

# **Document Technique : Application de gestion de stock pour la Maison des Ligues de la Lorraine (M2L)**

## **Introduction**

Ce document technique vise à fournir une description détaillée du développement d'une application 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 application de gestion de stock permettant la consultation, l'ajout, la modification et la suppression des produits.

## **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 une application de gestion de stock.

## **Objectifs du Projet**

Les principaux objectifs du projet incluent :

1. Création d'une application de gestion de stock: Développer une application robuste et sécurisé qui permettra à la M2L de pouvoir consulter, ajouter, modifier et supprimer les produits dans le stock.
2. Intégration avec les Systèmes Existant : Intégrer 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 de l'application repose sur les technologies suivantes :

- Flutter : c'est un framework de développement d'applications multiplateformes créé par Google. Il permet de construire des interfaces utilisateur interactives et réactives pour les applications mobiles, web et de bureau à l'aide d'un seul codebase.
- Package HTTP : Pour la communication avec le backend, le package HTTP a été utilisé.
- Image Picker : Pour l'ajout de produits avec des images, l'image picker a été utilisé. Cette fonctionnalité permet aux utilisateurs de sélectionner des images à partir de la galerie ou de prendre une nouvelle photo avec la caméra de leur appareil.
- Widgets : Flutter utilise des widgets comme éléments de construction pour l'interface utilisateur. Ces widgets sont personnalisables et peuvent être combinés pour créer des interfaces riches et dynamiques.

### Back-end

Le back-end de l'application 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 :

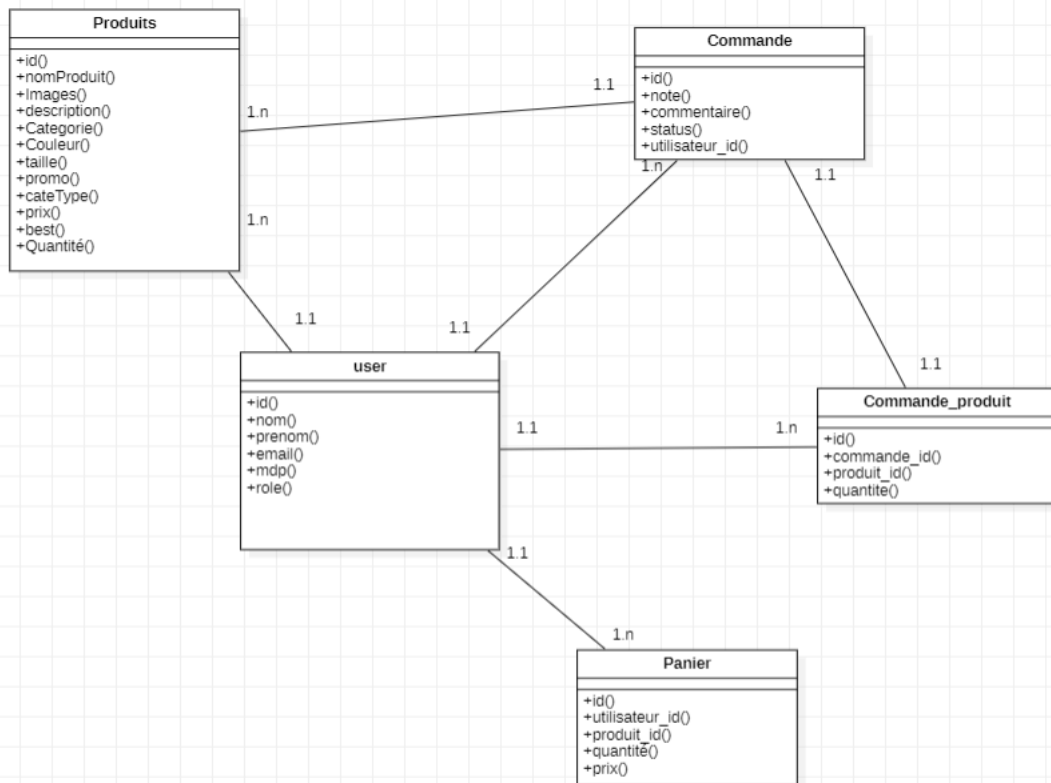
- Bcrypt.js : Bibliothèque utilisée pour le hachage des mots de passe afin de garantir la sécurité des données utilisateur et la protection contre les attaques par force brute.

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** : Stock 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 Commande** : Stock les détails des commandes passées, y compris la note attribuée par l'utilisateur, son commentaire, le statut de la commande et son identifiant.

5. **La table Panier** : Stock 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.

## 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.

- La fonction login qui me permet d'aller récupérer les données de l'utilisateur dans la base de données pour savoir si ça existe si c'est pas le cas j'ai une erreur et si c'est le cas ça me mène sur la page Dashboard

```
Future<void> login() async {
  String email = emailController.text.trim();
  String password = passwordController.text.trim();

  try {
    final response = await http.post(
      Uri.parse('http://localhost:3001/api/users/login'),
      headers: <String, String>{
        'Content-Type': 'application/json',
      },
      body: jsonEncode(<String, String>{
        'emailUser': email,
        'passwordUser': password,
      })),
    );

    if (response.statusCode == 200) {
      final data = jsonDecode(response.body);
```

```
      passwordUser: password,
    })),
  );

  if (response.statusCode == 200) {
    final data = jsonDecode(response.body);
    print('Connecté avec succès');
    print(data);
    Navigator.pushReplacementNamed(
      context, '/tabbar');
  } else {
    print('Échec de la connexion');
  }
} catch (error) {
  print('Erreur: $error');
}
}
```

- Le formulaire comprenant le champ Adresse email et mot de passe

```
Widget build(BuildContext context) {
  TextField(
    controller: emailController,
    style: TextStyle(color: Colors.white),
    decoration: InputDecoration(
      labelText: 'Adresse e-mail',
      labelStyle: TextStyle(color: Colors.white),
    ), // InputDecoration
  ), // TextField
  SizedBox(height: 20.0),
  TextField(
    controller: passwordController,
    obscureText: true,
    style: TextStyle(color: Colors.white),
    decoration: InputDecoration(
      labelText: 'Mot de passe',
      suffixIcon: Icon(Icons.visibility),
      labelStyle: TextStyle(color: Colors.white),
    ), // InputDecoration
  ), // TextField
}
```

- L'affichage des produit par catégorie : Homme , Femme , Enfant

```
You, 4 minutes ago | 1 author (You) | Codeium: Refactor | Explain
class _ProduitsState extends State<Produits> {
  late List<Map<String, dynamic>> produits = [];
  String selectedCategory = 'Homme';

  @override
  void initState() {
    super.initState();
    fetchProducts();
  }

  static const String apiUrl = 'http://127.0.0.1:3001';

  Future<void> fetchProducts() async {
    final response =
      await http.get(Uri.parse('$apiUrl/api/products/category/$selectedCategory'));
    if (response.statusCode == 200) {
      setState(() {

```

```
static const String apiUrl = 'http://127.0.0.1:3001';
```

Codeium: Refactor | Explain | Generate Function Comment | X

```
Future<void> fetchProducts() async {  
  final response =  
    await http.get(Uri.parse('$apiUrl/api/products/category/$selectedCategory'));  
  if (response.statusCode == 200) {  
    setState(() {  
      produits = List<Map<String, dynamic>>.from(json.decode(response.body));  
    });  
  } else {  
    throw Exception('Failed to load products');  
  }  
}
```

```
FilterButton(  
  text: 'Homme',  
  onPressed: () {  
    setState(() {  
      selectedCategory = 'Homme';  
      fetchProducts();  
    });  
  },  
  selected: selectedCategory == 'Homme',  
), // FilterButton
```

```
FilterButton(  
  text: 'Femme',  
  onPressed: () {  
    setState(() {  
      selectedCategory = 'Femme';  
      fetchProducts();  
    });  
  },  
  selected: selectedCategory == 'Femme',  
), // FilterButton  
FilterButton(  
  text: 'Enfants',  
  onPressed: () {  
    setState(() {  
      selectedCategory = 'Enfants';  
      fetchProducts();  
    });  
  },  
  selected: selectedCategory == 'Enfants',  
), // FilterButton
```

- Pop up pour ajouter un produit

```
class AjouterProduitPopup extends StatefulWidget {
  @override
  AjouterProduitPopupState createState() => _AjouterProduitPopupState();
}

You, 7 minutes ago | 1 author (You) | Codeium: Refactor | Explain
class _AjouterProduitPopupState extends State<AjouterProduitPopup> {
  final TextEditingController nomProduitController = TextEditingController();
  final TextEditingController descriptionController = TextEditingController();
  final TextEditingController categorieController = TextEditingController();
  final TextEditingController couleurController = TextEditingController();
  final TextEditingController tailleController = TextEditingController();
  final TextEditingController promoController = TextEditingController();
  final TextEditingController cateTypeController = TextEditingController();
  final TextEditingController prixController = TextEditingController();
  final TextEditingController quantiteController = TextEditingController();

  List<XFile> selectedImages = [];

  final String defaultImagePath = 'assets/images/logo.png';
}
```

- Fonction me permettant d'ajouter une ou plusieurs image pour un produit

```
Future<void> _addProductWithImages(Map<String, dynamic> productData, List<XFile> images) async {
  final url = Uri.parse('http://127.0.0.1:3001/api/product');
  final request = http.MultipartRequest('POST', url);

  productData.forEach((key, value) {
    request.fields[key] = value.toString();
  });

  if (images.isNotEmpty) {
    for (int i = 0; i < images.length; i++) {
      var stream = http.ByteStream(images[i].openRead());
      var length = await images[i].length();
      var multipartFile = http.MultipartFile('images', stream, length, filename: 'image_${i+1}.png');
      request.files.add(multipartFile);
    }
  }

  try {
    final response = await request.send();
    if (response.statusCode == 200) {
      print('Produit ajouté avec succès !');
    } else {
      print('Erreur lors de l\'ajout du produit');
    }
  } catch (e) {
    print('Erreur: $e');
  }
}
```

- L'utilisation de Picker filter

Codeium: Refactor | Explain | Generate Function Comment | ✕

```
Future<void> _selectImages() async {  
    List<XFile>? images = await ImagePicker().pickMultiImage(  
        imageQuality: 100,  
    );  
};
```

- Fonction pour récupérer toutes les commandes de la bdd

```
late List<Order> orders = [];  
  
@override  
void initState() {  
    super.initState();  
    fetchData();  
}  
  
Future<void> fetchData() async {  
    final response =  
        await http.get(Uri.parse('http://localhost:3001/api/admin-all-orders'));  
    if (response.statusCode == 200) {  
        final List<dynamic> jsonResponse = json.decode(response.body);  
        setState(() {  
            orders = jsonResponse.map((data) => Order.fromJson(data)).toList();  
        });  
    } else {  
        throw Exception('Failed to load orders');  
    }  
}
```

- Fonction me permettant de modifier un produit



```

void _modifierProduit() {
  String nomProduit = _nomController.text;
  String description = _descriptionController.text;
  String categorie = _categorieController.text;
  String couleur = _couleurController.text;
  String taille = _tailleController.text;
  double prix = double.parse(_prixController.text);
  double promo = double.parse(_promoController.text);
  String cateType = _cateTypeController.text;

  // Créer un objet représentant les données à envoyer
  Map<String, dynamic> data = {
    "nomProduit": nomProduit,
    "description": description,
    "categorie": categorie,
    "couleur": couleur,
    "taille": taille,
    "prix": prix,
    "promo": promo,
    "cateType": cateType,
  };
}

```

```

http.put(
  Uri.parse('${widget.apiUrl}/api/product/${widget.produit['id']}'),
  body: json.encode(data),
  headers: {'Content-Type': 'application/json'},
).then((response) {
  if (response.statusCode == 200) {
    print(widget.apiUrl);
    print('Produit mis à jour avec succès !');
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Modification réussie !'),
        duration: Duration(seconds: 2),
      ), // SnackBar
    );
    Future.delayed(Duration(seconds: 2), () {
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(
          builder: (context) => Produits(),
        ), // MaterialPageRoute
      );
    });
  } // Future.delayed
});

```

- Fonction me permettant de supprimer un produit

```

Codeium: Refactor | Explain | Generate Function Comment | X
void _supprimerProduit(Uri apiUrl) {
  var produitId = widget.produit['id'];
  var deleteUrl = Uri.parse('$apiUrl/api/products/$produitId');
  http.delete(deleteUrl).then((response) {
    if (response.statusCode == 200) {
      print('Produit supprimé avec succès !');
      SnackBar snackBar = SnackBar(
        content: Text('Produit supprimé avec succès !'),
        duration: Duration(seconds: 2),
      ); // SnackBar
      ScaffoldMessenger.of(context).showSnackBar(snackBar);


      Navigator.pop(context);
    } else {
      print('Échec de la suppression du produit : ${response.statusCode}');
    }
  });
}

```

## Création de l'API avec Node.js et Express

- Requête pour ajouter un produit

```

app.post('/api/product', upload.array('images', 5), (req, res) => {
  try {
    let imagePaths;
    if (!req.files || req.files.length === 0) {
      imagePaths = defaultImagePath;
    } else {
      imagePaths = req.files.map((file) => file.filename).join(',');
    }
     const { nomProduit, description, categorie, couleur, taille, promo, cateType, prix } =
      You, 1 second ago • Uncommitted changes
    const sql = 'INSERT INTO produits (nomProduit, images, description, categorie, couleur,
    db.query(sql, [nomProduit, imagePaths, description, categorie, couleur, taille, promo, c
    if (err) {
      console.error(err);
      res.status(500).send('Erreur lors de l\'insertion ');
    } else {
      console.log(result);
      res.status(200).send('Article ajouté avec succès !');
    }
  });
} catch (error) {
  console.error('Error handling file upload:', error);
}

```

- Requête pour modifier un produit

```

app.put('/api/product/:id', upload.array('images', 5), (req, res) => {
  const productId = req.params.id;
  const { nomProduit, description, categorie, couleur, taille, promo, cateType, prix } = req.body;

  console.log(req.body);

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

  const updateInfoSql = 'UPDATE produits SET nomProduit=?, description=?, categorie=?, couleur=?, taille=?, promo=?, cateType=? WHERE id=?';
  const updateInfoParams = [nomProduit, description, categorie, couleur, taille, promo, cateType, productId];

  db.query(updateInfoSql, updateInfoParams, (infoErr, infoResult) => {
    if (infoErr) {
      console.error(infoErr);
      res.status(500).send('Erreur lors de la mise à jour du produit.');
```

- Requête pour ajouter une commande

```

app.post('/passerCommande', async (req, res) => {
  try {
    const { userId, products, note, commentaire } = req.body;

    const userOrderHistoryQuery = 'SELECT * FROM commande WHERE utilisateur_id = ?';
    const userOrderHistory = await new Promise((resolve, reject) => {
      db.query(userOrderHistoryQuery, [userId], (err, result) => {
        if (err) {
          console.error(err);
          reject('Erreur lors de la récupération de l\'historique des commandes de l\'utilisateur');
        } else {
          resolve(result);
        }
      });
    });

    const commandeInsertQuery = 'INSERT INTO commande (utilisateur_id, note, commentaire) VALUES (?, ?, ?)';
    await new Promise((resolve, reject) => {
      db.query(commandeInsertQuery, [userId, note, commentaire], async (err, result) => {
        if (err) {
          console.error(err);
          reject('Erreur lors de l\'insertion de la nouvelle commande');
```

```

    await new Promise((resolve, reject) => {
      db.query(commandeInsertQuery, [userId, note, commentaire], async (err, result) => {
        if (err) {
          console.error(err);
          reject('Erreur lors de l\'insertion de la nouvelle commande');
        } else {
          for (const product of products) {
            const { productId, quantity } = product;
            const commandeProduitInsertQuery = 'INSERT INTO commande_produit (comm
            await new Promise((resolve, reject) => {
              db.query(commandeProduitInsertQuery, [result.insertId, productId,
                if (err) {
                  console.error(err);
                  reject('Erreur lors de l\'ajout du produit à la commande')
                } else {
                  resolve();
                }
              });
            });
          }
          resolve();
        }
      });
    });
  }
}

```

- Requête pour récupérer les commande par id et fusionner la table commande produit , utilisateur

```

app.get('/api/admin-all-orders', isAdmin, async (req, res) => {
  try {
    const allOrdersQuery = `
      SELECT
        c.id AS commande_id,
        c.utilisateur_id,
        u.nom AS nom_utilisateur,
        GROUP_CONCAT(cp.produit_id) AS produit_ids,
        GROUP_CONCAT(p.nomProduit) AS noms_produits,
        GROUP_CONCAT(p.prix) AS prix_produits,
        GROUP_CONCAT(p.images) AS images_produits,
        GROUP_CONCAT(cp.quantite) AS quantites_produits,
        c.note,
        c.commentaire,
        c.status
      FROM commande c
      JOIN commande_produit cp ON c.id = cp.commande_id
      JOIN produits p ON cp.produit_id = p.id
      JOIN user u ON c.utilisateur_id = u.id
      GROUP BY c.id
      ORDER BY c.id DESC;
    `;
  }
}

```