Documentation Technique M2L

2023-2024

Ahamada Abdillahi

AP3 : Création du Site web M2L

Contexte

La MAISON DES LIGUES DE LA LORAINNE, établissement du Conseil Régional de Lorraine, est responsable de la gestion du service des sports et en particulier des ligues sportives ainsi que d'autres structures hébergées.
M2L souhaiterais développer un site Web type e-commerce pour la location de matériel sportif.
La solution proposée est le développement d'un site Web e-commerce en React.

Elaboration du site

Le choix de React pour le développement du frontend de notre solution e-commerce repose sur sa flexibilité, son efficacité dans la création d'interfaces utilisateur dynamiques. Parallèlement, l'utilisation d'une API REST pour le backend garantit une communication efficace et structurée entre le frontend et le serveur, facilitant l'intégration de diverses bases de données et services tiers, tout en assurant une haute performance et une grande adaptabilité à l'évolution des besoins de l'entreprise.

Plan

I. Architecture du système

- **Vue d'ensemble de l'architecture :** Schéma illustrant l'interaction entre le frontend, le backend, et les autres services externes (base de données, services de paiement, etc.).
- **Détail des composants** : Liste et description des principaux composants du système**.**

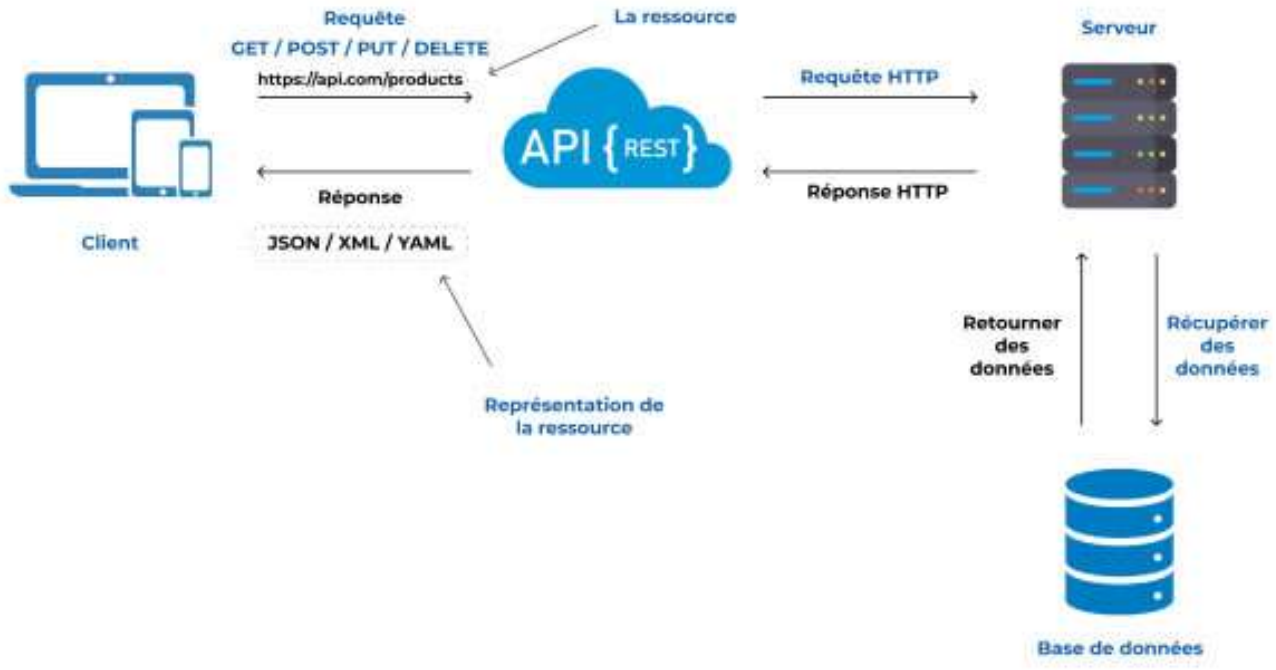
II. Développement Frontend

- **Structure du projet** : Organisation des dossiers et fichiers du projet React.
- **Composants principaux** :
    - Description des composants React utilisés.
    - Exemples de code pour les composants clés.
- **Navigation** :
    - Configuration du routage avec React Router.
    - Description des routes principales et de leur fonctionnalité.

III. Développement Backend

- **Architecture API REST** :
    - Structure générale de l'API.
    - Technologies utilisées (Node.js, base de données).
- **Gestion de la base de données** :
    - Modèle de données.
    - Schémas de la base de données et relations entre les tables.

I. Architecture du système



Le schéma ci-dessous illustre l'architecture de base d'une API REST utilisée pour faciliter la communication entre le frontend de notre site e-commerce, développé en React, et notre backend. Ce modèle d'interaction est essentiel pour la gestion des données relatives aux produits, commandes, et utilisateurs. Les clients interagissent avec l'application via des requêtes HTTP standards qui sont traitées par notre serveur, lequel récupère ou modifie les informations dans la base de données selon les besoins. Les réponses sont renvoyées au client sous divers formats, principalement en JSON, permettant une intégration facile et une grande flexibilité.

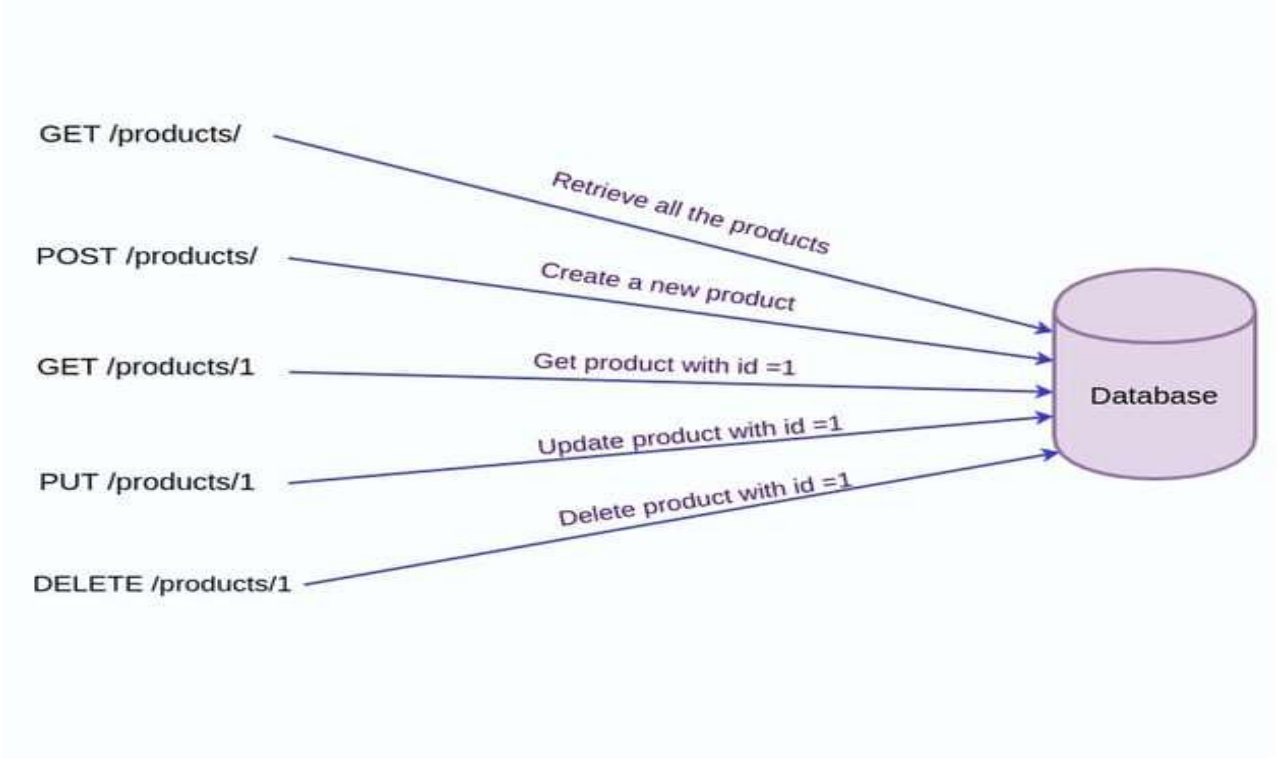
Détail de chaque composant

A. Client

- **Description** : Le client représente le frontend de l'application, où les utilisateurs interagissent avec l'interface web. Cela inclut les pages de produits, le panier, et les formulaires de connexion ou d’inscription.
- **Technologie** : Le client est construit en React, optimisé pour une interaction utilisateur réactive et dynamique.
- **Rôle** : Envoie des requêtes HTTP au serveur pour récupérer ou envoyer des données et reçoit des réponses que le frontend traite pour afficher les résultats souhaités.

B. Requêtes HTTP

- **Méthodes** : Utilise les méthodes GET pour récupérer des données, POST pour soumettre de nouvelles données, PUT pour mettre à jour des données existantes, et DELETE pour supprimer des données.
- **Fonctionnement** : Chaque requête est envoyée avec une URL spécifique et, selon la méthode, peut inclure des paramètres ou des corps de requête.



C. Serveur

- **Technologie** : Le serveur utilise Node.js avec un framework comme Express pour gérer les requêtes HTTP.
- **Rôle** : Traite les requêtes reçues, interagit avec la base de données pour effectuer les opérations demandées, et renvoie des réponses appropriées.
- **Sécurité** : Implémente des mesures de sécurité, telles que l'authentification JWT et le chiffrement HTTPS, pour protéger les données et les interactions.

D. API REST

- **Format des réponses** : Les réponses du serveur sont généralement retournées en JSON, mais peuvent également être configurées pour utiliser XML ou YAML si nécessaire.
- **Avantages** : Permet une standardisation des échanges de données, facilitant ainsi l'intégration avec divers clients ou services tiers.

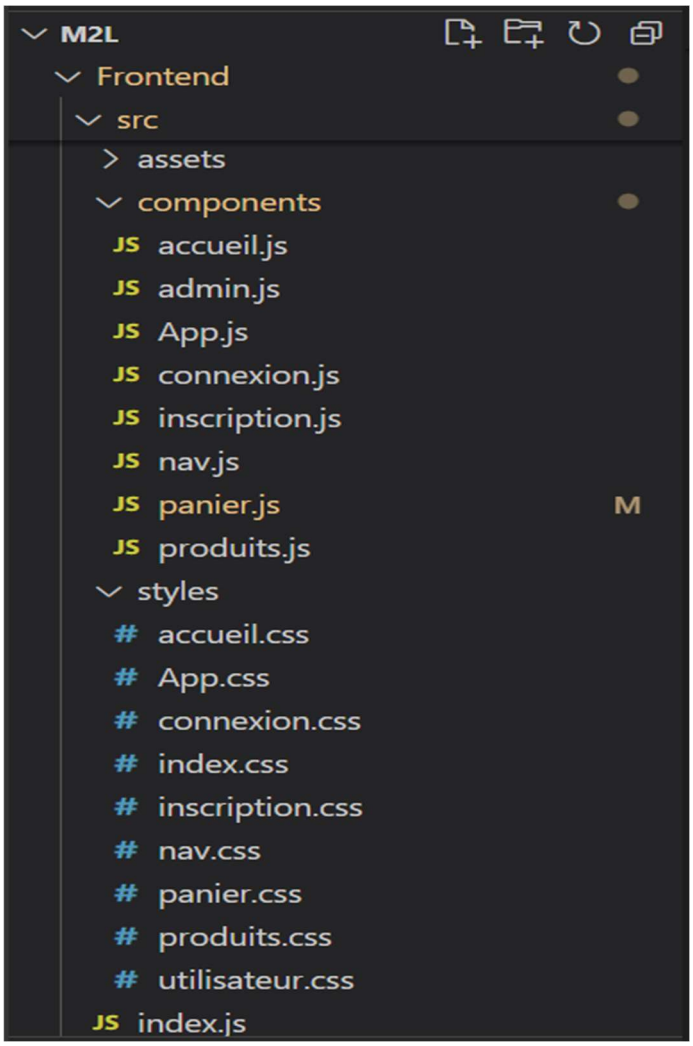
E. Base de données

- **Système utilisé** : Base de données relationnelle MySQL.
- **Rôle** : Stocke toutes les informations nécessaires à l'application, y compris les détails des produits, les informations utilisateur.
- **Interaction** : Le serveur interroge et met à jour la base de données en réponse aux actions des utilisateurs traitées par l'API.

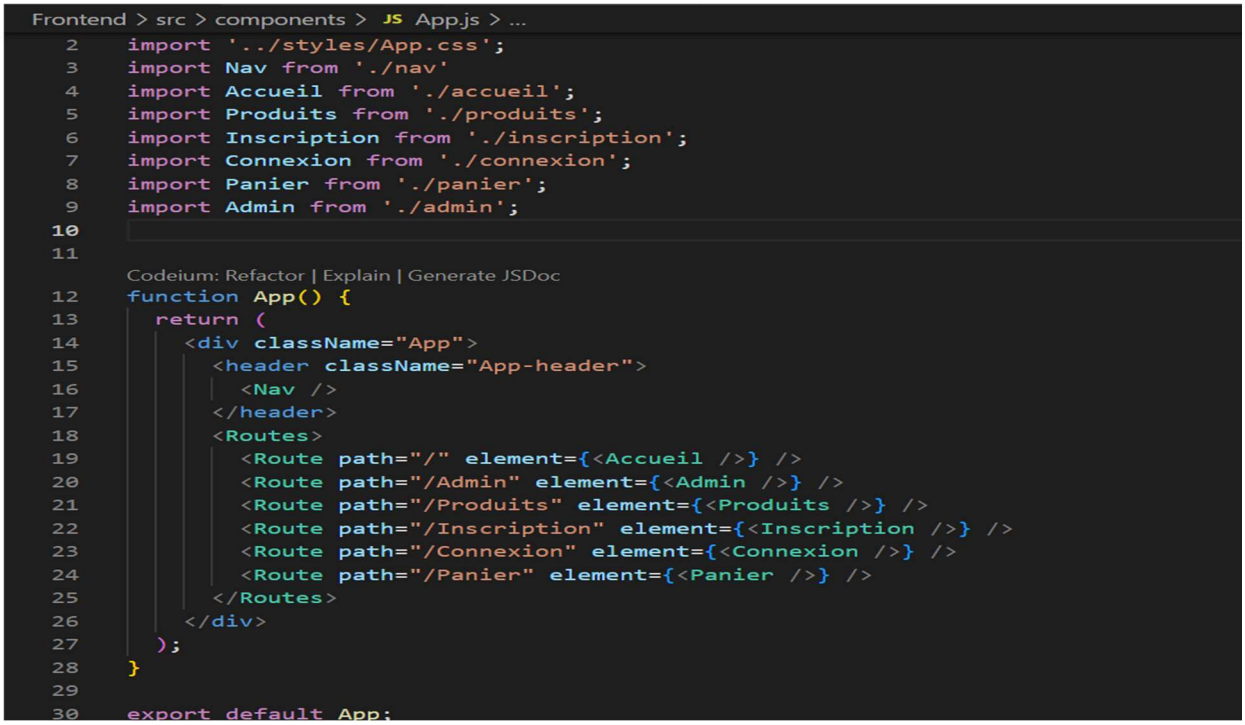
II. Développement Frontend

A. Structure du projet

- **Organisation des fichiers** : Notre projet est structuré de manière à séparer clairement les composants, les styles et les assets.
  Les composants React sont organisés dans le dossier `/components`, les styles dans `/styles`, et les images dans `/assets`.



- **Fichier principal** : Le fichier `App.js` sert de point d'entrée principal et orchestre les différents composants de l'application.



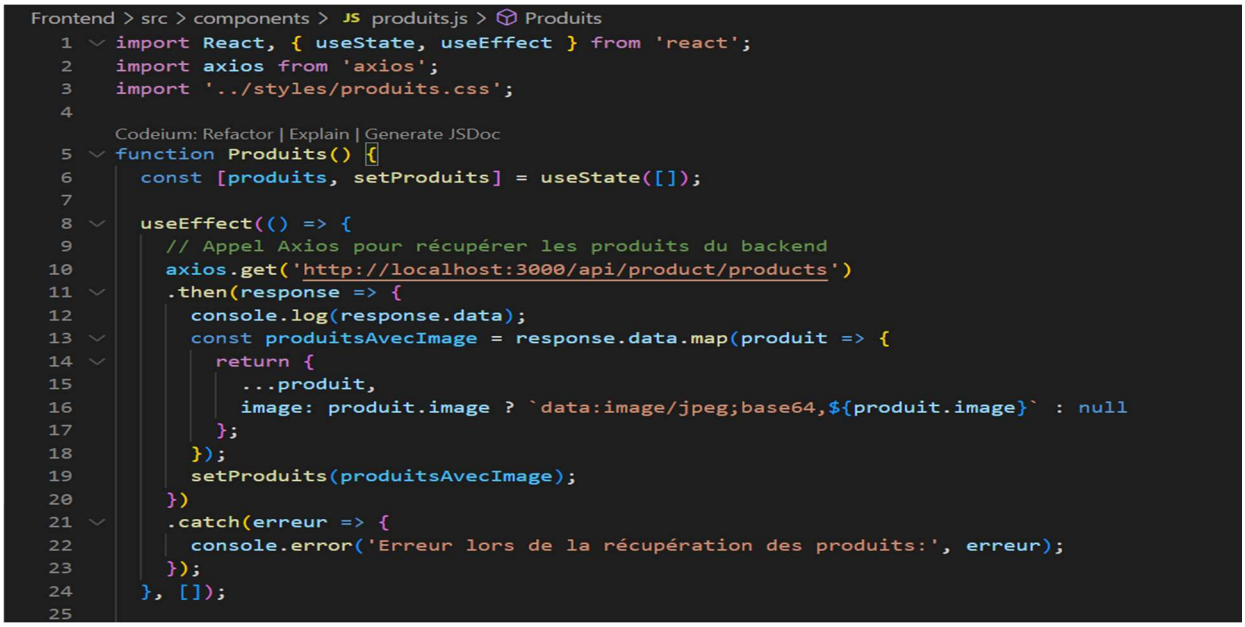
```
Frontend > src > components > JS App.js > ...
import '../styles/App.css';
import Nav from './nav'
import Accueil from './accueil';
import Produits from './produits';
import Inscription from './inscription';
import Connexion from './connexion';
import Panier from './panier';
import Admin from './admin';

Codeium: Refactor | Explain | Generate JSDoc
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <Nav />
      </header>
      <Routes>
        <Route path="/" element={<Accueil />} />
        <Route path="/Admin" element={<Admin />} />
        <Route path="/Produits" element={<Produits />} />
        <Route path="/Inscription" element={<Inscription />} />
        <Route path="/Connexion" element={<Connexion />} />
        <Route path="/Panier" element={<Panier />} />
      </Routes>
    </div>
  );
}

export default App;
```

B. Composants principaux

- **Description des composants** : Chaque composant React, comme `Nav`, `Produits`, et `Panier`, est conçu pour être réutilisable et gérer un aspect spécifique de l'interface utilisateur.
- **Exemple de code** : Par exemple, le composant `Produits` utilise un état local pour gérer la liste des produits et interagit avec le service d'API pour les récupérer.



```
Frontend > src > components > JS produits.js > Produits
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import '../styles/produits.css';

Codeium: Refactor | Explain | Generate JSDoc
function Produits() {
  const [produits, setProduits] = useState([]);

  useEffect(() => {
    // Appel Axios pour récupérer les produits du backend
    axios.get('http://localhost:3000/api/product/products')
    .then(response => {
      console.log(response.data);
      const produitsAvecImage = response.data.map(produit => {
        return {
          ...produit,
          image: produit.image ? `${produit.image}` : null
        };
      });
      setProduits(produitsAvecImage);
    })
    .catch(erreur => {
      console.error('Erreur lors de la récupération des produits:', erreur);
    });
  }, []);
```

C. Gestion des états

- **Utilisation de hooks** : Nous utilisons les hooks React, tels que `useState` et `useEffect`, pour gérer l'état local et les effets de bord dans les composants.

```
function Admin() {
    const [utilisateurs, setUtilisateurs] = useState([]);
    const [products, setProducts] = useState([]);
    const [isModalOpen, setIsModalOpen] = useState(false);
    const [currentUser, setCurrentUser] = useState(null);
    const [currentProduct, setCurrentProduct] = useState({ nom_produit: '', description: '', pr
    stock: '' });
    const [isProductModalOpen, setIsProductModalOpen] = useState(false);

    useEffect(() => {
        fetchUtilisateurs();
        fetchProducts();
    }, []);
```

- **Gestion de l'état global** : Pour les états qui nécessitent un partage à travers plusieurs composants, comme les détails de l'utilisateur et du panier.

D. Navigation

- **Configuration du routage** : React Router est utilisé pour gérer la navigation dans l'application, avec des routes définies pour chaque page principale comme la page d'accueil, la page de produits, et le panier.

```
Frontend > src > components > JS App.js > ...
1 import { Route, Routes } from 'react-router-dom';
```

```
<Routes>
  <Route path="/" element={<Accueil />} />
  <Route path="/Admin" element={<Admin />} />
  <Route path="/Produits" element={<Produits />} />
  <Route path="/Inscription" element={<Inscription />} />
  <Route path="/Connexion" element={<Connexion />} />
  <Route path="/Panier" element={<Panier />} />
</Routes>
```

- **Paramètres de route** : Les paramètres de route sont utilisés pour passer des informations, comme l'ID d'un produit, entre les pages.

III. Développement Backend

A. Architecture API REST

- **Structure générale** : Le backend est structuré autour de Server.js, qui gère les requêtes HTTP en fonction des routes définies.

Backend > JS server.js > ...

```
1   const express = require('express');
2   const app = express();
3   const { pool } = require('./database/database');
4   const bodyParser = require('body-parser');
5   const cors = require('cors');
6   const userRoute = require('./routes/userRoute');
7   const productRoute = require('./routes/productRoute');
8   const panierRoute = require('./routes/panierRoute');
9   app.use(bodyParser.json());
10  app.use(cors());
11
12  app.use('/api/user', userRoute);
13  app.use('/api/product', productRoute);
14  app.use('/api/panier', panierRoute);
15
16  module.exports = app.listen(3000, () => {
17      console.log('Server running on port 3000');
18  });
```

B. Endpoints de l'API

- **Documentation des endpoints** : Chaque endpoint est clairement documenté avec des informations sur les méthodes HTTP, les paramètres attendus, et les formats de réponse.

Backend > routes > JS userRoute.js > ...

```
1   const express = require('express');
2   const router = express.Router();
3   const userController = require('../controllers/userController');
4
5   router.get('/users', userController.getAllUsers);
6   router.put('/update/:id', userController.updateUser);
7   router.delete('/delete/:id', userController.deleteUser);
8   router.post('/connexion', userController.Login);
9   router.post('/inscription', userController.Register);
10
11
12  module.exports = router;
```

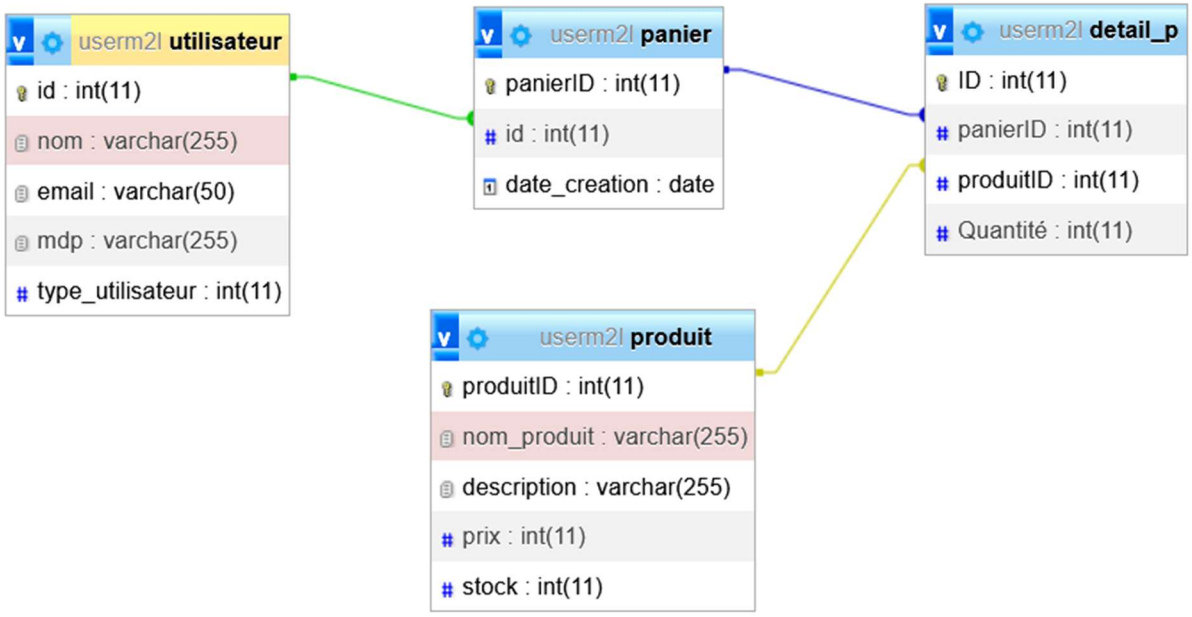
- **Sécurité des endpoints** : Des mesures telles que l'authentification par tokens et la validation des entrées sont implémentées pour sécuriser les endpoints.

```
exports.Register = async (req, res) => {
    try {
      const { nom, email, mdp } = req.body;
      try {
        const result = await pool.query('SELECT * FROM utilisateur WHERE email = ?', [email]);
        if (result.length > 0) {
          return res.status(400).json({ error: 'Cet utilisateur existe déjà.' });
        }
        const hashedPassword = await bcrypt.hash(mdp, 10);
        const insertUserQuery = 'INSERT INTO utilisateur (nom, email, mdp) VALUES (?, ?, ?)';
        const insertUserValues = [nom, email, hashedPassword];
        await pool.query(insertUserQuery, insertUserValues);
        const token = jwt.sign({ email }, process.env.API_KEY, { expiresIn: '1h' });

        res.json({ token });
      } catch (error) {
        console.log(error);
        throw error;
      }
      console.log('Inscription reussie');
    } catch (error) {
      console.error(error);
      res.status(500).json({ error: 'Une erreur est survenue lors de l\'inscription.' });
    }
  }
```

C. Gestion de la base de données

- **Modèle de données** : Un modèle relationnel est utilisé, avec des tables pour les utilisateurs, les produits, et le panier.



- **Connexion à la base de données :** Dans le projet, nous avons un dossier « database » et nous y avons ajouter un fichier qui s'appelle « database.js » qui nous permet d'effectuer la connexion.

Backend > database > JS database.js > ...

```
1   const mysql = require('mysql');
2   require('dotenv').config()
3
4   // Informations de connexion à la base de données
5   const pool = mysql.createConnection({
6     host: process.env.DB_HOST,
7     user: process.env.DB_USER,
8     password: process.env.DB_PASSWORD,
9     database: process.env.DB_NAME,
10    connectionLimit: 10,
11  });
12
13  module.exports = { pool: pool };
14
```