

## MySQL :

MySQL est un système de gestion de base de données relationnelle open source, réputé pour ses performances, sa fiabilité, sa sécurité, et sa communauté active. Il offre une solution économique avec une documentation abondante, une compatibilité étendue, et une évolutivité pour gérer des volumes de données croissants.

## Node.js :

Node.js est un environnement d'exécution JavaScript côté serveur, apprécié pour ses performances élevées, son évolutivité, son écosystème vaste avec npm, et sa popularité dans le développement rapide. Il excelle dans les applications en temps réel et dispose d'une large communauté de développeurs.

## React :

React, une bibliothèque JavaScript, favorise la construction d'interfaces utilisateur avec des composants réutilisables. Ses avantages incluent un Virtual DOM pour des performances optimisées, une gestion unidirectionnelle des données, et un écosystème dynamique avec une forte communauté de support. Utilisé largement, il est aussi compatible avec React Native pour le développement d'applications mobiles.

## Fonction de connexion/inscription

```
exports.Login = async (req, res) => {
  try {
    const { mail, mdp } = req.body;

    // Rechercher l'utilisateur dans la BDD
    const result = await db.pool.query('Select id, nom, prenom, mail, mdp, isAdmin FROM Client WHERE mail = ? and mdp = ?', [mail, mdp]);

    if (result.length === 0) {
      return res.status(401).json({ error: 'Utilisateur non trouvé.' });
    }

    const user = result[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.' });
  }
}
```

Cette fonction envoie une requête qui compare le mail et le mot de passe rentré dans le formulaire de login, et suite à cela si les deux correspondent on génère un token avec les informations de l'utilisateur ( sauf le mot de passe)

```
// Enregistrement d'un nouvel utilisateur
exports.Register = async (req, res) => {
  try {
    const { nom, prenom, mail, mdp } = req.body;

    // Vérifier si l'utilisateur existe déjà dans la BDD
    const results = await db.pool.query('SELECT * FROM Client WHERE mail = ?', [mail]);

    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." });
  }
}
```

Pour l'inscription la fonction récupère les informations du formulaire d'inscription, on vérifie si cet utilisateur existe déjà à partir de son mail, on va ensuite crypter son mot de passe (10 fois) à l'aide du module bcrypt et on va effectuer une requête insert Into en générant un uuid qui permet de générer des id aléatoires sécurisés.

Le site présente aussi une gestion de produits. On a par exemple l'ajout de celui-ci

```
//// Ajout d'un 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." });
    }
  );
}
```

On récupère du formulaire les données voulus et on crée un produit en envoyant ces nouvelles informations dans la Base de données

Modification d'un produit :

```
//// 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." });
    }
  );
}
```

Cette fonction envoie un update du produit qui prend en compte les informations des produits, et qui les modifie ensuite une fois le formulaire envoyé

Le delete :

```
//// 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." });
    }
  );
}
```

Le delete lui va récupérer un id par son produit et supprimer celui-ci.

Dans les fonctions précédente on peut aussi voir la présence de gestions d'erreurs si quelque chose n'aboutit pas correctement.

Le fichier server.js lui permet de créer des raccourcis pour la connexion à la BDD

```
const mysql = require('mysql');

const pool = mysql.createConnection({
  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
})

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

// ROUTES PRODUIT
const produitRoute = require('./routes/produitRoute');
app.use('/api/produit', produitRoute);

// ROUTES UTILISATEUR
const utilisateurRoute = require('./routes/utilisateurRoute');
app.use('/api/utilisateur', utilisateurRoute);

app.listen(8000, () => {
  console.log("Serveur à l'écoute")
})
```

Les données de pool sont stockés dans le .env et les routes font appels à d'autres routes qui vont chercher les différentes fonctions du CRUD dans les controllers

Pour la BDD on a deux tables, Produits et Clients qui comprend en fait tout les utilisateurs ainsi que les admins, le rôle permet de trier leurs accès, notamment grâce aux fonctions isLoggedIn et isAdmin dans le composants Navbar :

```

export default function Navbar() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [isAdmin, setIsAdmin] = useState(false);

  useEffect(() => {
    // Vérifier si l'utilisateur est connecté en vérifiant la présence du cookie 'token'
    const token = Cookies.get('token');
    const isAdminCookie = Cookies.get('isAdmin');

    setIsLoggedIn(!!token); // Met à jour l'état en fonction de la présence du token
    setIsAdmin(isAdminCookie === '1'); // Met à jour l'état isAdmin en fonction du cookie
  }, []);

  // Fonction de déconnexion
  const handleLogout = () => {
    // Supprimer les cookies et effectuer d'autres étapes de déconnexion si nécessaire
    Cookies.remove('token');
    Cookies.remove('isAdmin');

    // Rafraîchir la page pour appliquer les changements
    window.location.reload();
  };













  return (
    <div>
      <div className="navbar">
        <img src='../Logo_M2L.png' alt='Logo M2L' />
        <Link to="/">Home</Link>
        { /* Afficher "Produits", "A propos", "Mon panier" uniquement si l'utilisateur n'est pas admin */ }
        { !isAdmin && (
          <>
            <Link to="/produits">Produits</Link>
            <Link to="/aboutUs">A propos</Link>
          </>
        ) }

        { isLoggedIn && !isAdmin && (
          <>
            <Link to="/commandes">Mes commandes</Link>

```

On vérifie si un token a été généré ce qui signifie que un utilisateur est connecté et pour l'autre cas on vérifie, grâce au cookie généré précédemment si il est admin (c'est une valeur binaire 1 estAdmin, 0 ne l'est pas mais lors de la création de compte 0 est assigné, les admins sont créés manuellement.

Partie BDD :

	Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/>	client	     	11	InnoDB	utf8mb3_general_ci	32,0 kio	-
<input type="checkbox"/>	produit	     	9	InnoDB	utf8mb3_general_ci	16,0 kio	-
	2 tables	Somme	20	InnoDB	latin1_swedish_ci	48,0 kio	0 o

Dans la table client, on a nom, prenom, mail, mdp, isAdmin

Dans la classe produit on a nom, prix, image, description, stock