Marquer comme terminé

Ouvert le : jeudi 24 octobre 2024, 00:00

Cet exercice consiste à refactoriser une API REST en Node.js en séparant la logique des routes et des contrôleurs. Nous déplaçons les méthodes de gestion des utilisateurs (création, mise à jour, suppression) depuis les routes vers un fichier de contrôleur (`usersControllers.js`). Nous remplaçons les manipulations de données en JavaScript par des requêtes SQL pour interagir avec une base de données SQLite. L'exercice comprend également la création de nouvelles branches Git pour organiser le travail et l'ajout de vérifications pour valider les données avant de les insérer ou de les mettre à jour dans la base de données.

https://github.com/web-rest-api/sqlite-queries/tree/main

Comment forker et cloner ce projet

1. **Forker le projet** : Cliquez sur le bouton "Fork" en haut à droite de la page GitHub pour créer une copie de ce dépôt sur votre compte GitHub.
2. **Cloner le projet** : Sur votre dépôt forké, cliquez sur le bouton "Code" et copiez l'URL du dépôt. Ensuite, sur votre terminal, exécutez la commande suivante pour cloner le projet :

   ```
   git clone <URL_DU_DEPOT>
   ```

   Remplacez <URL_DU_DEPOT> par l'URL de votre dépôt GitHub forké.
3. **Naviguer dans le dossier :**

   ```
   cd nom-du-dossier-du-projet
   ```

Installation des dépendances

Une fois que vous avez cloné le projet, installez les dépendances en exécutant :

```
npm install
```

4. **Lancer le projet en local**

Pour démarrer le serveur localement, exécutez :

```
npm start
```

Le serveur devrait maintenant être en cours d'exécution à l'adresse http://localhost:3000. Vous devriez voir le message suivant dans votre terminal :

```
Example app listening at http://localhost:3000
```

Et vous pouvez tester l'API en accédant à http://localhost:3000/api/cars/test

5. **Initialiser la base de données** Le projet contient un fichier seed.js pour initialiser la base de données avec des données de test. Pour le lancer, exécutez :

```
node seed.js
```

Cela va ajouter quelques données de base dans la base de données SQLite.

6. **Ajouter les opérations CRUD**

Le fichier `routes/carsRoutes.js` contient des routes pour gérer les voitures. Actuellement, certaines routes de base sont définies. Voici les étapes pour ajouter les opérations CRUD (Create, Read, Update, Delete) :

GET - Récupérer toutes les voitures : Cette route est déjà configurée.

GET - Récupérer une voiture spécifique par id :

```
const { id } = req.params // obtenir l'id à partir des paramètres
```

obtenir une seule base de voiture sur son identifiant

```
db.get("SELECT * FROM cars WHERE id = ?", [id], (err, row) => {
        if (err) {
                res.status(500).json({ error: err.message })
        } else {
                res.json(row)
        }
})
```

POST - Ajouter une nouvelle voiture :

```
const { carName, carYear, carImg } = req.body
```

nous pouvons nous assurer que les données sont correctes ici. Nous devrions ajouter une sorte de validation

carName vide? carName string? carName des letre étranges ex: &é"'-è_ç*' ?

si tout est valide alors nous exécutons la requête

```
db.run(
        "INSERT INTO cars (carName, carYear, carImg) VALUES (?, ?, ?)",
        [carName, carYear, carImg],
        function (err) {
                if (err) {
                        res.status(500).json({ error: err.message })
                } else {
                        res.json({ id: this.lastID })
                }
        }
)
```

PUT - Mettre à jour une voiture par id :

cette fois nous avons besoin de l'identifiant et des données envoyées dans le corps

```
const { id } = req.params
const { carName, carYear, carImg } = req.body
```

on valide ?

la query

```
db.run(
        "UPDATE cars SET carName = ?, carYear = ?, carImg = ? WHERE id = ?",
        [carName, carYear, carImg, id],
        function (err) {
                if (err) {
                        res.status(500).json({ error: err.message })
                } else {
                        res.json({ changes: this.changes })
                }
        }
)
```

DELETE - Supprimer une voiture par id :

```
const { id } = req.params
```

si on le trouves pas ... error 404 🎀 🧨

query:

```
db.run("DELETE FROM cars WHERE id = ?", [id], function (err) {
        if (err) {
                res.status(500).json({ error: err.message })
        } else {
                res.json({ changes: this.changes })
        }
})
```

Tester les routes

Vous pouvez tester les routes à l'aide de Postman ou de tout autre outil similaire en envoyant des requêtes HTTP à http://localhost:3000/api/cars.

- GET http://localhost:3000/api/cars : Récupère toutes les voitures.
- GET http://localhost:3000/api/cars/:id : Récupère une voiture spécifique par id.
- POST http://localhost:3000/api/cars : Ajoute une nouvelle voiture en envoyant un JSON avec carName, carYear, et carImg.
- PUT http://localhost:3000/api/cars/:id : Met à jour les informations d'une voiture par id.
- DELETE http://localhost:3000/api/cars/:id : Supprime une voiture par id.

Ajouter un travail

Texte en ligne

Enregistrer Annuler