AP 3

Maison des Ligues

Contexte M2L

Documentation technique

Sommaire :

Présentation :

Suite à l'AP 2 , nous avons pour but de réaliser le site web e-commerce en tant qu'application lourde pour la maison des ligues de lorraine M2L.

Description :

Cela sera un site d'équipements sportifs réalisé en React Js ainsi qu'en Node Js. Les parties importantes sont une gestion des produits et des users via un compte admin et principalement la connexion et déconnexion des users et de l'admin.

Mise en place :

Nous avons choisi de vendre des équipements sportifs tels que des raquettes, des balles ainsi que des vêtements de sport. Nous avons aussi mis en place une base de données sous Heidi SQL. Un utilisateur peut donc dès maintenant s'inscrire et se connecter. Voici donc à quoi le site ressemble lorsque nous arrivons sur la page d'accueil.

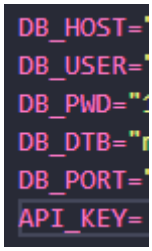
Afin d'avoir des fonctionnalités fonctionnelles nous avons dû configurer notre API coté BACKEND en créant des Routes

Tout d'abord la mise en place de la sécurisation de notre base de données Maria DB

```
const pool = mariadb.createPool({
    host: process.env.DB_HOST,
    database: process.env.DB_DTB,
    user: process.env.DB_USER,
    password: process.env.DB_PWD,
    port: process.env.DB_PORT,
    connectionLimit: 100, // Adjust the connection limit as needed
    acquireTimeout: 30000, // Increase the timeout value
});

module.exports = { pool: pool};
```

Cela permet de masquer les informations au cas ou quelqu'un ait le code source. Ces dernières sont stockés dans un fichier .env qui n'est pas rendu public



Exemple de configuration du .env

Ainsi notre base de données est sécurisée.

Nous avons une configuration de routes faite avec un controller c'est-à-dire que dans notre fichier prodroute nous avons

```js
const express = require('express')
const router = express.Router();
const prodController =  require('../controllers/prodController');
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });
router.use('/uploads', express.static('uploads'))

router.get('/produit', prodController.getAllProduits);
router.post('/produit', upload.single('image'), prodController.postP
router.put('/produit/:puid', prodController.putProdbypuid);
router.delete('/produit/:puid',prodController.deleteProd)

module.exports = router;
```

Ou nous définissons des routes . Notre dossier controller contient quand à lui les méthodes l'exemple ici est pour router.prod avec sa méthode dans prodcontroller.

```js
exports.postProd =   async (req, res) => {
    try {
        const conn = await pool.getConnection();

        if (req.file) {
            const imageFile = req.file;
            console.log('Insert request launched');
            console.log(req.body);
            /*console.log("image",imageFile)*/
            const request = 'INSERT INTO produit (puid, nom, description, prix, quantite, img) VALUES (?,?,?,?,?,?);';
            const rows = await conn.query(request, [
                crypto.randomUUID(),
                req.body.nom,
                req.body.description,
                req.body.prix,
                req.body.quantite,
                imageFile.path

            ]);

            console.log(rows);
            res.status(200).json(rows.affectedRows);
        } else {
            res.status(400).json({ error: 'No file uploaded.' });
        }
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: 'Error while inserting product.' });
    }
};
```

Avec tout cela nous avons donc un fichier server.js qui est allégé.

Du coté front end nous avons le formulaire d'inscription :

```jsx
import { useEffect, useState } from 'react';
import axios from 'axios';
import { Link } from 'react-router-dom';
import './CSS/Login.css'

export default function User() {
    const [formData, setFormData] = useState({ nom: '', email: '', mdp: '' });

    const [registrationMessage, setRegistrationMessage] = useState(''); // Declare registrationMess

    const handleChangeData = (e) => {
        setFormData((data) => ({ ...data, [e.target.name]: e.target.value }));
    }

    const handleCreate = async (e) => {
        e.preventDefault();
        console.log(formData);
        try {
            await axios.post('http://localhost:4000/api/user/user', formData);
            console.log("Create request executed successfully!");
            // Set registration message on successful create
            setRegistrationMessage('Inscription réussie!');
        } catch (error) {
            console.error(error);
            // Set registration message on create failure
            setRegistrationMessage('Échec de l\'inscription');
        }
    }

    const recup = async () => {
        await axios.get('http://localhost:4000/api/user/user').then((res) => {
            console.log(res);

        });
    }
```

En faisant un axios.post puis le port du serveur ainsi que le chemin d'accès à l'api nous avons le front et le back qui communiquent. Ainsi nous pouvons exécuter des requêtes. Ici par exemple avec le formulaire d'inscription le formdata correspond aux données du formulaire. Ici ce sont le nom , l'email et le mdp (mot de passe)

Et le handlechangeData nous permet via le formulaire ci -dessous d'envoyer ces informations en tant que formData et donc avec notre axios.post de pouvoir créer un utilisateur.

```
    return (
        <div className='loginsignup'>
            <div className='loginsignup-container'>
                <h1>Je suis nouveau ici</h1>
                <form className='loginsignup-fields' onSubmit={handleCreate}>
                    <input
                        onChange={handleChangeData}
                        type='text'
                        name='nom'
                        placeholder='Nom'
                        value={formData.nom}
                    />
                    <input
                        onChange={handleChangeData}
                        type='text'
                        name='email'
                        placeholder='Adresse e-mail'
                        value={formData.email}
                    />
                    <input
                        onChange={handleChangeData}
                        type='password'
                        name='mdp'
                        placeholder='Mot de passe'
                        value={formData.mdp}
                    />
                    <button type='submit'>S'inscrire</button>
                </form>
                <p className="registration-message">{registrationMessage}</p>
                <p className="loginsignup-login">
                    Déjà inscrit ? <Link to="/login" className="no-underline">Connectez-vous ici</Link>
                </p>
            </div>
        </div>
    );
```

Ici dans notre backend nous avons le code permettant de définir si une personne est admin

```
exports.conn = async (req, res) => {
    const { email, mdp } = req.body;

    const query = 'SELECT email, mdp, admin FROM user WHERE email = ?';

    try {
        const conn = await pool.getConnection();

        const rows = await conn.query(query, [email]);

        if (rows && rows.length > 0) {
            const user = rows[0];

            const isPasswordValid = await bcrypt.compare(mdp, user.mdp);

            if (isPasswordValid) {


                if (user.admin === 1) {
                    res.status(200).json({ message: 'Utilisateur connecté en tant qu\'admin', isAdmin: true });
                } else {
                    res.status(200).json({ message: 'Utilisateur connecté en tant qu\'utilisateur', isAdmin: false });
                }
            } else {
                res.status(401).json({ message: 'Identifiants invalides' });
            }
        } else {
            res.status(401).json({ message: 'Identifiants invalides' });
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Erreur de serveur' });
    }
};
```

Cela est défini par le if user.admin qui vérifie si la valeur de l'admin n'est pas 0 ,ainsi elle donne l'information de si l'utilisateur est isAdmin ou non. Ainsi dans notre formulaire d'inscription cette valeur isAdmin est vérifiée et donc l'admin accède à son dashbord. Dans le cas contraire la personne est simplement connectée en tant qu'utilisateur.

```
return (
  <>
    {isConnected ? (
      <div>
        <h1>Vous êtes connecté</h1>
        {isAdmin ? (
          <Admin></Admin>
        ) : (
          <p>Connecté en tant qu'utilisateur</p>
        )}
        <button onClick={handleLogout}>Se déconnecter</button>
      </div>
    ) : (
      <div className='loginsignup'>
        <div className='loginsignup-container'>
          <h1>Bonjour</h1>
          <form className="loginsignup-fields" onSubmit={handleLog}>
            <input
              onChange={(e) => setFormData({ ...formdata, email: e.target.value })}
              type='email'
              name='email'
              placeholder='Adresse e-mail'
            />
            <input
              onChange={(e) => setFormData({ ...formdata, mdp: e.target.value })}
              type='password'
              name='mdp'
              placeholder='Mot de passe'
            />
            <button type='submit' className="no-underline">Se connecter</button>
          </form>
          <p className="login-message">{loginMessage}</p>
          <p className="loginsignup-login">
            Vous êtes nouveau ici ? <Link to="/signup" className="no-underline">S'inscrire</Link>
          </p>
```