Utiliser EJS pour les templates

1. Installation des modules supplémentaires

npm install ejs

Ensuite, configurez votre application pour utiliser EJS comme moteur de rendu de templates :

```
// Configurer EJS comme moteur de templates
app.set('view engine', 'ejs');
```

controllers/products.js : Ce fichier contient toutes les fonctions CRUD pour manipuler les produits dans la base de données.

routes/products.js : Ce fichier gère les routes et appelle les fonctions définies dans controllers/products.js.

app.js : Vous utilisez ce fichier pour importer les routes depuis routes/products.js et les utiliser dans l'application.

1. **Créez un fichier db.js**

Dans le dossier racine de votre projet (ou dans un dossier config si vous préférez), créez un fichier db.js

```
// db.js
const sqlite3 = require('sqlite3').verbose();
// Connexion à la base de données SQLite (mémoire ou fichier)
const db = new sqlite3.Database(':memory:', (err) => {
  if (err) {
    console.error('Error opening database ' + err.message);
  } else {
    console.log('Connected to the SQLite database.');
  }
});

// Création de la table produits si elle n'existe pas encore
```

```
db.serialize(() => {
  db.run(` CREATE TABLE IF NOT EXISTS products (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    price REAL NOT NULL
  )`);
});
module.exports = db;
```

2. Créer un fichier controllers/products.js

```
// controllers/products.js
const db = require('../db'); // Importez la connexion à la base de données

// Fonction pour ajouter un produit (CREATE)
function addProduct(name, price) {
  return new Promise((resolve, reject) => {
    const query = ` INSERT INTO products (name, price) VALUES (?, ?)` ;
    db.run(query, [name, price], function (err) {
      if (err) {
        return reject(err.message);
      }
      resolve({ id: this.lastID, name, price });
    });
  });
}

// Fonction pour récupérer tous les produits (READ)
function getAllProducts() {
  return new Promise((resolve, reject) => {
    db.all("SELECT * FROM products", [], (err, rows) => {
      if (err) {
        return reject(err.message);
      }
      resolve(rows);
    });
  });
}
```

```
// Fonction pour mettre à jour un produit (UPDATE)
function updateProduct(id, name, price) {
  return new Promise((resolve, reject) => {
    const query = `UPDATE products SET name = ?, price = ? WHERE id = ?`;
    db.run(query, [name, price, id], function (err) {
      if (err) {
        return reject(err.message);
      }
      resolve({ id, name, price });
    });
  });
}

// Fonction pour supprimer un produit (DELETE)
function deleteProduct(id) {
  return new Promise((resolve, reject) => {
    const query = `DELETE FROM products WHERE id = ?`;
    db.run(query, id, function (err) {
      if (err) {
        return reject(err.message);
      }
      resolve(`Product with ID ${id} has been deleted.`);
    });
  });
}

module.exports = {
  addProduct,
  getAllProducts,
  updateProduct,
  deleteProduct
};
```

3. Créez un fichier routes/products.js pour y définir toutes les routes relatives aux produits

```
const express = require('express');

const router = express.Router();

const { addProduct, getAllProducts, updateProduct, deleteProduct } =
require('../controllers/products');
```

```
// Route pour ajouter un produit
router.post('/', async (req, res) => {
  const { name, price } = req.body;
  try {
    const product = await addProduct(name, price);
    res.status(201).json(product);
  } catch (err) {
    res.status(400).json({ error: err });
  }
});

// Route pour récupérer tous les produits
router.get('/', async (req, res) => {
  try {
    const products = await getAllProducts();
    res.status(200).render('products', { products });
  } catch (err) {
    res.status(500).json({ error: err });
  }
});

// Route pour mettre à jour un produit
router.put('/:id', async (req, res) => {
  const { id } = req.params;
  const { name, price } = req.body;
  try {
    const product = await updateProduct(id, name, price);
```

```
    res.status(200).json(product);
  } catch (err) {
    res.status(400).json({ error: err });
  }
});

// Route pour supprimer un produit
router.delete('/:id', async (req, res) => {
  const { id } = req.params;
  try {
    const message = await deleteProduct(id);
    res.status(200).json({ message });
  } catch (err) {
    res.status(400).json({ error: err });
  }
});

module.exports = router;
```

4. **Intégration dans app.js**

Assurez-vous que dans votre app.js, vous importez et utilisez le routeur comme ceci :

```
const express = require('express');
const db = require('./db'); // Importer la connexion à la base de données
const productRoutes = require('./routes/products');

const app = express();
const PORT = 3000;
```

```
// Configurer le middleware pour le JSON
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Utilisation des routes des produits
app.use('/products', productRoutes);

// Lancement du serveur
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

5. **Créer des templates EJS**

Créez un dossier views à la racine de votre projet et un fichier products.ejs pour afficher les produits.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Product List</title>
</head>
<body>
  <h1>Product List</h1>
  <ul>
    <% products.forEach(product => { %>
      <li><strong><%= product.name %></strong>: $<%= product.price %></li>
    <% }) %>
```

```
  </ul>
  <a href="/products/new">Add New Product</a>
</body>
</html>
```

Vous pouvez aussi créer une page pour ajouter un nouveau produit
views/newProduct.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Add New Product</title>
</head>
<body>
  <h1>Add New Product</h1>
  <form action="/products" method="POST">
    <label for="name">Product Name:</label>
    <input type="text" id="name" name="name" required><br><br>
    <label for="price">Product Price:</label>
    <input type="number" id="price" name="price" required><br><br>
    <button type="submit">Add Product</button>
  </form>
</body>
</html>
```

4. Mise à jour du fichier app.js

Maintenant, dans votre fichier app.js , vous pouvez organiser vos routes et inclure le

```
const express = require('express');
```

```
const db = require('./db'); // Importer la connexion à la base de données
const productRoutes = require('./routes/products');

const app = express();
const PORT = 3000;

// Configurer le middleware pour le JSON
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Utilisation des routes des produits
app.use('/products', productRoutes);

// Lancement du serveur
app.listen(PORT, () => {
   console.log(` Server is running on http://localhost:${PORT}` );
});
```