il est possible de les encrypter avec le module npm bcrypt, par exemple.

Postman

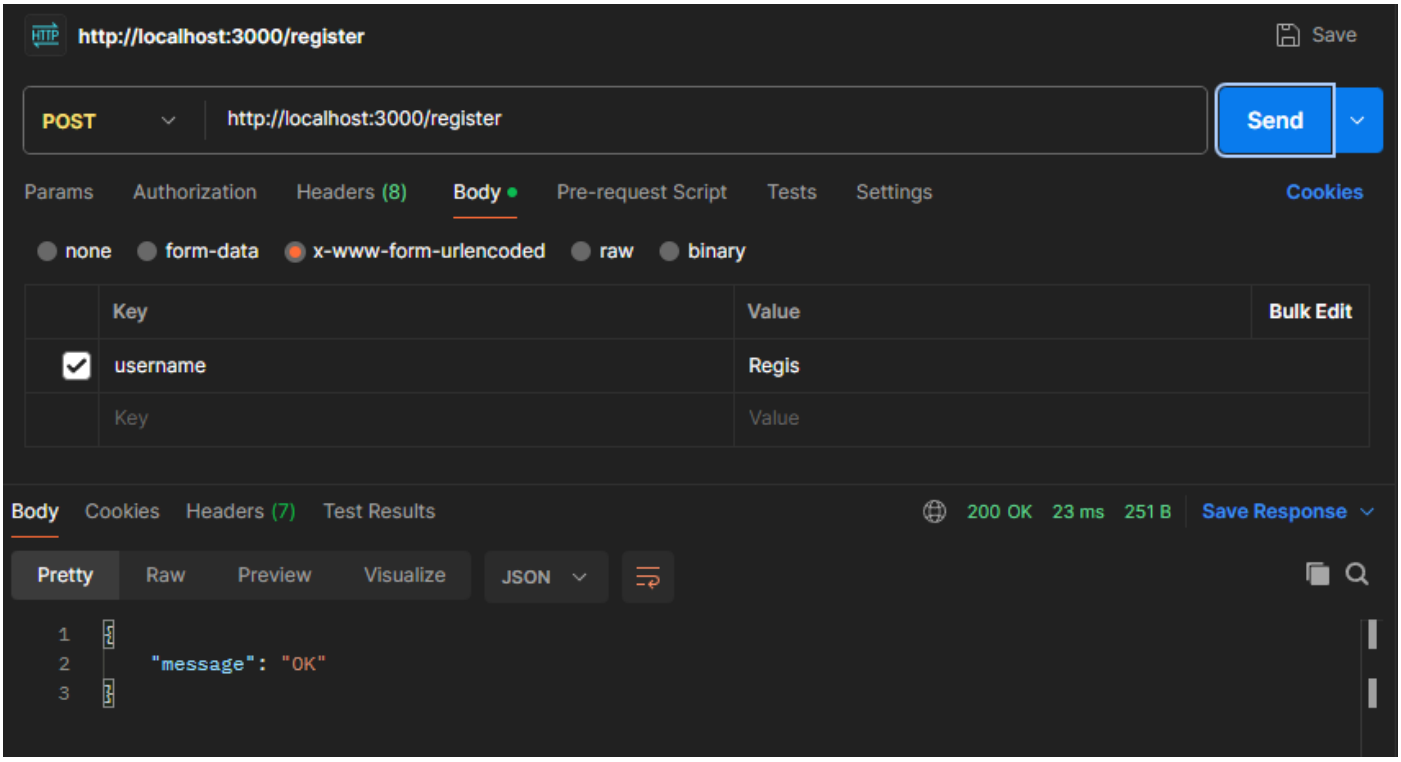
On va tester la fonction avant d'aller plus loin, pour cela on va utiliser Postman.

Postman est une application permettant de tester des API, disponible en client léger et sans création de compte.

Télécharger Postman : https://www.postman.com/downloads/

Une fois lancé, l'utiliser pour appeler notre API :

- sur l'url /register
- en POST
- le body en "x-www-form-urlencoded"
- notre données "username" avec une valeur



On peut maintenant utiliser Postman pour tester et debugger facilement l'API.

Une fois que l'appel fonctionne, on peut terminer la fonction, elle doit donc :

- Récupérer les données envoyées
- Lire le fichier users.json pour récupérer les utilisateurs
- Ajouter le nouvel utilisateur dans la liste
- Enregistrer la nouvelle liste

Authentification des utilisateurs

Maintenant que les utilisateurs sont créés, on va gérer l'authentification.

L'authentification sera gérée avec un token qui sera renvoyé par l'api si un utilisateur réussi à se connecter.
Ce token sera par la suite requis pour tous les endpoints nécessitant un utilisateur connecté.

On va créer un nouvel endpoint POST **`/login`** qui amène à une fonction **`Login(req, res)`** comme la précédente.

Cette fonction récupère également un username et un password, mais cette fois elle les utilise pour vérifier l'utilisateur et s'il est valide, rengistre le token dans l'utilisateur et le renvoie.

Exemple de réponse :

```
{
    message : "Authentification réussie",
    data :
    {
        token : "H2G242"
    }
}
```

Le token sera une suite de caractères alphanumériques au choix, pour ce TP la méthode de génération est libre du moment qu'il semble assez unique.

Pour les plus motivés, il est possible d'utiliser des modules spécialisés pour une gestion de token complète, exemples : token-generator, jsonwebtoken.

Données utilisateur

On va maintenant créer un simple endpoint qui renvoie les informations d'un utilisateur.

Créer un endpoint GET **`/user`** avec la fonction correspondante, toujours dans la même logique.

La fonction recoit un token et récupère l'utilisateur correspondant, puis le renvoi dans la réponse.

Renforcer les endpoints

Maintenant que la base de la gestion des utilisateurs est en place, on va renforcer le fonctionnement, une API doit être robuste !

Gérer au minimum les cas suivants en renvoyant un message adapté (comme dans l'exemple de la fonction `RegisterUser`) :

- RegisterUser
  - Pas de données envoyées
  - Username ou mot de passe absent
  - Utilisateur déjà existant
- Login
  - Pas de données envoyées
  - Username ou mot de passe absent
  - Utilisateur introuvable
- User
  - Token invalide

Créer un endpoint **/disconnect** qui reçoit un token, et le supprime de l'utilisateur si valide.

Gestion des cartes

On va maintenant s'occuper des cartes à collectionner.

On va commencer par créer la liste des cartes disponibles, pour cela créer un fichier **data/cards.json** et le remplir avec quelques données.

Exemple :

```
[
  { "id": 1, "name": "Bulbizarre", "rarity": "common" },
  { "id": 4, "name": "Salamèche", "rarity": "common" },
  { "id": 7, "name": "Carapuce", "rarity": "common" },
  { "id": 25, "name": "Pikachu", "rarity": "rare" },
  { "id": 150, "name": "Mewtwo", "rarity": "legendary" }
]
```

Créer un nouveau module **cards.js**

Ouverture d'un booster

On va maintenant créer l'ouverture d'un booster.

Dans le domaine des jeux de cartes à collectionner un booster est une pochette de cartes aléatoires destinée à rejoindre la collection d'un joueur.

Créer un nouvel endpoint POST **/booster** qui appelle une fonction **`OpenBooster`** dans le nouveau module.

Fonctionnement de la fonction :

- Recoit un token utilisateur et le vérifie
- Tire aléatoirement 5 cartes dans la liste des cartes disponibles
- Attribue les cartes à la collection de l'utilsateur
- Réponse : message de confirmation et les cartes du booster

Une fois la fonction opérationnelle, il est possible de consulter la collection depuis le endpoint **/user**

Amélioration

On améliore maintenant l'ouveture d'un booster avec de nouvelles fonctionnalités :

Délai

On va ajouter un délai de 5 minutes entre chaque ouverture de booster.
Pour cela, lors de l'ouverture, enregister dans l'utilisateur le moment d'ouverture :

```
user.lastBooster = Date.now();
```

Cela stocke la date actuelle en timestamp.

Ajouter une vérification au début de l'ouverture pour vérifier que "`lastBooster`" a plus de 5 minutes.

Rareté

Gérer la rareté des cartes.
Il s'agit de définir des chances d'apparition différentes selon la rareté, ce qui rend le tirage plus réaliste.
Exemple :

- Common : 80%
- Rare : 15%
- Legendary : 5%

Il est possible de trouver des sources externes pour cet algorithme, cependant elles doivent être complètement comprises si elles sont intégrées (très forte chance d'évaluation dessus !)