Documentation Technique M2L

Maison des Ligues

Technologies Utilisées

- Langage de programmation : JavaScript.
- Framework : Express.js
- Base de données : Identifier la base de données utilisée (MySQL).
- Dépendances :
- Js-cookie pour la gestion des sessions et la personnalisation du contenu
- bcrypt pour le hachage des mots de passe
- jsonwebtoken pour la gestion des tokens JWT
- crypto pour la génération d'UUID cryptographiquement sécurisés

Formulaire d’inscription

Maison des Ligues

Le composant InscriptionForm utilise le hook useState pour gérer les données saisies dans le formulaire, le nom, l'e-mail et le mot de passe. Lorsque l'utilisateur remplit le formulaire, la fonction handleChange est déclenchée à chaque modification d'un champ, mettant à jour les données du formulaire en temps réel.

```
const InscriptionForm = () => {
    const [formData, setFormData] = useState({
        name: '',
        email: '',
        password: '',
    });

    Codeium: Refactor | Explain | Generate JSDoc
    const handleChange = (e) => {
        setFormData({ ...formData, [e.target.name]: e.target.value });
    }; //Christian Latour
```

La fonction handleSubmit est appelée lorsque l'utilisateur soumet le formulaire d'inscription il envoie une requête POST à l’URL /api/usersroute/inscription avec les informations de l’user grâce a formData

```
const handleSubmit = async (e) => {
    e.preventDefault();

    try {
        const response = await fetch('http://localhost:3000/api/usersroute/inscription', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(formData),
        });

        if (response.ok) {
            window.location.href = '/connexion';
        } else {
            console.error('Erreur lors de la soumission du formulaire :', response.statusText);
        }
    } catch (error) {
        console.error('Erreur lors de la soumission du formulaire :', error);
    }
};

return (
    <form onSubmit={handleSubmit}>
        <label>Nom:</label>
        <input type="text" name="name" value={formData.name} onChange={handleChange} required />

        <label>Email:</label>
        <input type="email" name="email" value={formData.email} onChange={handleChange} required />

        <label>Mot de passe:</label>
        <input type="password" name="password" value={formData.password} onChange={handleChange} required autoComplete="current-password" />

        <button type="submit">S'inscrire</button>
    </form>//Christian Latour
);
```

Controller Inscription

Maison des Ligues

const { name, email, password } = req.body;: extrait les données du formulaire d'inscription envoyées par le front-end

const uid = crypto.randomUUID(); : génère un identifiant unique universel de manière sécurisée. Cet UUID est utilisé pour identifier de manière unique chaque utilisateur dans la base de données.

const hashedPassword = await bcrypt.hash(password, saltRounds); : utilise la fonction hash de la bibliothèque bcrypt pour hacher le mot de passe de l'utilisateur. Le mot de passe haché est stocké de manière sécurisée dans la base de données, et Le salage est une technique de sécurité qui ajoute une chaîne de caractères aléatoire au mot de passe avant de le hacher, ce qui renforce la sécurité.

```
exports.inscription = async (req, res) => {
    const { name, email, password } = req.body;
    const uid = crypto.randomUUID();
    try {
        const saltRounds = 10;
        const hashedPassword = await bcrypt.hash(password, saltRounds);
        const [result] = await db.execute(
            'INSERT INTO users (uid, name, email, password) VALUES (?, ?, ?, ?)',
            [uid, name, email, hashedPassword]
        );
        res.status(200).json({ message: 'Inscrit avec succès' });
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Une erreur s\'est produite' });
    } //Christian Latour
```

const [result] = await db.execute(...); exécute une requête SQL pour insérer les informations de l'utilisateur (UUID, nom, e-mail et mot de passe haché) dans la base de données. La requête SQL est paramétrée pour éviter les injections SQL.

Formulaire de connexion

Maison des Ligues

Const Connexion : utilise le hook useNavigate pour faciliter la navigation entre les pages. Il utilise également l'état local pour stocker les données du formulaire , e-mail et le mot de passe, ainsi que les erreurs de connexion. La fonction handleChange est déclenchée à chaque modification d'un champ du formulaire, mettant à jour les données du formulaire

```
const Connexion = () => {
    const navigate = useNavigate();


    const [formData, setFormData] = useState({
        email: '',
        password: '',
    });
    const [error, setError] = useState(null);

    Codeium: Refactor | Explain | Generate JSDoc
    const handleChange = (e) => {
        setFormData({ ...formData, [e.target.name]: e.target.value });
    }; //Christian Latour
```

La fonction handleSubmit gère la soumission du formulaire de connexion en envoyant les données au back-end avecune requête POST à l'URL 'api/usersroute/connexion' . En cas de succès, elle stocke le token JWT dans un cookie et déclenche un événement tokenUpdated

```
const handleSubmit = async (e) => {
    e.preventDefault();

    try {
        const response = await fetch('http://localhost:3000/api/usersroute/connexion', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            credentials: 'include',
            body: JSON.stringify(formData),
        });

        if (response.ok) {
            const { token } = await response.json();

            Cookies.set('token', token);


            window.dispatchEvent(new Event('tokenUpdated'));
            handleTokenUpdate(token); //Christian Latour
```

```
<form onSubmit={handleSubmit}>
    <label>Email:</label>
    <input
        type="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
        required
    />

    <label>Mot de passe:</label>
    <input
        type="password"
        name="password"
        value={formData.password}
        onChange={handleChange}
        required
    />

    <button type="submit">Se connecter</button>
```

Formulaire de connexion-Controller Connexion

Maison des Ligues

- La fonction handleTokenUpdate décode un token JWT reçu en paramètre pour extraire les informations sur l'utilisateur. Si l'utilisateur est un administrateur, il est redirigé vers la page du tableau de bord de l'administrateur, sinon il est redirigé vers la page d'accueil.

```
const handleTokenUpdate = (token) => {
    const decodedToken = jwtDecode(token);
    console.log('Decoded Token:', decodedToken);
    if (decodedToken.isAdmin) {
        navigate('/admin/dashboard');
        window.location.reload();
    } else {
        navigate('/acceuil');
    }
};//Christian Latour
```

Cette fonction gère les demandes de connexion des utilisateurs.

Elle récupère les données d'identification (e-mail et mot de passe) fournies par l'utilisateur dans le corps de la requête.

En utilisant ces informations, elle interroge la base de données pour trouver un utilisateur correspondant à l'e-mail fourni. Avec const [result] = await connection.query

Elle compare le mot de passe fourni avec celui stocké dans la base de données pour vérifier son exactitude, en utilisant la fonction de hachage bcrypt.

Si le mot de passe correspond, elle génère un token JWT signé contenant des informations sur l'utilisateur et son statut d'administrateur , Ce token est renvoyé au client dans la réponse

```
exports.connexion = async (req, res) => {
    const { email, password } = req.body;
    try {
        const connection = await db.getConnection();
        const [result] = await connection.query(
            'SELECT * FROM users WHERE email = ?',
            [email]
        );
        connection.release();
        if (result.length === 0) {
            return res.status(401).json({ error: 'Utilisateur non trouvé' });
        }
        const user = result[0];
        const passwordMatch = await bcrypt.compare(password, user.password);
        if (!passwordMatch) {
            return res.status(401).json({ error: 'Mot de passe incorrect' });
        }
        const isAdmin = user.admin === 1;
        const token = jwt.sign({
            uid: user.uid,
            email: user.email,
            isAdmin: isAdmin
        }, process.env.API_KEY, { expiresIn: '1h' });
        res.json({ token });   //Christian Latour
    } catch (error) {
```

Deconnexion

Maison des Ligues

La fonction handleLogout est appelée lorsqu'un utilisateur souhaite se déconnecter.

Elle envoie une requête POST à l'endpoint /api/usersroute/deconnexion de votre serveur, qui gère la déconnexion.

Dans cette requête, aucun corps de message JSON n'est nécessaire, car la déconnexion ne nécessite pas de données supplémentaires.

Si la réponse du serveur est positive (code de statut 200), cela signifie que l'utilisateur a été déconnecté avec succès. Le token JWT stocké dans les cookies du navigateur est supprimé à l'aide de la bibliothèque Cookies.

```
const handleLogout = async () => {
  try {
    const response = await fetch('http://localhost:3000/api/usersroute/deconnexion', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },

      body: JSON.stringify({}),
    });
    if (response.ok) {
      Cookies.remove('token');
      window.location.href = '/';
    } else {                          //Christian Latour
```

```
exports.deconnexion = async (req, res) => {
    try {

        res.clearCookie('token');
        res.status(200).json({ message: 'Déconnexion réussie' });
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Une erreur s\'est produite lors de la déconnexion' });
    }//Christian Latour
};
Codeium: Refactor | Explain | Generate JSDoc
```

L'endpoint /api/usersroute/deconnexion est associé à la fonction exports.deconnexion.
Cette fonction est exécutée lorsqu'une requête POST est reçue à cet endpoint.
Elle utilise res.clearCookie('token') pour supprimer le cookie JWT nommé 'token' du navigateur de l'utilisateur.
Ensuite, elle renvoie une réponse avec le code de statut 200 et un message indiquant que la déconnexion a réussi.

Affichage des produit

Maison des Ligues

La fonction Shop charge les produits depuis l'API lorsque la page de la boutique est chargée pour la première fois, puis les stocke localement pour être affichés sur la page. La fonction Shop appelle le chemin /api/produitsroute/produit pour récupérer les produits depuis l'API.

```
function Shop() {
    const [products, setProducts] = useState([]);
    const [quantities, setQuantities] = useState({});

    useEffect(() => {
        axios.get('http://localhost:3000/api/produitsroute/produit')
            .then(response => {
                console.log("Products trouvé :", response.data);
                setProducts(response.data);
            })
            .catch(error => {
                console.error('Error  products:', error);
            });//Christian Latour

    }, []);
```

la fonction affichéproduit renvoie la liste des produits stockés dans la base de données. Elle exécute une requête SQL pour sélectionner tous les produits depuis la table products de la base de données.

```
exports.afficheproduit = async (req, res) => {
    try {
        const [result] = await db.query('SELECT * FROM products');
        res.status(200).json(result);
    } catch (error) {
        console.error(error);
        res.status(500).send({ error: 'Une erreur s\'est produite' });
    } //Christian Latour
```

Affichage des produit du panier de l'user

Maison des Ligues

La fonction fetchPanier est chargée de récupérer les produits du panier à partir de l'API

const token = Cookies.get('token'); = Obtention du token d'authentification

const decodedToken = jwtDecode(token); = Décodage du token

response=awaitaxios.get 'api/produitsroute/panier', { params: { uid: decodedToken.uid } = Requête à l'API serveur en incluant l'identifiant de l'utilisateur comme paramètre dans la requête.

Mise à jour du panier : setPanier

la fonction retourne l'interface utilisateur du panier. Cette interface affiche les produits du panier sous forme de liste, avec leur image, nom, prix, quantité et un bouton pour les supprimer du panier. Un bouton supplémentaire en bas de la liste permet de confirmer le panier, affichant un message avec le montant total lorsque cliqué.

```
const fetchPanier = async () => {
    try {
        const token = Cookies.get('token');
        if (!token) {
            throw new Error('Token non trouvé');
        }

        const decodedToken = jwtDecode(token);

        const response = await axios.get('http://localhost:3000/api/produitsroute/panier', {
            params: { uid: decodedToken.uid }
        });
        setPanier(response.data);//Christian Latour

    } catch (error) {
        console.error('Erreur lors de la récupération du panier :', error);
    }
```

```
return (
    <div className="panier-container">
        <h1>Contenu du panier :</h1>
        <ul>
            {panier.map((produit, index) => (
                <li key={index} className="produit">
                    <div className="produit-info">
                        <img src={`http://localhost:3000/${produit.image}`} alt={produit.nom} />
                        <span className="produit-name">{produit.name}</span>
                        <span className="produit-price">{produit.price} €</span>
                        <span className="produit-quantity">{produit.quantity}</span>
                        <button onClick={() => handleDeleteProduct(produit.pid)}>Supprimer</button>
                    </div>
                </li>
            ))}
        </ul>
        <button onClick={() => alert(`Le montant total est de ${total} €`)}>Confirmer le panier</button>
    </div>
);
```

Affichage des produit du panier de l’user –Controller

Maison des Ligues

La fonction getContenuPanier est une route API qui permet de récupérer le contenu du panier d'un utilisateur spécifique

```
exports.getContenuPanier = async (req, res) => {
    const { uid } = req.query;
    try {
        const [panierRows] = await db.execute('SELECT * FROM panier WHERE uid = ?', [uid]);
        res.status(200).json(panierRows);
    } catch (error) {
        console.error('Erreur lors de la récupération du contenu du panier:', error);
        res.status(500).json({ error: 'Une erreur s\'est produite lors de la récupération du contenu du panier'
    } //Christian Latour
```

Extraction de l'identifiant de l'utilisateur
const { uid } = req.query;

Requête à la base de données : requête SQL pour sélectionner toutes les lignes de la table panier où l'identifiant de l'utilisateur correspond à celui fourni dans la requête. 'SELECT * FROM panier WHERE uid = ?', [uid]);

Réponse à la requête : Si la requête SQL réussit, la fonction renvoie les données du panier récupérées = Res.status (200)

Ajout de produit au panier

Maison des Ligues

La fonction addToCart est une fonction
qui gère l'ajout d'un produit au panier.
Voici un résumé de son fonctionnement :

```
const addToCart = async (product) => {
    try {
        const token = Cookies.get('token');
        if (!token) {
            throw new Error('Token not found');
        }
        const decodedToken = jwtDecode(token);
        console.log('Decoded Token:', decodedToken);
        const quantity = quantities[product.pid] || 1;
        const data = {
            pid: product.pid,
            uid: decodedToken.uid,
            name: product.name,
            details: product.details,
            price: product.price,
            image: product.image,
            quantity: quantity
        };
        const response = await fetch('http://localhost:3000/api/produitsroute/ajout', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${token}`
            },
            body: JSON.stringify(data)
        }); //Christian Latour
        if (response.ok) {
            console.log('Product added to cart successfully');
            window.location.reload();
        } else {
            console.error('Failed to add product to cart');
        }
    } catch (error) {
        console.error('Error while adding to cart:', error);
    }
};
```

Extraction du token d'authentification : La fonction extrait le token d'authentification de l'utilisateur à partir des cookies.

Vérification de l'existence du token : Ensuite, elle vérifie si le token a été trouvé. Si aucun token n'est trouvé, elle lance une erreur.

Décodage du token : décrypte le token pour obtenir les informations utilisateur, telles que l'identifiant de l'utilisateur associé.

Récupération de la quantité : La fonction récupère la quantité du produit à ajouter au panier à partir de l'état local ou définit une quantité par défaut de 1 si aucune quantité n'est spécifiée.

Création des données à envoyer : Elle crée un objet contenant les données du produit à ajouter au panier, y compris l'identifiant de l'utilisateur, les détails du produit et la quantité.

Envoi de la requête au serveur : Elle envoie une requête POST au serveur (/api/produitsroute/ajout) avec les données du produit à ajouter au panier. Elle inclut le token d'authentification dans les en-têtes de la requête pour l'authentification.

Controller Ajout de produit au panier

Maison des Ligues

1- extraction des données du produit :
On extrait les données du produit telles que l'identifiant (pid), l'identifiant de l'utilisateur (uid), le nom (name), les détails (details), le prix (price), l'image (image) et la quantité (quantity) à partir du corps de la requête. : const { pid, uid, name, details, price, image, quantity } = req.body;

```
exports.ajouterAuPanier = async (req, res) => {
    const { pid, uid, name, details, price, image, quantity } = req.body;

    if (!pid || !uid || !name || !details || !price || !image || !quantity) {
        return res.status(400).json({ error: 'Paramètres manquants' });
    }
    try {
        const [productRow] = await db.execute('SELECT * FROM products WHERE pid = ?', [pid]);
        if (productRow.length === 0) {
            return res.status(404).json({ error: 'Produit non trouvé' });
        }
        if (productRow[0].quantity < quantity) {
            return res.status(400).json({ error: 'La quantité demandée est supérieure à la quantité disponible' });
        }
        const newQuantity = productRow[0].quantity - quantity;
        await db.execute('UPDATE products SET quantity = ? WHERE pid = ?', [newQuantity, pid]);
        await db.execute('INSERT INTO panier (pid, uid, name, details, price, image, quantity) VALUES (?, ?, ?, ?, ?, ?, ?)', [pid, uid, name, details, price, image, quantity]);
        res.status(200).json({ message: 'Produit ajouté au panier avec succès' });   //Christian Latour
    } catch (error) {
        console.error('Erreur lors de l\'ajout du produit au panier:', error);
        res.status(500).json({ error: 'Une erreur s\'est produite lors de l\'ajout du produit au panier' });
    }
};
```

2-Vérification de l'existence du produit :
On vérifie si le produit existe dans la base de données en recherchant son identifiant. : const [productRow] = await db.execute('SELECT * FROM products WHERE pid = ?', [pid]);

3-On vérifie si la quantité demandée du produit est disponible en stock. if (productRow[0].quantity < quantity) {return res.status(400).json({ error: ‘message d’erreur indisponible ' }); Si la quantité est disponible, on met à jour la quantité en stock dans la base de données const newQuantity = productRow[0].quantity - quantity;
await db.execute('UPDATE products SET quantity = ? WHERE pid = ?', [newQuantity, pid]);

Controller Ajout de produit au panier

Ajout du produit au panier :

On ajoute le produit au panier dans la base de données avec la quantité spécifiée.

await db.execute('INSERT INTO panier (pid, uid, name, details, price, image, quantity) VA

```
    await db.execute('INSERT INTO panier (pid, uid, name, details, price, image, quantity) VALUES (?, ?, ?, ?, ?, ?, ?)', [pid, uid, name, details, price, image, quantity]);
    res.status(200).json({ message: 'Produit ajouté au panier avec succès' });   //Christian Latour
} catch (error) {
    console.error('Erreur lors de l\'ajout du produit au panier:', error); //Christian Latour
    res.status(500).json({ error: 'Une erreur s\'est produite lors de l\'ajout du produit au panier' });
}
```

Supprimer du Panier

La fonction handleDeleteProduct est une fonction utilisée pour gérer la suppression d'un produit du panier utilisateur.

Récupération du token d'authentification : La fonction récupére le token d'authentification à partir des cookies de l'utilisateur. Ce token est utilisé pour autoriser la requête de suppression du produit du panier.

Envoi de la requête DELETE : la fonction envoie une requête DELETE à l'URL pour supprimer le produit du panier. Elle inclut le token d'authentification dans les en-têtes de la requête pour l'authentification.

Données de la requête : Les données de la requête contiennent l'identifiant du produit à supprimer (pid) ainsi que l'identifiant de l'utilisateur associé au panier (uid). Ces données sont envoyées dans le corps de la requête.

Traitement de la réponse : La fonction attend la réponse de la requête DELETE. Si la réponse indique que la suppression a été effectuée avec succès (statut de réponse 200),.

```
const handleDeleteProduct = async (pid) => {
    try {
        const token = Cookies.get('token');
        const response = await axios.delete(`http://localhost:3000/api/produitsroute/suprimerpanier`, {
            headers: {
                Authorization: `Bearer ${token}`
            },
            data: {
                pid: pid,
                uid: panier[0].uid
            }
        });                                          //Christian Latour
        if (response.ok) {
            setPanier(panier.filter(product => product.pid !== pid));
        } else {
            console.error('Erreur lors de la suppression du produit du panier :', response.statusText);
        }
    } catch (error) {
        console.error('Erreur lors de la suppression du produit du panier :', error);
    }
};
```

```
<button onClick={() => handleDeleteProduct(produit.pid)}>Supprimer</button>
```

CONTROLLER-Supprimer du Panier

Maison des Ligues

La fonction supprimerDuPanier est une API serveur qui permet de supprimer un produit spécifique du panier d'un utilisateur.

```
exports.supprimerDuPanier = async (req, res) => {
    const { pid, uid } = req.body;
    await db.execute('DELETE FROM panier WHERE pid = ? AND uid = ?', [pid, uid]);
    res.status(200).json({ message: 'Produit supprimé du panier avec succès' });
}; //Christian Latour
```

Extraction des données de la requête : La fonction extrait les identifiants du produit (pid) et de l'utilisateur (uid) à partir du corps de la requête HTTP. const { pid, uid } = req.body;

Suppression du produit du panier : Ensuite, la fonction exécute une requête SQL DELETE pour supprimer toutes les entrées de la table panier où l'identifiant du produit (pid) et l'identifiant de l'utilisateur (uid) correspondent aux valeurs extraites de la requête. await db.execute('DELETE FROM panier WHERE pid = ? AND uid = ?', [pid, uid]);

Ajout d’un produit Admin

Maison des Ligues

Création de l'objet FormData : Lorsque l'utilisateur remplit le formulaire pour ajouter un produit, les données saisies sont collectées dans un objet FormData. Chaque champ du formulaire (nom, détails, prix, image et quantité) est ajouté à cet objet

Envoi de la requête POST : Une fois que toutes les informations du produit sont collectées dans l'objet FormData, une requête POST est envoyée au serveur (api/produitsroute/produit). Cette requête contient l'objet FormData en tant que corps de la requête, ce qui permet d'envoyer les données du produit au serveur pour être traitées.

```
const handleAddProduct = async () => {
    const formDataToSend = new FormData();
    formDataToSend.append('name', name);
    formDataToSend.append('details', details);
    formDataToSend.append('price', price);
    formDataToSend.append('image', image);
    formDataToSend.append('quantity', quantity);

    try {
        const response = await fetch('http://localhost:3000/api/produitsroute/produit', {
            method: 'POST',
            body: formDataToSend, //Christian Latour
```

```
if (response.ok) {
    const newProduct = await response.json();
    setProducts([...products, newProduct]);

    setName('');
    setDetails('');
    setPrice('');
    setImage(null);
    setQuantity('');
```

Controller Ajout d’un produit Admin

Maison des Ligues

```
exports.ajoutproduit = async (req, res) => {
    const { name, details, price, quantity } = req.body;
    const pid = crypto.randomUUID();
    const image = req.file ? req.file.path : null;
    try {
        if (!req.user.isAdmin) {
            console.log('Seuls les administrateurs sont autorisés à ajouter des produits.');
            return res.status(403).json({ error: 'Seuls les administrateurs sont autorisés à ajouter des produits.' });
        }
        console.log('Data received for adding a product:');
        console.log('Name:', req.body.name);
        console.log('details:', req.body.details);
        console.log('Price:', req.body.price);
        console.log('Quantity:', req.body.quantity);
        console.log('Image:', req.file);
        if (!name) {                                                    //Christian Latour
            console.log('Le champ "Name" est manquant.');
            return res.status(400).json({ error: 'Le champ "Name" est manquant.' });
        }
```

Cette partie du code gère l'ajout d'un produit à la base de données lorsque l'utilisateur soumet un formulaire d'ajout de produit sur le siteExtraction des données du corps de la requête : Les données du produit à ajouter sont extraites du corps de la requête. Cela inclut le nom, la description, le prix et la quantité du produit. De plus, un identifiant unique est généré pour le produit (PID), et le chemin de l'image est extrait du fichier joint à la requête.

Vérification des autorisations : Avant de procéder à l'ajout du produit, le système vérifie si l'utilisateur qui soumet la requête est un administrateur. Seuls les administrateurs sont autorisés à ajouter des produits. Si l'utilisateur n'est pas un administrateur, un message d'erreur est renvoyé et aucun produit n'est ajouté.

Validation des données : Les données du produit sont validées pour s'assurer qu'aucun champ requis n'est manquant. Si des champs essentiels comme le nom, la description, le prix, la quantité ou l'image manquent, un message d'erreur approprié est renvoyé.

```
const [result] = await db.execute(
    'INSERT INTO products (pid, name, details, price, image, quantity) VALUES (?, ?, ?, ?, ?, ?)',
    [pid, name, details, price, image, quantity]
);
res.status(200).json({ message: 'Objet ajouté avec succès' });
```

Ajout du produit à la base de données : Une fois que toutes les données du produit sont validées, une requête SQL est exécutée pour insérer le nouveau produit dans la table des produits de la base de données. L'identifiant unique du produit, le nom, la description, le prix, le chemin de l'image et la quantité sont insérés dans la base de données.

Mofication Produit Admin

handleChange est responsable de la mise à jour de l'état local du composant en réponse aux saisies de l'utilisateur, handleUpdateProduct gère l'envoi des données mises à jour au serveur pour effectuer la modification du produit dans la base de données avec une requête PUT a l'adresse api/produitsroute/produit/${selectedProduct.pid}

```jsx
<h1>Liste des Produits :</h1>
<div className='liste'>
    {products.map(product => (
        <div key={product.pId} className="box-product">
            {product.image && <img src={`http://localhost:3000/${product.image}`} alt={product.name} style={{ maxWidth: '100%' }} />}
            <div className="details">
                <p>{product.name}</p>
                <p>{product.details}</p>
                <p>Prix : {product.price} €</p>
                <p>Quantité disponible : {product.quantity}</p>
                <button className='btn2' onClick={() => handleEditProduct(product)}>Modifier</button> //Christian Latour
```

```jsx
const handleUpdateProduct = async () => {
    const formDataToSend = new FormData();
    formDataToSend.append('name', name);
    formDataToSend.append('details', details);
    formDataToSend.append('price', price);
    formDataToSend.append('image', image);
    formDataToSend.append('quantity', quantity);

    try {
        const response = await fetch(`http://localhost:3000/api/produitsroute/produit/${selectedProduct.pid}`, {
            method: 'PUT',
            body: formDataToSend,
        });

        if (response.ok) {
            const updatedProduct = await response.json();
            const updatedProducts = products.map(product =>
                product.pid === updatedProduct.pid ? updatedProduct : product
            );
            setProducts(updatedProducts);

            setName('');
            setDetails('');
            setPrice('');
            setImage(null);
            setQuantity('');
            setSelectedProduct(null);
```

```jsx
const handleChange = (e) => {
    if (e.target.name === 'image') {
        setImage(e.target.files[0]);
    } else if (e.target.name === 'name') {
        setName(e.target.value);
    } else if (e.target.name === 'details') {
        setDetails(e.target.value);
    } else if (e.target.name === 'price') {
        setPrice(e.target.value);
    } else if (e.target.name === 'quantity') {
        setQuantity(e.target.value);
    }
}; //Christian Latour
```

Modification Controller

Maison des Ligues

Extraction des données de la requête : La fonction commence par extraire l'identifiant du produit (pid) et les autres données du produit à mettre à jour à partir des paramètres et du corps de la requête :

const { pid } = req.params;const { name, details, price, quantity } = req.body;

const image = req.file ? req.file.path : null;

Validation des données : Ensuite, la fonction vérifie la présence des données obligatoires du produit (nom, description, prix et quantité) et renvoie une erreur si l'une d'elles est manquante ;

if (!name || !details || !price || !quantity) {

return res.status(400).json({ error: 'Champs requis manquants' });

Construction de la requête SQL de mise à jour : La fonction envoie la requête SQL de mise à jour en fonction des données fournies dans la requête

let updateQuery = 'UPDATE products SET name = ?, details = ?, price = ?, quantity = ?';

const queryParams = [name, details, price, quantity];

if (image) {

updateQuery += ', image = ?';

queryParams.push(image);

updateQuery += ' WHERE pid = ?';

queryParams.push(pid);

Et exécution de la requête const [result] = await db.execute(updateQuery, queryParams);

```
exports.updateProduct = async (req, res) => {
    const { pid } = req.params;
    const { name, details, price, quantity } = req.body;
    const image = req.file ? req.file.path : null;

    try {
        console.log('ID du produit à mettre à jour :', pid);
        console.log('Nouvelles données du produit :', { name, details, price, quantity, image });
        if (!name) {
            console.log('Le champ "Name" est manquant.');
            return res.status(400).json({ error: 'Le champ "Name" est manquant.' });
        }
        if (!details) {
            console.log('Le champ "Description" est manquant.');
            return res.status(400).json({ error: 'Le champ "Description" est manquant.' });
        }
        if (!price) {
            console.log('Le champ "Price" est manquant.');
            return res.status(400).json({ error: 'Le champ "Price" est manquant.' });
        }
        if (!quantity) {
            console.log('Le champ "Quantity" est manquant.');
            return res.status(400).json({ error: 'Le champ "Quantity" est manquant.' });
        }

        let updateQuery = 'UPDATE products SET name = ?, details = ?, price = ?, quantity = ?';
        const queryParams = [name, details, price, quantity];

        if (image) {
            updateQuery += ', image = ?';
            queryParams.push(image);
        }

        updateQuery += ' WHERE pid = ?';
        queryParams.push(pid);

        console.log('Requête de mise à jour :', updateQuery);
        console.log('Paramètres de la requête :', queryParams)
        const [result] = await db.execute(updateQuery, queryParams);
```

Suppression Produit Admin

Maison des Ligues

Suppression d'un produit côté client : Cette fonction handleDeleteProduct est utilisée côté client pour supprimer un produit de la base de données. Elle envoie une requête DELETE à l'API en utilisant son identifiant (pid) a l'adresse : /api/produitsroute/produit/${pid}`,

```
const handleDeleteProduct = async (pid) => {
    try {
        const response = await fetch(`http://localhost:3000/api/produitsroute/produit/${pid}`, {
            method: 'DELETE',
        });

        if (response.ok) {
            setProducts(products.filter(product => product.pid !== pid));
        } else {
            console.error('Erreur lors de la suppression du produit :', response.statusText);
            setError('Erreur lors de la suppression du produit.');
        }                                                                    //Christian Latour

    } catch (error) {
        console.error('Erreur lors de la suppression du produit :', error);
        setError('Une erreur s\'est produite lors de la suppression du produit.');
    }
```

Supprimerproduit : supprime un produit de la base de données en utilisant son identifiant (pid). Elle commence par extraire l'identifiant du produit à partir des paramètres de la requête, puis exécute une requête SQL DELETE pour supprimer le produit correspondant dans la table products

```
exports.supprimerproduit = async (req, res) => {
    const pid = req.params.pid;
    try {
        const [result] = await db.execute(
            'DELETE FROM products WHERE pid = ?',
            [pid]
        );                                                    //Christian Latour
        if (result.affectedRows > 0) {
            res.status(200).json({ message: 'Objet supprimé avec succès' });
        } else {
            res.status(404).json({ error: 'Produit non trouvé' });
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Une erreur s\'est produite lors de la suppression du produit' });
    }
};
```

Serveur

- Importation des modules nécessaires :

express: Framework web Node.js minimal et flexible pour créer des applications web.

cors: Middleware pour permettre les requêtes cross-origin.

mysql: Pilote MySQL pour Node.js.

crypto: Module pour fournir des fonctionnalités cryptographiques.

bcrypt: Module pour le hachage de mots de passe.

multer: Middleware pour gérer les téléchargements de fichiers.

path: Module pour manipuler les chemins de fichiers.

produitsroute: Module contenant les routes liées aux produits.usersroute: Module contenant les routes liées aux utilisateurs.

cookieParser: Middleware pour analyser les cookies des requêtes entrantes.


```
const express = require('express');
const app = express();
const cors = require('cors');
const mysql = require("mysql2/promise");
const crypto = require('crypto');
const bcrypt = require('bcrypt')
const multer = require('multer');
const path = require('path');
const produitsroute = require('./routes/produitsroute');
const usersroute = require('./routes/usersroute');
const cookieParser = require('cookie-parser'); //Christian Latour
```

Serveur

Maison des Ligues

- Configuration d'Express :
- app.use(cookieParser()): Utilisation du middleware cookieParser pour analyser les cookies des requêtes entrantes.
- app.use(express.json()): Utilisation du middleware intégré express.json() pour analyser le corps des requêtes entrantes au format JSON.
- app.use(cors()): Utilisation du middleware cors pour permettre les requêtes cross-origin depuis http://localhost:3001.
- app.use('/uploads', express.static('uploads')): Définition d'un chemin statique pour servir les fichiers téléchargés à partir du répertoire uploads.
- app.use('/api/produitsroute', produitsroute): Utilisation des routes définies dans le module produitsroute pour les demandes commençant par /api/produitsroute.
- app.use('/api/usersroute', usersroute): Utilisation des routes définies dans le module usersroute pour les demandes commençant par /api/usersroute.
- Lancement du serveur :app.listen(3000, () => {}): Démarrage du serveur Express sur le port 3000. Le serveur est prêt à écouter les requêtes entrantes.

```
app.use(cookieParser());
app.use(express.json());
app.use(cors({
    origin: ['http://localhost:3001',],
    credentials: true, //Christian Latour

}));
app.use('/uploads', express.static('uploads'));
app.use('/api/produitsroute', produitsroute);
app.use('/api/usersroute', usersroute);
```

```
module.exports =
    app.listen(3000, () => {
//Christian Latour
```

Lancement du serveur :

app.listen(3000, () => {}): Démarrage du serveur Express sur le port 3000. Le serveur est prêt à écouter les requêtes entrantes.

Routes Users

Maison des Ligues

Importation des modules nécessaires :

express: Framework web Node.js minimal et flexible pour créer des applications web.

multer: Middleware pour gérer les téléchargements de fichiers.

Configuration d'Express :

app.use('/uploads', express.static('uploads')): Définition d'un chemin statique pour servir les fichiers téléchargés à partir du répertoire uploads.

Configuration du stockage des fichiers téléchargés :

Utilisation de multer.diskStorage pour spécifier le dossier de destination et le nom de fichier pour les téléchargements.

Configuration d'une instance upload de Multer avec le stockage défini.

```
const express = require('express');
const multer = require('multer');
const app = express();
const produitscontroller = require('../controllers/produitscontroller');
const usercontroller = require('../controllers/userscontroller');
const { authenticator } = require('../midleware/middleware');
const { isadmin } = require('../midleware/middleware');

const path = require('path');
app.use('/uploads', express.static('uploads'));

const storage = multer.diskStorage({
    Codeium: Refactor | Explain | Generate JSDoc
    destination: function (req, file, cb) {
        cb(null, 'uploads/');
    },
    Codeium: Refactor | Explain | Generate JSDoc
    filename: function (req, file, cb) {
        cb(null, Date.now() + path.extname(file.originalname));
        console.log(Date.now() + path.extname(file.originalname))
    },
});

const upload = multer({ storage: storage }); //Christian Latour
```

Routes Users

- Définition des routes :

- app.post('/produit',): Route pour l'ajout d'un produit. Utilise le middleware upload.single('image') pour gérer le téléchargement de l'image.
- app.get('/produit',): Route pour afficher tous les produits.
- app.delete('/produit/:pid',): Route pour supprimer un produit spécifié par son ID.
- app.post('/ajout',): Route pour l'ajout d'un produit au panier. Cette route reçoit les données du produit à ajouter au panier et appelle la fonction ajouterAuPanier du contrôleur des produits.
- app.get('/panier',): Route pour récupérer le contenu du panier.
- app.put('/produit/:pid',): Route pour mettre à jour un produit spécifié par son ID.
- app.get('/PrixTotalPanier',): Route pour récupérer le prix total du panier.
- app.delete('/suprimerpanier',): Route pour supprimer le panier.

```
app.post('/produit', upload.single('image'), produitscontroller.ajoutproduit);
app.get('/produit', produitscontroller.afficheproduit);
app.delete('/produit/:pid', produitscontroller.supprimerproduit);
app.post('/ajout', (req, res) => {···
});
app.get('/panier', produitscontroller.getContenuPanier);
app.put('/produit/:pid', upload.single('image'), produitscontroller.updateProduct);
app.get('/PrixTotalPanier', produitscontroller.getPrixTotalPanier);
app.delete('/suprimerpanier', produitscontroller.supprimerDuPanier);   //Christian Latour
```

Routes Produits

Maison des Ligues

Ce fichier définit les routes de l'API liées à la gestion des utilisateurs. Voici une explication de chaque partie :

Importation des modules nécessaires :

express: Framework web Node.js minimal et flexible pour créer des applications web.

crypto: Module intégré de Node.js pour la cryptographie.

bcrypt: Bibliothèque de hachage de mots de passe pour Node.js.

cookie-parser: Middleware pour parser les cookies des requêtes HTTP.

Configuration d'Express :

Utilisation de cookieParser() pour parser les cookies des requêtes HTTP.

Création d'un routeur Express :

const router = express.Router(): Création d'un routeur Express pour définir les routes liées aux utilisateurs.

```
const express = require('express');
const crypto = require('crypto');
const bcrypt = require('bcrypt')
const app = express();
const usercontroller = require('../controllers/userscontroller');
const { isadmin } = require('../midleware/middleware');
const { authenticator } = require('../midleware/middleware');
//Christian Latour
```

Routes Produits

- Définition des routes :

router.post('/connexion',): Route pour la connexion des utilisateurs.

router.post('/inscription',): Route pour l'inscription des utilisateurs.

router.get('/utilisateurs',): Route pour récupérer tous les utilisateurs.

router.get('/administrateurs',): Route pour récupérer tous les administrateurs.

router.post('/modifieradmin',): Route pour modifier le nom et le mail des administrateur.

router.delete('/utilisateurs/:uid',): Route pour supprimer un utilisateur spécifié par son ID.

router.post('/deconnexion',): Route pour la déconnexion des utilisateurs.

router.post('/connexionadmin',): Route pour la connexion des administrateurs.(pour l'application Mobile)

```
router.post('/connexion', usercontroller.connexion);
router.post('/inscription', usercontroller.inscription);
router.get('/utilisateurs', usercontroller.utilisateurs);
router.get('/administrateurs', usercontroller.administrateurs);
router.post('/modifieradmin', usercontroller.modifieradmin);
router.delete('/utilisateurs/:uid', usercontroller.delete);
router.post('/deconnexion', usercontroller.deconnexion);
router.post('/connexionadmin', usercontroller.connexionAdmin);
//Christian Latour
```

Controller-Produit

Maison des Ligues

ajoutproduit : Cette fonction est appelée lorsque l'utilisateur soumet un formulaire pour ajouter un nouveau produit sur le site.

afficheproduit : Cette fonction est appelée lorsque l'utilisateur accède à la page où tous les produits sont affichés. Généralement, cela se produit lorsqu'il ouvre la page d'accueil ou une page dédiée à la liste des produits.

updateProduct : Cette fonction est appelée lorsque l'utilisateur soumet un formulaire pour mettre à jour les détails d'un produit existant. Cela peut se produire sur une page d'édition de produit.

supprimerproduit : Cette fonction est appelée lorsque l'utilisateur choisit de supprimer un produit. Cela peut se faire en cliquant sur un bouton de suppression sur la page de détails du produit ou sur une page d'administration.

ajouterAuPanier : Cette fonction est appelée lorsque l'utilisateur clique sur un bouton pour ajouter un produit à son panier. Cela se produit généralement sur la page de détails du produit.

getContenuPanier : Cette fonction est appelée lorsque l'utilisateur consulte son panier. Elle est utilisée pour afficher les produits que l'utilisateur a ajoutés à son panier.

getPrixTotalPanier : Cette fonction est appelée lorsque l'utilisateur consulte son panier pour afficher le prix total des produits qu'il a ajoutés.

supprimerDuPanier : Cette fonction est appelée lorsque l'utilisateur choisit de supprimer un produit spécifique de son panier. Cela se produit généralement lorsqu'il clique sur un bouton de suppression à côté d'un produit dans son panier.

```
const { db } = require('../Database/database');
const crypto = require('crypto');

Codeium: Refactor | Explain | Generate JSDoc
exports.ajoutproduit = async (req, res) => {···
};
Codeium: Refactor | Explain | Generate JSDoc
exports.afficheproduit = async (req, res) => {···
};
Codeium: Refactor | Explain | Generate JSDoc
exports.updateProduct = async (req, res) => {···
};
Codeium: Refactor | Explain | Generate JSDoc
exports.supprimerproduit = async (req, res) => {···
};
Codeium: Refactor | Explain | Generate JSDoc
exports.ajouterAuPanier = async (req, res) => {···
};
Codeium: Refactor | Explain | Generate JSDoc
exports.getContenuPanier = async (req, res) => {···
};
Codeium: Refactor | Explain | Generate JSDoc
exports.getPrixTotalPanier = async (req, res) => {···
};
Codeium: Refactor | Explain | Generate JSDoc
exports.supprimerDuPanier = async (req, res) => {
    const { pid, uid } = req.body;
    await db.execute('DELETE FROM panier WHERE pid = ? AND uid = ?', [pid, uid]);
    res.status(200).json({ message: 'Produit supprimé du panier avec succès' });
}; //Christian Latour
```

Controller Users

Maison des Ligues

connexion : Cette fonction est appelée lorsque l'utilisateur soumet le formulaire de connexion sur le site.

inscription : Cette fonction est appelée lorsque l'utilisateur soumet le formulaire d'inscription pour créer un nouveau compte sur le site.

administrateurs : Cette fonction est appelée lorsque l'administrateur souhaite afficher la liste des utilisateurs ayant le rôle d'administrateur.

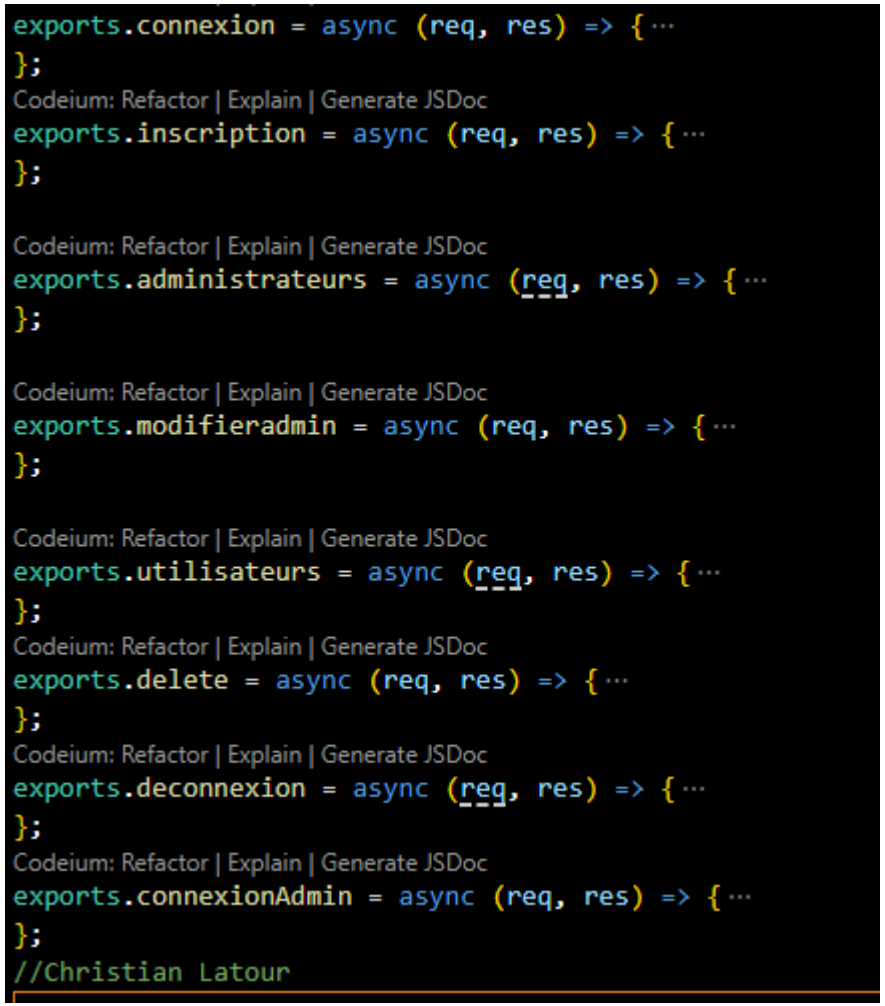
modifieradmin : Cette fonction est appelée lorsque l'administrateur souhaite modifier les détails d'un autre administrateur, comme son nom, son adresse e-mail ou son mot de passe.

utilisateurs : Cette fonction est appelée lorsque l'administrateur souhaite afficher la liste des utilisateurs qui ne sont pas des administrateurs.

delete : Cette fonction est appelée lorsque l'administrateur souhaite supprimer un utilisateur du système.

deconnexion : Cette fonction est appelée lorsque l'utilisateur ou l'administrateur se déconnecte du site. Elle peut effectuer des actions supplémentaires, comme supprimer le cookie du token côté serveur.

connexionAdmin : Cette fonction est appelée lorsque l'administrateur souhaite se connecter au site en tant qu'administrateur. Elle vérifie les informations d'identification et génère un token JWT pour l'administrateur si les informations sont correctes.



Middlware

Maison des Ligues

Middleware authenticator :

Ce middleware est une fonction qui vérifie l'authenticité des utilisateurs en utilisant JSON Web Tokens (JWT).

Il extrait le token JWT des cookies de la requête (req.cookies.token).

S'il trouve un token valide, il utilise la clé secrète stockée dans l'environnement (process.env.API_KEY) pour vérifier sa validité à l'aide de la méthode jwt.verify.

Si le token est valide, il extrait les informations de l'utilisateur (telles que son email) et les ajoute à l'objet req.user.

Si le token est invalide ou s'il n'y a pas de token, il renvoie une réponse avec le code d'erreur 401.

Middleware isadmin :

Il cherche le token JWT dans les en-têtes de la requête (Authorization) ou dans les paramètres de l'URL (token).

Comme pour le middleware authenticator, il vérifie la validité du token à l'aide de la clé secrète.

S'il trouve un token valide, il extrait l'email de l'utilisateur ainsi que son statut d'administrateur (si l'utilisateur est un administrateur ou non).

Ces informations sont ajoutées à l'objet req.user.

En fonction du statut d'administrateur extrait du token, il autorise ou non l'accès à certaines fonctionnalités réservées aux administrateurs.

```
Codeium: Refactor | Explain | Generate JSDoc
exports.authenticator = (req, res, next) => {
    const token = req.cookies.token;
    console.log('Token in authenticator:', token);
    if (token && process.env.API_KEY) {
        jwt.verify(token, process.env.API_KEY, (err, decoded) => {
            if (err) {
                console.error('Error in authenticator:', err);
                return res.status(401).json({ error: 'Invalid token' });
            } else {
                req.user = decoded;
                next();
            }
        });
    } else {
        console.log('No token provided');
        return res.status(401).json({ error: 'No token provided' });
    }
};
Codeium: Refactor | Explain | Generate JSDoc
exports.isadmin = (req, res, next) => {
    const token = req.headers.authorization || req.query.token;
    console.log('Token dans le middleware isadmin:', token);
    if (token && process.env.API_KEY) {
        jwt.verify(token, process.env.API_KEY, (err, decoded) => {
            if (err) {
                console.error('Erreur dans le middleware isadmin:', err);
                return res.status(401).json({ error: 'Token invalide' });
            } else {
                const { email, isAdmin } = decoded;
                console.log('Email extrait du token:', email);
                console.log('Statut d\'administrateur extrait du token:', isAdmin);
                req.user = { email, isAdmin };
                next();
            }
        });
    } else {
        console.log('Aucun token fourni');
        return res.status(401).json({ error: 'Aucun token fourni' });
    }
};
```

Pour activer les midlwares il faut les importer coter route et les appeler avant le controller Dans cet exemple pour supprimer un produit il faut être authentifié et admit

```
app.delete('/produit/:pid', authenticator ,isadmin, produitscontroller.supprimerproduit); //Christian Latour
```

.ENV

```
DATABASE =mligue
DATABASE_HOST =localhost
DATABASE_USER =root
DATABASE_PASSWORD =
API_KEY = mdpapimdp
```

- .env est utilisé pour stocker des variables d'environnement
- DATABASE : Cela spécifie le nom de la base de données que votre application doit utiliser. Dans ce cas, le nom de la base de données est mligue.
- DATABASE_HOST : Cela spécifie l'hôte où se trouve votre base de données. Dans ce cas, votre base de données est hébergée localement sur votre propre machine, donc l'hôte est localhost.
- DATABASE_USER : Cela spécifie le nom d'utilisateur utilisé pour se connecter à la base de données. Dans ce cas, le nom d'utilisateur est root.
- DATABASE_PASSWORD : Cela spécifie le mot de passe associé à l'utilisateur pour se connecter à la base de données. Dans ce cas, le mot de passe est vide, ce qui signifie qu'aucun mot de passe n'est défini.
- API_KEY : Cela spécifie la clé API utilisée pour signer les tokens JWT ou pour toute autre fonctionnalité d'authentification basée sur une clé API. Dans ce cas, la clé API est définie comme mdpapimdp.