Création d'une API Pokémon

Nous allons tâcher de créer une API permettant d'effectuer des opérations CRUD sur les pokémons. Ce projet sera réalisé avec Express pour la gestion des routes et avec une base de donnée NoSQL.

Prérequis

- NodeJS version LTS
- Compte MongoDB Cloud et les liens d'accès au cluster de base de données
- Postman : pour vérifier nos requêtes

Configurer votre projet

Dans le fichier `index.js` instancier une connexion à la base de données et à express. Vous pouvez utiliser le package **dotenv** en supplément si vous souhaitez une meilleure gestion des variables d'environnement.

Gestion des endponts Pokemons

Ainsi notre fonction renverra l'objet Promesse. Mais elle n'est pas utilisable en tant que telle, pour l'utiliser il va falloir passer par la fonction `then` qui sera chaîné à la promesse. La fonction prendra elle-même en paramètre un callback qui prendra en paramètre le retour du resolve.

Models

Avant de commencer nos routes ou controlleurs il serait judicieux de créer notre modèle afin de savoir ce que nous cherchons à intégrer. Ce dernier permettra d'intégrer un pokémon suivant cet exemple dans le fichier `pokemon.model.js` du dossier `src/pokemons`:

```
{
  "pokedexNumber":1,
  "name":"Bulbizarre",
  "types":["plante","poison"],
  "height":160,
  "weight":200

}
```

Règles de schéma :

- Trouver un moyen de ne pouvoir insérer qu'un des types de la liste définie par le jeu et aucun autre. Les types sont : acier, combat, dragon, eau, électrik, feu, fée, glace, insecte, normal, plante, poison, psy, roche, sol, spectre, ténèbres et vol.
- Le nom et le pokedexId sont uniques
- Attention le pokedexId correspondant au numéro dans le pokédex, il est différent de la clé `_id` qui sera crée automatiquement à la création d'un document dans la base.

Les champs pokdexId,name et types sont obligatoire.

Routes

Nous allons maintenant attaquer le gros du travail en commençant par créer dans le dossier `src/pokemons` le fichier `pokemons.route.js`. Ce fichier contiendra une route pour chacun des endpoints suivant:

- `GET:/pokemons` : permet d'obtenir la liste de tous les pokémons
- `GET:/pokemons/:pokemonId` : permet d'obtenir un pokémon spécifique en fonction de la clé `_id` passé en paramètre
- `POST:/pokemons` : permet d'enregistrer le pokémon dont les propriétés sont passées dans le body de la requête
- `PATCH:/pokemons/pokemonId` : permet de modifier le pokémon donc l'`_id` est passé en paramètre et les propriétés passées dans le body.
- `DELETE:/pokemons/pokemonId:` : permet de supprimer le pokémon renseigné avec son `_id`.

Ces fonctions seront extrêmement simple pour le moment et n'auront pour but d'appeler que les fonctions controllers associées à chacun des endpoints que nous développerons dans la partie suivante.

N'oubliez pas les routes ne doivent pas contenir de logiques particulières ou de vérifications c'est le rôle des controllers.

Controllers

Pour chacune des routes il y a une méthode CRUD associée

- GET: lecture un ou plusieurs documents
- POST: création d'un document
- PATCH: modification d'un document
- DELETE : suppression d'un document

Il va donc falloir un controller qui sera associé à chacun des routes précedemment définis. Nous aurons donc les fonctions controllers suivantes: `listPokemons`,`getPokemonById`,`createPokemon`,`updatePokemon` et `deletePokemon`.

Créer ses fonctions dans le fichier `pokemons.controller.js` du dossier `src/pokemons` et implémenter les. L'étape finale de ces fonctions sera chacun d'appeler le modèle Pokémon et d'utiliser la fonction mongoose associées à l'opération CRUD en cours.

N'oubliez pas de tester vos endpoint avec Postman.

Utilisateurs et permission d'accès

Nous allons essayer d'ajouter des fonctionnalités de permissions afin que certains endpoint ne soient pas accessibles si vous n'êtes pas connectés. Par exemple il serait intéressant que seuls les utilisateurs authentifiés puissent créer, modifier ou supprimer des pokémons.

Modèles users

Créer un modèle dans `src/users` qui correspond à un utilisateur selon l'exemple ci-dessous:

```
{
  "email":"fourny.valentin@gmail.com",
  "password":"12345"
}
```

L'email et le mot de passe seront obligatoire. Le mail devra être unique.

Endpoints users

Créer l'endpoint de création d'un utilisateur `POST:/users` à travers les fichiers `users.route.js` et `users.controller.ts`. Attention dans la partie controlleur veillez bien à crypter le mot de passe entré dans le body afin qu'il ne soit pas stocké en clair dans la base de donnée.

Pour ce faire vous pouvez utiliser le package: `npm install bcrypt`. Renseignez vous dans la doc concernant la méthode `hash`. Elle renvoie une promesse que vous devrez gérer.

On aura ensuite un second un second endpoint `POST:/users/login`. Cet endpoint récupèrera l'utilisateur en base de données avec les informations email et mot de passe entrés dans le body de la requête. Il renverra ensuite un JWT construit à l'aide du package `jsonwebtoken` et du retour de la requête en BDD permettant d'obtenir les informations utilisateurs.

Nous utilons ici une méthode POST et non GET caril s'agit d'un point de vue applicatif de la création d'une session utilisateur. Dans le cas d'une création le POST est donc plus logique.

Middleware de permission

Maintenant que nous gérons les utilisateurs nous ajouter une vérification de la validité du JWT dans certaines de nos requêtes. Pour ce faire créer un fichier `jwt.middleware.js` dans le dossier `src/common`. Ce dossier contient toutes les logiques communes à plusieurs endpoints.

Ce fichier contiendra une fonction `verifyJWT(req,res,next)` qui vérifiera l'objet `req.headers.authorization` pour récupérer le JWT entré en paramètre de votre requête. Il vérifiera le JWT grâce au package `jsonwebtoken`. Si le jwt est valide il appelera la fonction `next()` afin de continuer le processus de la requête sinon il renverra une réponse `401 Unauthorized`.

Connecter le middleware de permission à nos routes

Jusque là les fichiers de route ne faisaient pas grande chose en dehors de l'appel du controller associé. Désormais vous allez vérifier la validité du JWT avec le middleware que nous venons de créer avant l'appel de chaque controller pour les routes suivantes:

- Création d'un pokémon
- Modification d'un pokémon
- Suppression d'un pokémon

Test des permissions

Essayer de tester votre application en indiquant ou non dans votre requêtre votre JWT. Désormais les routes où le JWT est requis devraient vous renvoyer des erreurs et ne pas réaliser d'opérations sur la BDD

si vous n'avez pas de JWT valide.