Site e-commerce de la M2L – Documentation technique

Sommaire :

I) Langage utilisés.....

II) Déploiement du Backend.....

III) Fonctionnement de l'application.....

I) Langages utilisés :

JavaScript :

- Créer les différents composants du site pour définir sa structure
- Communiquer avec l'API REST afin de récupérer et d'ajouter des informations dans la base de données du site

MySQL :

- Créer la base de données du site

NodeJs :

- Créer l'API REST et permettre la liaison entre le front et le back

II) Déploiement du Backend :

La partie backend, réalisée en NodeJS, du projet est déployée sur une machine Ubuntu. En suivant les étapes suivantes :

1. Mettre à jour l'environnement Ubuntu.

sudo apt update && sudo apt upgrade

2. Installer Git, NodeJS, npm et mariadb-server sur la machine.

sudo apt install git nodejs npm mariadb-server

3. Configurer mariadb-server.

sudo mysql_secure_installation

4. Récupérer le backend sur GitHub ou GitLab et le mettre dans le dossier /opt.

Cd /opt/

Sudo git clone nomDuRepository

5. Exporter votre base de données en .sql dans la machine Ubuntu.

6. Déployer la base de données sur le server MySQL d'Ubuntu et créer un utilisateur qui aura accès à la base de données.

- *sudo mysql -u root -p*
- *CREATE DATABASE votreDatabase;*
- *use nom BDD;*
- *sudo mysql -u root -p votreDatabase < chemin/votreDatabase.sql*
- *GRANT ALL PRIVILEGES ON votreDatabase.* TO votreUser@localhost identified by "votreMotDePasse";*
- *FLUSH PRIVILEGES;*

7. Créer le fichier .env dans le dossier du projet et lui donner la configuration nécessaire.

sudo nano /opt/votreServeurNode/.env

8. Essayer de lancer votre serveur et d'y accéder via votre ordinateur.

Pour voir l'adresse IP de la machine : {ip a}. Il faut d'abord s'être mis en connexion NAT sur la configuration de la VM et avoir effectué une redirection de port vers celui sur lequel est configuré votre serveur.

Pour lancer votre serveur : npm start (à l'emplacement où se trouve votre server.js)

9. Mettre en place le fichier permettant de lancer le serveur au démarrage. Créer un fichier serveurnode.service dans le dossier etc/systemd/system/

sudo nano /etc/systemd/system/serveurnode.service

10. Dans ce fichier, modifier la configuration suivante pour qu'elle soit conforme pour votre machine. Il faut modifier le nom de l'user ainsi que les chemins vers le back de l'application.

```
[Unit]
Description=My Node.js App
After=network.target

[Service]
User=name
WorkingDirectory=/path/to/your/app
ExecStart=/usr/bin/node /path/to/your/app/server.js
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

11. Si le fichier est bien mis en place, actualisez les unités systemsd puis activez le service que vous venez de créer.

- *sudo systemctl daemon-reload*
- *sudo systemctl enable serveurnode.service*
- *sudo systemctl start serveur.service*

Démarrez le service puis redémarrer la machine. Si tout est bien réalisé, votre api se lancera au démarrage. S'il y a des erreurs, revoir les étapes précédentes.

III) Fonctionnement de l'application :

1. L'API REST :

Pour l'installation du back end il faut commencer par la création de l'API qui communiquera entre l'application et la base de données :

```
const express = require('express') // la récupération d'express
const app = express() // variable utilisant la librairie express
let cors = require('cors')

require('dotenv').config()

// connexion à la bdd
const mariadb = require('mariadb');

const pool = mariadb.createPool({
    host: process.env.DB_HOST,
    database: process.env.DB_DTB,
    user: process.env.DB_USER,
    password: process.env.DB_PWD,
});

app.use(express.json())
app.use(cors())

//Début de l'api côté produit

app.get('/produit', async(req,res) => { // affichage des produits
    let conn;
    try{
        console.log("lancement de la connexion")
        conn = await pool.getConnection();
        console.log("lancement de la requete")
        const rows= await conn.query('SELECT * FROM produit');
        console.log(rows);
        res.status(200).json(rows)
    }
    catch(err){
        console.log(err);
    }
})
```

Dans l'exemple ci-dessus, on retrouve la constante « pool » qui permet à l'aide de variable de se connecter à la base de données. Les variables étant définit dans le fichier .env dans lequel se trouve les informations de connexion de la base.

On y retrouve aussi un exemple de requête (ici un get) qui sera plus tard utilisée dans les composants de l'application. C'est dans ce fichier que sont définit toutes les routes qui communiqueront avec la base de données.

2. Modules utilisés :

Pour l'installation du front end, nous importons dans chaque composant du projet les modules externes nécessaires.

```
1  import { useParams, useNavigate } from "react-router-dom";
2  import { useEffect, useState } from 'react';
3  import { useForm } from "react-hook-form";
4  import { Link } from 'react-router-dom'
5  import axios from 'axios';
```

- useParams et useNavigate permettent respectivement d'extraire les paramètre de l'URL dans un composant de l'application et de naviguer vers une autre URL

- useEffect et useState permettent d'exécuter du code en réponse à un changement d'état ou de propriété d'un composant et de déclarer et mettre à jour des variables d'état

- useForm facilite la gestion des formulaires en React

- Link permet de créer des liens de navigation au sein de l'application

- Axios va permettre d'effectuer les requêtes http directement à l'API

Il faut bien penser à installer tous ces modules au préalable via un terminal.

3. Liaison avec l'API :

Une fois le front end et le back end installés, il faut maintenant les liés afin que l'application fonctionne comme prévu.

Pour cela, nous allons utiliser le module axios, installé plus tôt dans la création du front end.

```
const recup = async () => {
  await axios.get(`http://localhost:8000/produit`)
      .then(res => {
          console.log(res)
          setProduits(res.data)
          setAffichage(true)
      })
```

Ce module va permettre d'envoyer une requête sur le serveur où l'on retrouve le back end.

4. Connexion au site :

Lorsqu'un utilisateur se connecte au site, la condition suivante vérifie son rôle afin de le rediriger sur la section client ou administrateur selon s'il est égale à 1 ou 2. Le rôle étant définit par défaut à 1, si la valeur est nulle cela veut dire que l'utilisateur n'existe pas et le site ne lui permet pas la connexion.

```
34    const recup = async (e) => {
35      e.preventDefault()
36      await axios.post(`http://localhost:8000/connexion`,{
37        mail: mail,
38        mdp: mdp,
39      })
40      .then(res => {
41        console.log(res)
42        if (res.status === 200 & res.data.role === 1) {
43
44          alert("Vous êtes connecté")
45          navigate("/produit");
46        }
47        else if(res.status === 200 & res.data.role === 2) {
48          alert("Vous êtes connecté")
49          navigate("/adminProduit")
50        }
51        else {
52          alert("Identifiants incorrectes")
53        }
54    }).catch((error) => {console.log(error)})
55  }
```

5. Affichage des produits :

Pour afficher les produits (pour les client ou l'administrateur), on utilise un .map qui va parcourir les données de la base et les faire apparaître.

```
49              produits.map(produit => (
50                  <div key={`produit-${produit.id}`} className="box">
51                      <div className='box-title' >
52                        <img src={`${process.env.PUBLIC_URL}/images/${produit.img}`} className="prod"/>
53                      </div>
54                      <div className='box-body'>
55                          {produit.nom}
56                          <br />
57                          {produit.prix} €
58                      </div>
```

6. Modification de produit :

La fonction EditProduit est définie en tant que composant fonctionnel exporté par défaut.

La ligne suivante extrait l'ID du produit de l'URL en utilisant le hook useParams :

```
8       let { id } = useParams()
```

Le hook useForm est utilisé pour gérer les données de formulaire et obtenir la fonction handleSubmit pour la soumission du formulaire et la gestion des erreurs.

```
10      const { handleSubmit, formState: { errors } } = useForm();
```

Ensuite, des états sont initialisés pour stocker les données du produit et une fonction de récupération est définit pour récupérer les données du produit à partir de l'API et mettre à jour les états :

```
10      const { handleSubmit, formState: { errors } } = useForm();
11      let navigate = useNavigate();
12
13      const [nom, setNom] = useState("")
14      const [prix, setPrix] = useState("")
15      const [description, setDescription] = useState("")
16      const [img, setImg] = useState("")
17      const [stock, setStock] = useState("")
18
19      const recup = async () => {
20          await axios.get(`http://localhost:8000/produit/` + id)
21              .then(res => {
22                  setNom(res.data[0].nom)
23                  setPrix(res.data[0].prix)
24                  setDescription(res.data[0].description)
25                  setImg(res.data[0].img)
26                  setStock(res.data[0].stock)
27              })
28      }
```

Une fonction editProduit est également définie pour envoyer les données mises à jour du produit à l'API en utilisant la méthode PUT.

```
30      const editProduit = async () => {
31          await axios.put(`http://localhost:8000/produit/` + id, {
32              nom: nom,
33              prix: prix,
34              description: description,
35              img: img,
36              stock: stock,
37          })
38              .then(res => {
39                  console.log(res)
40                  if (res.status === 200) {
41                      alert("Envoie réussi")
42                      navigate("/adminProduit");
43                  }
44                  else {
45                      alert("Erreur d'envoi")
46                  }
47              })
48      }
```

Enfin, le rendu du formulaire est défini avec des champs de saisie pré-remplis avec les valeurs récupérées. La fonction handleSubmit est appelée lors de la soumission du formulaire pour appeler la fonction editProduit et soumettre les données au serveur.

```
59              <form onSubmit={handleSubmit(editProduit)}>
60                  <label>Nom </label>
61                  <input defaultValue={nom} onChange={(e) => setNom(e.target.value)} />
62                  <br/>
63                  <label>Prix </label>
64                  <input defaultValue={prix} onChange={(e) => setPrix(e.target.value)} />
65                  <br/>
66                  <label>Description </label>
67                  <input defaultValue={description} onChange={(e) => setDescription(e.target.value)} />
68                  <br/>
69                  <label>Chemin de l'image </label>
70                  <input defaultValue={img} onChange={(e) => setImg(e.target.value)} />
71                  <br/>
72                  <label>Stock </label>
73                  <input defaultValue={stock} onChange={(e) => setStock(e.target.value)} />
74
75                  {(errors.nom || errors.prix || errors.description || errors.img || errors.stock) ? <span>
76                      Tous les champs doivent être remplis
77                  </span> : ""}
```

7. Ajout de produit / utilisateur :

Comme pour la modification, des états sont initialisés pour stocker les données du produit

```
15  function FormBlog()
16  {
17      const [nom, setNom] = useState('')
18      const [img, setImg] = useState('')
19      const [prix, setPrix] = useState('')
20      const [stock, setStock] = useState('')
21      const [description, setDescription] = useState('')
22
```

Une fonction Ajouter est également définie pour envoyer les données du nouveau produit à l'API en utilisant la méthode POST.

```
26    const Ajouter = async () => {
27      console.log("rhet")
28      await axios.post(`http://localhost:8000/produit`, {
29          nom: nom,
30          img: img,
31          prix: prix,
32          stock: stock,
33          description: description,
34      })
```

Enfin, la fonction handleSubmit est appelée lors de la soumission du formulaire pour appeler la fonction Ajouter et soumettre les données au serveur.

```
47  return (
48  <div className="App">
49      <div className="form">
50      <form onSubmit={Ajouter}>
51          <h2> Ajouter un produit</h2>
52          <div className="input-container ic4">
53              <input name="nom" type="text" onChange={(e) => {setNom(e.target.value)}} placeholder = "Nom du produit" required />
54          </div>
55          <div className="input-container ic4">
56              <input name="img" type="text" onChange={(e) => {setImg(e.target.value)}} placeholder = "Chemin de l'image" required />
57          </div>
58          <div className="input-container ic4">
59              <input name="prix" type="text" onChange={(e) => {setPrix(e.target.value)}} placeholder = "Prix" required />
60          </div>
61          <div className="input-container ic4">
62              <input name="stock" type="text" onChange={(e) => {setStock(e.target.value)}} placeholder = "Stock" required />
63          </div>
64          <div className="input-container ic4">
65              <input name="description" type="text" onChange={(e) => {setDescription(e.target.value)}} placeholder = "Description" required />
66          </div>
67
68          <input name="AjoutBtn" type="submit" value="Ajouter"/>
69      </form>
70      </div>
71  </div>
```

8. Suppression de produit / utilisateur :

On fait appel ici à la méthode DELETE qui va supprimer les données associées à l'id passée en paramètre dans l'url.

```
14      const suppressionProduit = async () => {
15
16          await axios.delete(`http://localhost:8000/produit/` + id)
17              .then(res => {
18                  console.log(res)
19                  if (res.status === 200) {
20                      alert("Suppression réussi")
21                      navigate("/");
22                  }
23                  else {
24                      alert("Erreur de supression")
25                  }
26              })
27      }
```

En validant la suppression, cela va appeler la fonction handleSubmit qui va à son tour appeler la fonction suppressionProduit.

```
28      return (
29          <div className='container'>
30              <form className='form' onSubmit={handleSubmit(suppressionProduit)} >
31                  <h2> Confirmer la suppression ?</h2>
32                  <input type="submit" value="Valider" />
33                  <Link to="/adminProduit" className='bouton-annuler'> Annuler </Link>
34              </form>
35          </div>
36      )
37  }
```