TP NodeJS - API TCG

Troisième partie

Migration vers une base de données

L’objectif est de migrer l'API REST depuis un stockage par fichiers JSON vers une base de données, tout en conservant la logique métier et les fonctionnalités existantes.

Deux propositions de base de données sont proposées : MongoDB ou SQL.

MongoDB

ODM (Object Document Mapper)

Un ODM permet de manipuler une base de données orientée document (comme MongoDB) avec des objets JavaScript. Il fournit des schémas, des validations, et des méthodes facilitant la manipulation des données.

Mongoose

Mongoose est un package qui sert ODM pour MongoDB.
Il permet de définir des schémas de données, de valider les entrées, et d’effectuer des requêtes sur MongoDB avec une API JS intuitive.

https://www.npmjs.com/package/mongoose

Documentation : https://mongoosejs.com/docs/index.html

SQL

ORM (Object Relational Mapper)

Similaire à un ODM mais pour des bases de données relationnelles comme PostgreSQL ou MySQL. Il permet de représenter les tables et les relations sous forme d'objets JS.

Sequelize

Permet de connecter l'application à une base relationnelle (PostgreSQL, MySQL, SQLite...)
Fournit des outils pour les migrations, les relations, et les requêtes avancées.

https://www.npmjs.com/package/sequelize

Documentation : https://sequelize.org

Connecteur

Créer un fichier de config **`db.js`**

Ce fichier va posséder la connexion à la base de données qu'il exporte en tant que module, on va le "require" partout où on a besoin de la bdd.

Exemples :

```
var bdd = require("mongoose");

bdd.connect("mongodb://localhost/tcg", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

module.exports = bdd;
```

```
const { Sequelize } = require("sequelize");

var bdd = new Sequelize("database", "username",
"password", {
  host: "localhost",
  dialect: "mysql",
});

await bdd.authenticate();

module.exports = bdd;
```

Modèles

Les manipulations de la base de données passent à travers des structures d'objets appelées "modèles", pour les deux connecteurs.

Exemples de modèles avec leur utilisations :

```
const mongoose = require("mongoose");

// Modèle
const User = new mongoose.Schema({
  username: { type: String, unique: true,
required: true },
  birthday: { type: Date, default: null },
});


// Exemple de création
let newUser = new User();
newUser.username = Simon';
await user.save();

// Exemple de sélection
let users = await User.find({});
```

```
const { DataTypes } = require('sequelize');
const bdd = require('db.js');


// Modèle
const User = bdd.define('User', {
 username: DataTypes.STRING,
 birthday: DataTypes.DATE,
});

// Exemple de création
let newUser = await User.create({
 username: 'Simon',
 birthday: new Date(1994, 5, 12),
});

// Exemple de sélection
let users = await User.findAll();
```

Créer un dossier **`Models`** qui accueillera tous les fichiers de modèles correspondants à la base de données.
Faire un fichier pour chaque modèle (avec export).

Migration des endpoints

Maintenant que le connecteur et les modèles sont prêts, il faut transformer le code pour manipuler les données en base de données et non plus depuis les fichiers JSON.

Adapter les endpoints /register, /login, etc… pour utiliser les méthodes des connecteurs en se basant sur leur documentation.

Vérifier avec Postman et l'interface leur bon fonctionnement au fur et à mesure des transformations.

Nouvelle fonctionnalité : les enchères

Fonctionnement

Les enchères vont utiliser la monnaie créée dans le précédent TP.

Un vendeur peut mettre en vente une de ses cartes, elle est alors retirée de sa collection et placée en vente pour une durée limitée.

Lorsqu'un utilisateur participe à une enchère en augmentant le cout, le montant choisi est débité de son solde.
S'il y a un précédent enchérisseur il est alors remboursé.
Le vendeur ne peut pas enchérir sur sa propre enchère.

Si une enchère est terminée, le vendeur ou l'acheteur peut la clôturer.
Lors de la clôture le vendeur reçoit la somme en monnaie et la carte est ajoutée au gagnant.

Enchère

Nouvelle structure d'objet pour les enchères :

Attribut	Type	Description
id	int	Identifiant unique d'une enchère
card_id	int	Identifiant de la carte vendue
seller_id	int	Identifiant du vendeur
end_date	date	Date de fin de l'enchère
bidder_id	int	Identifiant de l'acheteur
bid	int	Montant de l'enchère

Nouveaux endpoints

Méthode	Route	Description	Paramètres
POST	`/enchere`	Créer une enchère	token, id de la carte
POST	`/encherir`	Placer une enchère	token, id de l'enchère, montant
GET	`/encheres`	Récupère les enchères	token
POST	`/cloturer`	Clôture une enchère	token, id de l'enchère

Penser à gérer les cas spéciaux avec des codes et message d'erreurs adaptés,
en voici quelques-uns :

- Créer une enchère sans avoir la carte
- Enchérir avec une monnaie insuffisante
- Enchérir sur sa propre enchère
- Tenter de clôturer une enchère non terminée
- Etc...

Créer une page côté front affichant toutes les enchères, et implémenter tout le fonctionnement.

Sur la page de profil utilisateur, afficher la liste des enchères concernées (en tant que vendeur et acheteur).