Document Technique : Site E-commerce pour la Maison des Ligues de la Lorraine (M2L)

Introduction

Ce document technique vise à fournir une description détaillée du développement d'un site e-commerce pour la Maison des Ligues de la Lorraine (M2L). En tant qu'établissement du Conseil Régional de Lorraine, la M2L est responsable de la gestion du service des sports et de l'hébergement des ligues sportives ainsi que d'autres structures connexes. Afin de mieux répondre aux besoins de ses ligues sportives affiliées et de fournir des services supplémentaires, la M2L souhaite mettre en place une plateforme e-commerce permettant la vente en ligne de produits et services.

Contexte du Projet

La Maison des Ligues de la Lorraine (M2L), en tant qu'organisme responsable de la gestion du service des sports dans la région, doit fournir les infrastructures matérielles, logistiques et les services nécessaires à l'ensemble des ligues sportives installées. Cela inclut la mise à disposition de services et de support technique pour les ligues déjà implantées ou à venir dans la région. Dans le cadre de cette mission, la M2L souhaite mettre en place un site e-commerce afin d'offrir à ses ligues affiliées une plateforme en ligne pour la vente de produits et services liés au sport.

Objectifs du Projet

Les principaux objectifs du projet incluent :

1. Création d'une Plateforme E-commerce : Développer un site web e-commerce robuste et sécurisé qui permettra à la M2L et à ses ligues affiliées de vendre des produits et services en ligne, tels que des équipements sportifs, des vêtements, des billets d'événements, etc.

2. Facilitation des Transactions : Offrir aux utilisateurs une expérience de navigation conviviale et intuitive, permettant une recherche, une sélection et un processus d'achat efficaces, tout en assurant la sécurité et la confidentialité des transactions en ligne.

3. Intégration avec les Systèmes Existant : Intégrer le site e-commerce avec les systèmes et les processus existants de la M2L, tels que la gestion des membres, des stocks et des commandes, pour assurer une gestion cohérente et efficace des opérations.

Développement Technique

Front-end

Le développement du front-end du site e-commerce repose sur les technologies suivantes :

- React : Framework JavaScript moderne utilisé pour la construction de l'interface utilisateur interactive et réactive.

- React Router Dom : Librairie permettant la gestion de la navigation et des routes dans une application React, utilisée pour créer une expérience utilisateur fluide à travers différentes vues et pages du site.

- Axios : Bibliothèque HTTP utilisée pour effectuer des requêtes réseau côté client, notamment pour communiquer avec le back-end et récupérer des données.

Back-end

Le back-end du site e-commerce est développé en utilisant les technologies suivantes :

- Node.js : Environnement d'exécution JavaScript côté serveur utilisé pour exécuter le code JavaScript côté serveur.

- Express.js : Framework web Node.js minimaliste et flexible utilisé pour construire des applications web et des API RESTful.

- MySQL : Système de gestion de base de données relationnelle utilisé pour stocker et gérer les données des produits, des utilisateurs et des commandes.

Authentification et Sécurité

Pour l'authentification et la sécurité, les technologies suivantes sont mises en œuvre :

- Bcrypt.js : Bibliothèque utilisée pour le hachage des mots de passe afin de garantir la sécurité des données utilisateur et la protection contre les attaques par force brute.

Outils de Développement

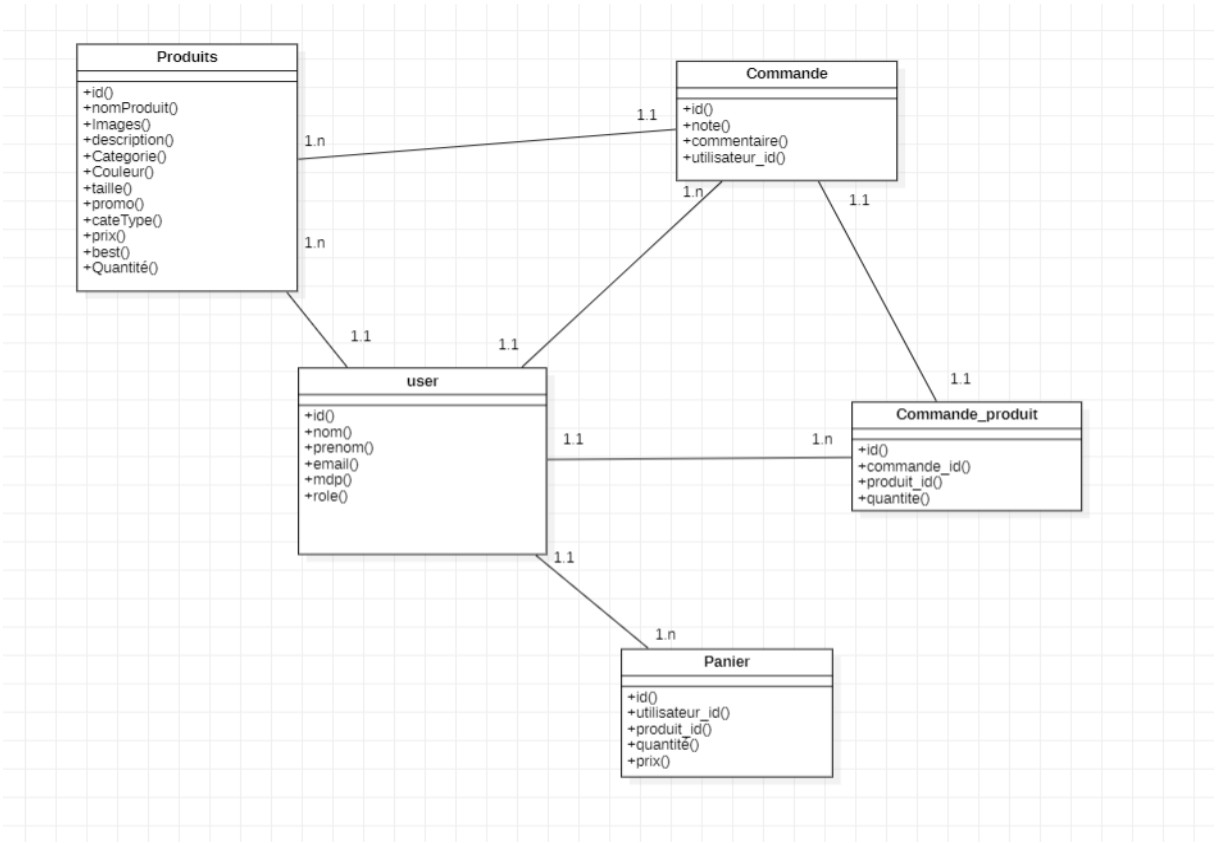
Divers outils sont utilisés pour faciliter le développement, le contrôle de version et la gestion du code source :

- Git/GitHub : Système de contrôle de version distribué utilisé pour le suivi des modifications de code et la collaboration entre les membres de l'équipe de développement.

- dotenv : Module utilisé pour charger les variables d'environnement à partir d'un fichier.env, permettant de configurer les paramètres sensibles de l'application de manière sécurisée.

Création de base de données

La décision d'utiliser MySQL comme base de données pour ce projet s'appuie sur plusieurs considérations. Tout d'abord, MySQL est l'une des bases de données relationnelles les plus largement utilisées et éprouvées dans l'industrie, offrant une fiabilité et une stabilité éprouvées. De plus, MySQL est open source, ce qui signifie qu'il est libre d'utilisation et offre une grande communauté de soutien et une abondance de ressources disponibles en ligne.



1. La table Produits : Stocke tous les détails d'un produit, y compris son nom, ses images, sa description, ses catégories, ses couleurs, ses tailles, sa disponibilité en promotion, son prix, son statut en tant que meilleur vendeur (best seller) et la quantité en stock.

2. La table Utilisateurs :Stocke les informations des utilisateurs telles que leur nom, prénom, adresse e-mail, mot de passe et leur rôle dans le système.

3. La table Commande_Produits : Stocke les associations entre les commandes et les produits, en enregistrant l'identifiant de la commande, l'identifiant du produit et la quantité commandée.

4. La table Commandes : Stocke les détails des commandes passées, y compris la note attribuée par l'utilisateur, son commentaire et son identifiant.

5. La table Panier :Stocke les articles ajoutés au panier par les utilisateurs, en enregistrant l'identifiant de l'utilisateur, l'identifiant du produit, la quantité et le prix unitaire de chaque article.

Création de l'API avec Node.js et Express

- Importation des modules

```
const express = require('express');
const app = express();
const port = 3001;
const db = require('./dbb/connexion');
const cors = require('cors');
const multer = require('multer');
const bodyParser = require('body-parser');
```

- Middleware pour afficher les fichiers statiques

```
app.use('/uploads', express.static('../sport/uploads'));
```

- Configuration des Routes

```
const userRoutes = require('./Routes/UserRoutes');
const productRouter = require('./Routes/ProductRoutes');
const panierRouter = require('./Routes/PanierRoutes');
```

```
app.use('/api/users', userRoutes);
app.use('/api/products', productRouter);
app.use('/api/cart', panierRouter);
```

- Requête pour avoir tous les produits

```
Codeium: Refactor | Explain | Generate JSDoc | ×
exports.getAllProducts = (req, res) => {
    const sql = 'SELECT * FROM produits ORDER BY id ASC';
    db.query(sql, (err, data) => {
      if (err) {
        console.error('Erreur lors de la récupération des produits :', err);
        return res.status(500).json(err);
      }
      return res.json(data);
    });
};
```

- Requête pour obtenir tous les produits en promotion

```
Codeium: Refactor | Explain | Generate JSDoc | ×
exports.getPromoProducts = (req, res) => {
    const sql = 'SELECT * FROM produits where promo > 0';
    db.query(sql, (err, data) => {
      if (err) {
        console.error('Erreur lors de la récupération des produits :', err);
        return res.status(500).json(err);
      }
      return res.json(data);
    });
};       You, 3 weeks ago • deploy
```

- Requête pour obtenir tous les produits Best Sellers

```
exports.getBestSellingProducts = (req, res) => {
    const sql = 'SELECT * FROM produits WHERE best = "best" ORDER BY id ASC ';
    db.query(sql, (err, data) => {
      if (err) {
        console.error('Erreur lors de la récupération des produits :', err);
        return res.status(500).json(err);
      }
      return res.json(data);
    });
};
```

- Requête pour obtenir tous les produits par catégorie

```
exports.getProductsByCategoryAndType = (req, res) => {
    const { categorie, typeProduit } = req.params;
    const sql = 'SELECT * FROM produits WHERE categorie = ? AND cateType = ?';
    db.query(sql, [categorie, typeProduit], (err, data) => {
      if (err) {
        console.error('Erreur lors de la récupération des produits :', err);
        return res.status(500).json(err);
      }
      return res.json(data);
    });
};
```

- Requête pour obtenir tous les produits qui contiennent le même nom

```
Codeium: Refactor | Explain | Generate JSDoc | ×
exports.getProductBysame = (req, res) => {
  const nomProduit = req.params.nomProduit;
  const sql = `SELECT * FROM produits WHERE  nomProduit = ?`;
  db.query(sql, [nomProduit], (err, data) => {
    if (err) {
      console.error('Error executing SQL query:', err);
      return res.status(500).json(err);
    }

    console.log('Query result:', data);
    return res.json(data);
  });
};
```

- Requête pour obtenir tous les catégories (homme , femme , enfant)

```
Codeium: Refactor | Explain | Generate JSDoc | ×
exports.getProductsByCategory = (req, res) => {
    const categorie = req.params.categorie;
    const sql = 'SELECT * FROM produits WHERE categorie = ?';
    db.query(sql, [categorie], (err, data) => {
      if (err) return res.status(500).json(err);
      return res.json(data);
    });
};
```

- Requête permettant d'obtenir un produit à partir de son identifiant (ID)

```
Codeium: Refactor | Explain | Generate JSDoc | ×
exports.getProductById = (req, res) => {
    const productId = req.params.id;
    const sql = 'SELECT * FROM produits WHERE id = ?';
    db.query(sql, [productId], (err, data) => {
      if (err) return res.status(500).json(err);
      return res.json(data);
    });
};
```

- Requête pour obtenir les produits de nouvelle collection

```
Codeium: Refactor | Explain | ×
exports.getNewCollection = (req, res) => {
  const nomProduit = "Under Armour Haut Zippé Tech Homme";
  const sql = 'SELECT * FROM produits WHERE nomProduit = ?';
  db.query(sql, [nomProduit], (err, data) => {
    if (err) {
      console.error('Erreur lors de la récupération des produits de la nouvelle collec
      return res.status(500).json(err);
    }
    return res.json(data);
  });
};
```

- Requête pour supprimer un produit à partir d'un ID

```
Codeium: Refactor | Explain | Generate JSDoc | ×
exports.deleteProductById = async (req, res) => {
    const productId = req.params.id;

    const sql = 'DELETE FROM produits WHERE id = ?';
    db.query(sql, [productId], (err, result) => {
      if (err) {
        console.error(err);
        res.status(500).send('Erreur lors de la suppression du produit.');
      } else {
        console.log(result);
        res.status(200).send('Produit supprimé avec succès !');
      }
    });
};
```

- Requête pour modifier les informations d'un produit

```
exports.updateProduct = async (req, res) => {
    try {
      const productId = req.params.id;
      const { nomProduit, description, categorie, couleur, taille, promo, cateType, pr
      let newImagePaths = [];

      if (req.files && req.files.length > 0) {
        newImagePaths = req.files.map((file) => file.filename);
      }

      const updateInfoSql = 'UPDATE produits SET nomProduit=?, description=?, categori
      const updateInfoParams = [nomProduit, description, categorie, couleur, taille, p

      db.query(updateInfoSql, updateInfoParams, (infoErr, infoResult) => {
        if (infoErr) {
          console.error(infoErr);
          res.status(500).send('Erreur lors de la mise à jour du produit.');
        } else {
          if (newImagePaths.length > 0) {
            const updateImagesSql = 'UPDATE produits SET images=? WHERE id=?';
            const updatedImagePaths = [...newImagePaths];

            db.query(updateImagesSql, [updatedImagePaths.join(','), productId], (image
```

```
      db.query(updateInfoSql, updateInfoParams, (infoErr, infoResult) => {
            db.query(updateImagesSql, [updatedImagePaths.join(','), productId], (image
              console.error(imageErr);
              res.status(500).send('Erreur lors de la mise à jour des images du proc
            } else {
              console.log(imageResult);
              res.status(200).send('Produit et images mis à jour avec succès !');
            }
          });
        } else {
          console.log(infoResult);
          res.status(200).send('Produit mis à jour avec succès !');
        }
      }
    });
  } catch (error) {
    console.error('Erreur lors de la modification du produit:', error);
    res.status(500).send('Erreur lors de la modification du produit');
  }
};
```

- Requête pour obtenir les produits de nouvelle collection

```
exports.createUser = async (req, res) => {
  const { nom, prenom, email, mdp } = req.body;
  const sqlUser = 'INSERT INTO user (nom, prenom, email, mdp) VALUES (?, ?, ?, ?)';

  db.query(sqlUser, [nom, prenom, email, mdp], (errUser, resultUser) => {
    if (errUser) {
      console.error('Erreur lors de l\'insertion de l\'utilisateur :', errUser);
      return res.status(500).json(errUser);
    }

    return res.status(200).json({ message: 'Utilisateur enregistré avec succès' });
  });
};
```

- Requête permettant de vérifier l'existence d'un utilisateur dans la base de données avant de lui permettre de se connecter.

```
exports.loginUser = async (req, res) => {
  const { emailUser, passwordUser } = req.body;

  if (!emailUser || !passwordUser) {
    return res.status(400).json({ message: 'Veuillez remplir tous les champs' });
  } else {
    const sql = 'SELECT * FROM user WHERE email = ?';
    db.query(sql, [emailUser], async (err, data) => {
      if (err) {
        console.error('Erreur lors de la sélection des utilisateurs :', err);
        return res.status(500).json({ message: 'Erreur serveur' });
      }

      if (data.length === 0) {
        return res.status(401).json({ message: 'Identifiants incorrects' });
      }

      const user = data[0];
      const hashedPassword = user.mdp;

      const passwordMatch = await bcrypt.compare(passwordUser, hashedPassword);
      if (!passwordMatch) {
        return res.status(401).json({ message: 'Identifiants incorrects' });
```

```
        const passwordMatch = await bcrypt.compare(passwordUser, hashedPassword);
        if (!passwordMatch) {
          return res.status(401).json({ message: 'Identifiants incorrects' });
        }

        const userData = {
          id: user.id,
          nom: user.nom,
          prenom: user.prenom,
          role: user.role
        };

        return res.json(userData);
      });
    }
  };
```

- Requête pour obtenir toutes les informations de l'utilisateur dans la base de données

```
Codeium: Refactor | Explain | Generate JSDoc | ×
exports.getUsers = (req, res) => {
  const sql = 'SELECT * FROM user';
  db.query(sql, (err, data) => {
    if (err) {
      console.error('Erreur lors de la sélection des utilisateurs :', err);
      return res.status(500).json(err);
    }
    return res.json(data);
  });
};
```

L'intégration entre le backend et le frontend

cette partie expose le code frontend illustrant la manière dont les éléments du backend sont affichés à l'aide de l'API.

```
const [bestSellers , setBestSellers] = useState([])
const [newcollection , setnewcollection] = useState([])
const API = "http://localhost:3001/api/products/best-sellers";      You, 3 weeks ago • de
const NewcollectionAPI = "http://localhost:3001/api/products/same/Under%20Armour%20Haut%20
const localhost = "http://localhost:3001"
const [category, setCategory] = useState('all');
const { addToCart: addToCartContext } = useCart()
const encodedImageUrl = encodeURIComponent("http://localhost:3001/uploads/image-1700136110


useEffect(() => {
  axios.get(NewcollectionAPI)
    .then((res) => {
      setnewcollection(res.data);
      console.log(res.data)
    })
    .catch((error) => {
      console.error('Erreur lors de la récupération des données de l\'API :', error);
    });
}, []);
```

- L'affichage des produits de nouvelle collection

```
        {newcollection.map((newcollect, index)=> (
          <>

          <div className='each-collection'>
          <Link to={`/ProduitDetails/${newcollect.nomProduit}/${newcollect.id}`}>
            <div className='prod-collection'>
            <img src={`${localhost}/uploads/${newcollect.images.split(',')[0]}`} alt=""
            </div>/.prod-collection
            </Link>
          </div>/.each-collection

          </>
        ))}
```

```
{bestSellers.map((populaire, index)=> (
    </div>/.i

    <div className='theImg'>
    {populaire .images && populaire .images.length > 0 ? (
      <>
      <Link to={`/ProduitDetails/${populaire.nomProduit}/${populaire.id}`}>
     <img src={`${localhost}/uploads/${populaire.images.split(',')[0]}`} alt=""
     <img className='BackImg' src={`${localhost}/uploads/${populaire.images.spli
     </Link>
     </>
     ) : (
    <img src={`${localhost}/uploads/default-image.jpg`} alt="Default" />
    )}
```

La sécurité de vos données

La sécurité des données revêt une importance capitale dans ce contexte pour plusieurs raisons essentielles. Tout d'abord, les utilisateurs font souvent confiance aux applications web pour protéger leurs informations personnelles, telles que les données d'identification, les informations de paiement, etc. Une faille de sécurité pourrait compromettre la confidentialité de ces données, entraînant des conséquences dommageables telles que le vol d'identité ou la fraude financière. De plus, les réglementations telles que le RGPD imposent des obligations strictes aux entreprises en matière de protection des données, avec des sanctions sévères en cas de non-respect. Enfin, la confiance des utilisateurs dans l'application est étroitement liée à sa réputation et à sa fiabilité, ce qui souligne l'importance critique de mettre en œuvre des mesures de sécurité robustes pour garantir l'intégrité et la confidentialité des données transitant par l'application.

- Hachage de mot de passe
  Les mots de passe des utilisateurs sont sécurisés en utilisant le module bcrypt pour les hacher avant de les stocker dans la base de données. Le hachage des mots de passe ajoute une couche de sécurité supplémentaire en rendant les données originales illisibles, même en cas de compromission de la base de données.

  Le mot de passe est haché côté client pour éviter de l'envoyer en clair jusqu'au serveur, de cette façon, les attaques dites de man-in-the-middle sont évitées dans ce contexte.

```
import bcrypt from 'bcryptjs';
import Cookies from 'js-cookie';
```

```
if (mdp !== mdpcheck) {
  console.error("Les mots de passe ne correspondent pas");
  return;
}
const saltRounds = 10;
const hashedPassword = await bcrypt.hash(mdp, saltRounds);

const userData = {nom, prenom, email, mdp:hashedPassword };
```

Cela aboutira à un mot de passe de 255 caractères dans la base de données et plus complexe et sécuriser

```
$2a$10$5SzhR6jy56.YOtbKlQVBEO.7K380bdvpR5q0rhPME61...
```