Manuel technique du site e-commerce

Sommaire

1) CRUD des produits

Création d'un produit :
On donne le nom, le prix, l'image, la description et le stock d'un produit sur le formulaire et ensuite une requête préparée est effectuée permettant de créer le produit.

```
const crypto = require('crypto')

exports.addAProduct = async (req, res) => {
    const { nom, prix, image, description, stock } = req.body; // Récupérer les données du formulaire

    if (!nom || !prix || !image || !description  || !stock ) {
        return res.status(400).json({ message: "Tous les champs sont obligatoires." });
    }

    db.pool.query(
        'INSERT INTO Produit (id, nom, prix, image, description, stock) VALUES (?, ?, ?, ?, ?, ?)',
        [crypto.randomUUID(), nom, prix, image, description, stock],
        function (error, results, fields) {
        if (error) {
            console.error(error);
            return res.status(500).json({ message: "Erreur lors de l'ajout du produit." });
        }

        const newProduct = {
            id: results.insertId,
            nom,
            prix,
            image,
            description,
            stock,
        };

        res.status(200).json({ message: "Produit ajouté avec succès." });
        }
    );
}
```

Récupération d'un produit :
On effectue un SELECT * FROM Produit qui permet de récupérer tout la liste de tout les produits et leurs caractéristiques.

```
//// Récupération des produits
exports.getAllProducts = async (req, res) => {
    try {
        console.log("Lancement de la requête d'affichage")
        const rows = await db.pool.query('Select * FROM Produit',
        function (error, results, fields) {
            if (error) throw error;
            res.status(200).json(results)
        });
    }
    catch (err) {
        console.log(err)
    }
}
```

Modification d'un produit :
Cela permet de modifier un produit avec de nouvelles informations.

```
//// Modification d'un produit
exports.editAProduct = async (req, res) => {
    const id = req.params.id;
    const { nom, prix, image, description, stock } = req.body; // Récupérer les données mises à jour

    // Vérifiez que les données requises sont présentes
    if (!nom || !prix || !image || !description || !stock) {
      return res.status(400).json({ message: "Tous les champs sont obligatoires." });
    }

    // Requête SQL pour mettre à jour le produit dans la base de données
    db.pool.query(
      'UPDATE Produit SET nom=?, prix=?, image=?, description=?, stock=? WHERE id=?',
      [nom, prix, image, description, stock, id],
      function (error, results, fields) {
        if (error) {
          console.error(error);
          return res.status(500).json({ message: "Erreur lors de la mise à jour du produit." });
        }

        res.status(200).json({ message: "Produit mis à jour avec succès." });
      }
    );
  }
```

Suppression d'un produit :
Cette fonction permet de supprimer un produit.

```
//// Suppression d'un produit
exports.deleteAProduct = async (req, res) => {
    const id = req.params.id;

    // Supprimez le produit de la base de données
    db.pool.query(
      'DELETE FROM Produit WHERE id = ?',
      [id],
      function (error, results, fields) {
        if (error) {
          console.error(error);
          return res.status(500).json({ message: "Erreur lors de la suppression du produit" });
        }

        res.status(200).json({ message: "Produit supprimé avec succès." });
      }
    );
}
```

2) Inscription/Connexion

Lors de la tentative d'inscription on vérifie si l'utilisateur existe déjà et sinon on hache son mot de passe avec bcrypt plusieurs fois (10) et ensuite on créer le compte dans la base de données et on créé un token qui permet a l'utilisateur d'être connecté 1h.

```
exports.Register = async (req, res) => {
        const results = await db.pool.query('SELECT * FROM client WHERE mail = ?', [mail],
        async function (error, results, fields)  {
            if (results.length > 0) {
                return res.status(400).json({ error: 'Cet utilisateur existe déjà.' });
            }

            // Hacher le mot de passe avec bcrypt
            const hashedPassword = await bcrypt.hash(mdp, 10);

            // Enregistrer le nouvel utilisateur dans la BDD avec isAdmin à 0
            const insertUserQuery = 'INSERT INTO client (id, nom, prenom, mail, mdp, isAdmin) VALUES (?, ?, ?, ?, ?, ?)';
            const insertUserValues = [crypto.randomUUID(), nom, prenom, mail, hashedPassword, 0];
            await db.pool.query(insertUserQuery, insertUserValues);

            // Générer un token JWT pour l'utilisateur nouvellement inscrit
            const token = jwt.sign({ mail }, process.env.API_KEY, { expiresIn: '1h' });

            // Envoyer le token en réponse
            res.json({ token });
             });

    }
    catch (err) {
        console.log(err);
        res.status(500).json({ error: "Erreur lors de l'inscription." });
    }
}
```

Lors de la tentative de connexion, on utilise le mail et le mot de passe du client ou de l'admin, on vérifie si l'utilisateur existe et ensuite on utilise la fonction bcrypt compare pour voir si le mot de passe rentré correspond à celui du compte. Si tout se déroule correctement alors un token est créé et envoyé avec les infos du compte (sauf le mot de passe)

```
//// Authentification d'un utilisateur
exports.Login = async (req, res) => {
    try {
        const { mail, mdp } = req.body;
        console.log(mail);
        // Rechercher l'utilisateur dans la BDD
        const result = await db.pool.query('Select * FROM Client WHERE mail = ?', [mail],
        function (error, results, fields) {
            if (error) throw error;
            if (results.length === 0) {
                return res.status(401).json({error: 'Utilisateur non trouvé.'});
            }

            const user = results[0];

            // Vérifier le mot de passe avec bcrypt
            const passwordMatch = bcrypt.compare(mdp, user.mdp);
            if (!passwordMatch) {
                return res.status(401).json({ error: 'Mot de passe incorrect.'})
            }

        // Générer un token JWT pour l'utilisateur nouvellement inscrit
        const token = jwt.sign({
            id: user.id,
            nom: user.nom,
            prenom: user.prenom,
            mail: user.mail,
            isAdmin: user.isAdmin,
        }, process.env.API_KEY, { expiresIn: '1h' });

        // Envoyer le token et le nom en réponse
        res.json({ token, isAdmin: user.isAdmin, nom: user.nom });
    }
        )}
    catch (err) {
        console.log(err);
        res.status(500).json({ error: "Erreur lors de la connexion." });
    }
}
```

3) Test Unitaire

```
const request = require('supertest');
const app = require('./server');

describe('Test Route GET', () => {
    it('Recuperation Produits', async () => {
        const res = await request(app)
    .get('api/produits/allProducts')
    expect(res.status).toEqual(200);
    });
})

describe('Test Route POST', () => {
    it('Création d\'un produit', async () => {
        const res = await request(app)
        .post('/api/produits/addProduct')
        .send({
            nom: "Produit",
            prix: 60,
            description: "description",
            image:"image.png",
            stock: 1
        })
        expect(res.status).toEqual(200);
        expect(res.body).toEqual(1);
    });
});
```

```
describe('Test Route PUT', () => {
    it('Modification d\'un produit', async () => {
        const res = await request(app)
        .post('/api/produits/editProduct')
        .send({
            id: 1,
            nom: "Produit modifié "
        })
        expect(res.status).toEqual(200);
    });
});

describe('Test Route DELETE', () => {
    it('Suppression d\'un produit', async () => {
        const res = await request(app)
        .post('/api/produits/deleteProduct')
        .send({
            id: 1
        })
        expect(res.status).toEqual(200);
    });
});
```

Les tests unitaires servent à vérifier le bon fonctionnement de nos routes en asynchrone,

je présente ici comment ceux-ci fonctionne pour la gestion des produits.