2 - Deuxième séance : Base de données SQLite

4.1 Qu'est-ce qu'une base de données ?

Une base de données est un système organisé pour stocker, gérer et récupérer des informations. Nous utilisons SQLite, qui stocke toutes les données dans un seul fichier `.db`.

Concepts SQL de base :

- **Table** : Structure qui contient les données (comme un tableau Excel)
- **Colonne** : Un attribut (brand, model, year, etc.)
- **Ligne (row)** : Une entrée dans la table (une voiture)
- **Clé primaire (PRIMARY KEY)** : Identifiant unique pour chaque ligne

4.2 Création du fichier database.js

Créez un fichier `database.js` à la racine du projet. Ce fichier contiendra toute la logique de connexion et d'initialisation de la base de données.

Concepts importants :

1. **sqlite3.Database()** : Crée ou ouvre une base de données
2. **db.run()** : Exécute une requête SQL qui modifie la base
3. **db.get()** : Récupère une seule ligne
4. **db.all()** : Récupère toutes les lignes correspondantes

```
const sqlite3 = require('sqlite3').verbose(); const path =
require('path'); // Chemin vers le fichier de base de données const dbPath
= path.resolve(__dirname, 'cars.db'); // Création/ouverture de la base de
données const db = new sqlite3.Database(dbPath, (err) => { if (err) {
console.error('❌ Erreur de connexion à la base de données:',
err.message); } else { console.log('✅ Connecté à la base de données
SQLite'); } }); // Création de la table cars si elle n'existe pas const
createTableQuery = ` CREATE TABLE IF NOT EXISTS cars ( id INTEGER PRIMARY
KEY AUTOINCREMENT, brand TEXT NOT NULL, model TEXT NOT NULL, year INTEGER
NOT NULL, color TEXT, price REAL, mileage INTEGER, description TEXT,
created_at DATETIME DEFAULT CURRENT_TIMESTAMP ) `;
db.run(createTableQuery, (err) => { if (err) { console.error('❌ Erreur
lors de la création de la table:', err.message); } else { console.log('✅
Table cars créée ou déjà existante'); } }); // Export de la base de
données module.exports = db;
```

Explication du schéma de table :

- `INTEGER PRIMARY KEY AUTOINCREMENT` : ID unique qui s'incrémente automatiquement
- `TEXT NOT NULL` : Champ texte obligatoire
- `INTEGER` : Nombre entier
- `REAL` : Nombre décimal
- `DATETIME DEFAULT CURRENT_TIMESTAMP` : Date de création automatique

✏️ **Exercice 5 :**

- Créez le fichier `database.js` avec le code ci-dessus
- Redémarrez votre serveur
- Vérifiez qu'un fichier `cars.db` a été créé à la racine du projet

4.3 Création du fichier seed.js

Le fichier `seed.js` permet d'insérer des données de test dans la base. C'est très utile pour le développement.

```
const db = require('./database'); // Données de test const sampleCars = [ { brand: 'Ferrari', model: '250 GTO', year: 1962, color: 'Rouge', price: 45000000, mileage: 12000, description: 'Voiture de collection exceptionnelle' }, { brand: 'Porsche', model: '911 Carrera RS', year: 1973, color: 'Blanc', price: 850000, mileage: 45000, description: 'Légendaire modèle RS' }, { brand: 'Jaguar', model: 'E-Type', year: 1961, color: 'Bleu', price: 320000, mileage: 78000, description: 'Icône du design automobile' }, { brand: 'Mercedes-Benz', model: '300 SL', year: 1955, color: 'Argent', price: 1200000, mileage: 34000, description: 'Portes papillon emblématiques' }, { brand: 'Aston Martin', model: 'DB5', year: 1964, color: 'Gris', price: 750000, mileage: 56000, description: 'La voiture de James Bond' } ]; // Fonction pour insérer les données function seedDatabase() { // D'abord, on vide la table db.run('DELETE FROM cars', (err) => { if (err) { console.error('❌ Erreur lors du vidage de la table:', err.message); return; } console.log('🗑️ Table vidée'); // Puis on insère les nouvelles données const insertQuery = ` INSERT INTO cars (brand, model, year, color, price, mileage, description) VALUES (?, ?, ?, ?, ?, ?, ?) `; let insertedCount = 0; sampleCars.forEach((car) => { db.run( insertQuery, [car.brand, car.model, car.year, car.color, car.price, car.mileage, car.description], (err) => { if (err) { console.error('❌ Erreur lors de l\'insertion:', err.message); } else { insertedCount++; console.log(`✅ Voiture insérée: ${car.brand} ${car.model}`); if (insertedCount === sampleCars.length) { console.log('\n🎉 Base de données initialisée avec succès !'); db.close(); } } } ); }); }); } // Exécution du seed seedDatabase();
```

✏️ **Exercice 6 :**

- Créez le fichier `seed.js`
- Ajoutez un script dans `package.json` :

```
"scripts": { "start": "node index.js", "dev": "nodemon index.js", "seed": "node seed.js" }
```

- Exécutez `npm run seed`
- Vérifiez que les données ont bien été insérées

4.4 Connexion de la base aux routes

Maintenant, modifiez votre fichier `index.js` pour connecter réellement la base de données à vos routes.

Ajoutez en haut du fichier :

```
const db = require('./database');
```

Modifiez la route GET pour récupérer toutes les voitures :

```
app.get('/api/cars', (req, res) => { const query = 'SELECT * FROM cars ORDER BY year DESC'; db.all(query, [], (err, rows) => { if (err) { return res.status(500).json({ error: 'Erreur lors de la récupération des voitures', details: err.message }); } res.json({ message: 'Liste des voitures', count: rows.length, data: rows }); }); });
```

Modifiez la route GET pour récupérer une voiture par ID :

```
app.get('/api/cars/:id', (req, res) => { const id = req.params.id; const query = 'SELECT * FROM cars WHERE id = ?'; db.get(query, [id], (err, row) => { if (err) { return res.status(500).json({ error: 'Erreur serveur', details: err.message }); } if (!row) { return res.status(404).json({ error: 'Voiture non trouvée' }); } res.json({ message: 'Voiture trouvée', data: row }); }); });
```

✏️ **Exercice 7** :

- Modifiez votre `index.js` avec le code ci-dessus
- Testez avec Postman :
  - GET `http://localhost:3000/api/cars` → Devrait retourner 5 voitures