DOCUMENTATION TECHNIQUE SITE M2L

La présente documentation technique est destinée à fournir une vue d'ensemble complète et détaillée du site e-commerce de l'entreprise "M2L". Cette plateforme est spécifiquement conçue pour la vente de vêtements de boxe et d'équipements sportifs, répondant ainsi aux besoins des pratiquants et amateurs de boxe. L'objectif principal de cette documentation est de faciliter le développement, la maintenance et l'évolution du site en fournissant toutes les informations nécessaires concernant son architecture, sa mise en place, et son fonctionnement.

SOMMAIRE

INTRODUCTION

La présente documentation technique est destinée à fournir une vue d'ensemble complète et détaillée du site e-commerce de l'entreprise "M2L". Cette plateforme est spécifiquement conçue pour la vente de vêtements de boxe et d'équipements sportifs, répondant ainsi aux besoins des pratiquants et amateurs de boxe. L'objectif principal de cette documentation est de faciliter le développement, la maintenance et l'évolution du site en fournissant toutes les informations nécessaires concernant son architecture, sa mise en place, et son fonctionnement.

Objectif de la Documentation

Cette documentation vise plusieurs objectifs :

- **Faciliter la Compréhension** : Offrir une compréhension claire de l'architecture du site, des technologies utilisées, et des processus de développement.
- **Accompagner le Développement** : Servir de guide pour les développeurs, testeurs, et gestionnaires de projet impliqués dans le développement et la maintenance du site.
- **Assurer la Cohérence** : Aider à maintenir la cohérence dans les pratiques de développement et les standards techniques au fil des versions.

Version du Site

La version actuelle du site est **1.0**. Cette version initiale comprend toutes les fonctionnalités de base nécessaires pour la recherche, la sélection, et la commande de produits sur le site, bien que le paiement se fasse en personne, sur place, en espèces.

Public Cible

Le site est principalement destiné aux habitants de la région Alsace-Lorraine, où la marque "M2L" est basée. En visant ce public local, "M2L" souhaite renforcer sa présence dans la région et offrir une option pratique et accessible pour l'achat d'équipements de boxe de qualité. Cette focalisation régionale permet à "M2L" de créer une connexion plus forte avec sa clientèle et de répondre de manière plus ciblée à ses besoins spécifiques.

ARCHITECTURE DU SITE

Architecture du Site

La structure de "M2L" repose sur une architecture client-serveur classique avec une séparation claire entre le backend (serveur) et le frontend (client). Le backend est construit avec Node.js et communique avec une base de données MySQL, tandis que le frontend est développé avec React.js.

Architecture Générale

Le site "M2L" est divisé en deux parties principales :

Backend (Serveur) : Le backend est responsable de la logique métier, du traitement des données et de la communication avec la base de données. Il expose une API RESTful pour être consommée par le frontend.

- **Controllers** : Les contrôleurs gèrent la logique des requêtes entrantes et renvoient les réponses appropriées.
  - `admin` : Contrôleurs spécifiques à la gestion administrative.
  - `inscriptionConnexionController.js` : Gère l'inscription et la connexion des utilisateurs.
  - `produitsController.js` : Gère les opérations sur les produits.
  - `usersController.js` : Gère les informations des utilisateurs.
- **Database** : Contient les configurations nécessaires pour établir la connexion avec la base de données MySQL.
- **Middleware** : Des fonctions intermédiaires qui sont exécutées avant que les requêtes n'atteignent les contrôleurs ou après que les contrôleurs aient terminé leur traitement.
- **Routes** : Définit les itinéraires de l'API et associe les requêtes aux contrôleurs appropriés

ARCHITECTURE DU SITE

Frontend (Client) : Le frontend est la partie visible par l'utilisateur, construite avec React.js. Il communique avec le backend via l'API pour afficher et gérer les données.

- **Components** :
  - **`AdminProduit.jsx`** et **`AdminUser.jsx`** : Composants pour la gestion administrative des produits et des utilisateurs.
  - **`Boutique.jsx`** : Affiche les produits disponibles à la vente.
  - **`Panier.jsx`** : Gère l'affichage et la modification du panier d'achats.
  - **`Produit.jsx`** : Affiche les détails d'un produit spécifique lorsque cliqué dans la boutique.

Technologies Utilisées

- **Frontend** :
  - React.js : Bibliothèque JavaScript pour construire l'interface utilisateur.
  - CSS : Utilisé pour le style des composants.
- **Backend** :
  - Node.js : Environnement d'exécution JavaScript côté serveur.
  - Express.js : Infrastructure web rapide, non-opinionated et minimaliste pour Node.js.
  - MySQL : Système de gestion de base de données relationnelle.
- **Autres** :
  - PHPMyAdmin : Outil d'administration pour la base de données MySQL.
  - Middleware : Utilisés pour différentes tâches comme l'authentification, la journalisation, la gestion des erreurs, etc.

Structure de la Base de Données

La base de données MySQL contient toutes les données nécessaires pour les utilisateurs, les produits, les commandes, etc. Le fichier **`m21.sql`** contient les scripts de création de la base de données et des tables.

Diagramme de l'Architecture

Un diagramme de l'architecture serait utile pour visualiser la relation entre les différents composants du frontend et du backend. (Note : Un diagramme spécifique n'a pas été fourni, mais il peut être créé en utilisant des outils comme Lucidchart, Draw.io, etc.)

DÉVELOPPEMENT FRONTEND

Le dossier **`front`** contient tous les fichiers nécessaires pour le côté client de l'application. La structure est comme suit :

- **public** : Contient les fichiers statiques comme l'HTML de base et les icônes.
- **src** : Contient le code source de l'application.
  - **assets** : Dossiers pour les ressources statiques telles que les images, les fichiers CSS globaux, etc.
  - **component** : Contient tous les composants React utilisés dans l'application.
    - **`AdminProduit.jsx`** : Composant pour la gestion des produits par l'administrateur.
    - **`AdminUser.jsx`** : Composant pour la gestion des utilisateurs par l'administrateur.
    - **`App.jsx`** : Composant racine de l'application React.
    - **`APropos.jsx`** : Composant pour la page À propos.
    - **`Boutique.jsx`** : Composant pour la page Boutique affichant les produits.
    - **`Footer.jsx`** : Composant pour le pied de page.
    - **`Home.jsx`** : Composant pour la page d'accueil.
    - **`InscriptionConnexion.jsx`** : Composant pour l'inscription et la connexion des utilisateurs.
    - **`Navbar.jsx`** : Composant pour la barre de navigation.
    - **`Panier.jsx`** : Composant pour la gestion du panier d'achat.
    - **`Produit.jsx`** : Composant pour l'affichage d'un produit individuel.
    - **`Profil.jsx`** : Composant pour la page de profil de l'utilisateur.

DÉVELOPPEMENT FRONTEND

  - `style` : Dossier contenant les feuilles de style pour les composants.
  - `index.jsx` : Point d'entrée de l'application React.
- **package.json** & **package-lock.json** : Fichiers définissant les paquets npm et leurs versions verrouillées pour assurer la cohérence entre les environnements de développement.
- **README.md** : Fichier Markdown contenant des informations générales sur le projet, comment l'installer et le démarrer, etc.
- **.gitignore** : Fichier utilisé par git pour ignorer les fichiers et dossiers non nécessaires dans le dépôt.
- **m2l.sql** : Script SQL pour la création de la base de données utilisée par l'application.

DÉVELOPPEMENT FRONTEND

•Composants Clés

•Les composants clés tels que **`AdminProduit.jsx`**, **`AdminUser.jsx`**, et **`Boutique.jsx`** jouent un rôle important dans l'interface administrative et la fonctionnalité de la boutique en ligne. Ils sont généralement composés de sous-composants pour gérer des tâches spécifiques et sont connectés à des états globaux ou locaux pour réagir aux interactions de l'utilisateur.

•Gestion des États

•La gestion des états dans l'application peut être réalisée à l'aide de la bibliothèque React Context ou Redux pour un état global, ou des hooks tels que **`useState`** et **`useEffect`** pour un état local au niveau des composants.

•Styles et Thèmes

•Les styles sont gérés à travers des feuilles de style CSS spécifiques à chaque composant, situées dans le dossier **`style`**. Le thème global ou les styles réutilisables peuvent également être définis ici pour assurer la cohérence visuelle à travers l'application.

DÉVELOPPEMENT BACKEND

•Le dossier **`back`** contient les fichiers et les dossiers nécessaires pour le serveur backend de l'application. La structure est organisée comme suit :

•**controllers** : Contient la logique de contrôle pour les routes de l'API.

- •**Admin :**
  - •**`adminProduitsController.js`** : Gère la logique pour les produits dans la section admin.
  - •**`adminUsersController.js`** : Gère la logique pour les utilisateurs dans la section admin.
- •**`inscriptionConnexionController.js`** : Contrôle les processus d'inscription et de connexion des utilisateurs.
- •**`produitsController.js`** : Contrôle les opérations liées aux produits pour les utilisateurs.
- •**`usersController.js`** : Gère les opérations relatives aux utilisateurs.

•**database** : Contient les fichiers pour la configuration de la base de données.

- •**`database.js`** : Fichier de configuration de la base de données.

DÉVELOPPEMENT BACKEND

- **middleware**
  - `middleware.js` : Contient les middlewares utilisés, comme l'authentification.
- **routes**
  - **admin**
    - `adminProduitsRoute.js` : Routes pour la gestion des produits par les administrateurs.
    - `adminUsersRoute.js` : Routes pour la gestion des utilisateurs par les administrateurs.
  - `inscriptionConnexionRoute.js` : Routes pour l'inscription et la connexion.
  - `produitsRoute.js` : Routes pour les opérations sur les produits.
  - `usersRoute.js` : Routes pour les opérations sur les utilisateurs.
- `package.json` & `package-lock.json` : Fichiers définissant les paquets npm et leurs versions verrouillées.
- `server.js` : Le point d'entrée principal du serveur Node.js.
- **API Endpoints**

Les endpoints de l'API sont définis dans le dossier `routes` et sont organisés par fonctionnalité et rôle utilisateur. Chaque route fait appel à un contrôleur spécifique qui traite la requête et renvoie une réponse.

DÉVELOPPEMENT BACKEND

•**Authentification et Autorisation**

•L'authentification et l'autorisation sont gérées via le middleware dans **`middleware.js`**, qui pourrait implémenter des vérifications de jetons JWT, des sessions ou d'autres mécanismes de sécurité pour s'assurer que seuls les utilisateurs autorisés peuvent accéder à certaines routes et fonctionnalités.

•**Gestion des Sessions**

•La gestion des sessions peut être réalisée via des cookies, des jetons JWT, ou des sessions stockées dans une base de données. Le contrôleur **`inscriptionConnexionController.js`** et le middleware associé jouent un rôle clé dans ce processus pour maintenir les sessions utilisateur sécurisées et persistantes.

BASE DE DONNEE

•**Modèle de données :**

•Deux tables sont définies : **`produit`** et **`utilisateur`**.

•**Schémas et relations :**

•La table **`produit`** comprend les champs suivants :

- •**`id`** : identifiant unique du produit (type char(36)).
- •**`nom`** : nom du produit (varchar(255)).
- •**`prix`** : prix du produit (decimal(10,2)).
- •**`quantite`** : quantité disponible (int).
- •**`description`** : description du produit (text).
- •**`date_creation`** : la date et l'heure de création du produit (timestamp).
- •**`date_mise_a_jour`** : la date et l'heure de la dernière mise à jour du produit (timestamp).

•La table **`utilisateur`** comprend les champs suivants :

- •**`id`** : identifiant unique de l'utilisateur (type char(36)).
- •**`nom`** : nom de famille de l'utilisateur (varchar(255)).
- •**`prenom`** : prénom de l'utilisateur (varchar(255)).
- •**`user_name`** : nom d'utilisateur (varchar(255)).
- •**`date_creation`** : la date et l'heure de création du compte utilisateur (timestamp).
- •**`date_mise_a_jour`** : la date et l'heure de la dernière mise à jour du compte utilisateur (timestamp).
- •**`mot_de_passe`** : mot de passe de l'utilisateur, apparemment haché (varchar(255)).
- •**`admin`** : un booléen indiquant si l'utilisateur a des droits d'administration (tinyint(1)).

BASE DE DONNEE

•**Migration et seeding :**

•Les instructions **`DROP TABLE IF EXISTS`** et **`CREATE TABLE IF NOT EXISTS`** sont des commandes de migration qui assurent la création des tables si elles n'existent pas déjà.

•Les instructions **`INSERT INTO`** sont utilisées pour le seeding, c'est-à-dire pour insérer des données initiales dans les tables.

•**Accès aux données et ORM :**

•L'accès aux données peut être réalisé à travers un ORM (Object-Relational Mapping), qui permettrait de manipuler ces données comme des objets dans le langage de programmation utilisé. Cependant, le dump ne fournit pas de détails spécifiques sur l'ORM utilisé.

SECURITE

- **Stratégies de sécurisation :**
- **Gestion des connexions à la base de données :**
  - Le middleware **`WithDBConnection`** gère la connexion à la base de données. C'est une bonne pratique d'encapsuler la logique de connexion pour éviter les répétitions de code et centraliser la gestion des erreurs de connexion.
- **Authentification avec JWT (JSON Web Tokens) :**
  - Le middleware **`Authentificator`** vérifie si un token JWT est présent et valide. Cela assure que seules les requêtes avec un token valide (c'est-à-dire les utilisateurs authentifiés) peuvent accéder aux routes protégées. L'utilisation de **`process.env.API_KEY`** pour vérifier le token ajoute une couche de sécurité en utilisant une clé secrète stockée dans les variables d'environnement, qui est généralement plus sécurisée que les clés codées en dur.
- **Vérification des rôles :**
  - **`CheckRole`** est un middleware qui vérifie si l'utilisateur authentifié a le rôle d'administrateur. Cela empêche les utilisateurs non autorisés d'accéder à certaines fonctionnalités ou routes qui nécessitent des privilèges élevés.

SECURITE

•**Gestion des vulnérabilités :**

•**Injections SQL :**

•Le code utilise des requêtes paramétrées (par exemple, `'SELECT admin FROM utilisateur WHERE id = ?', [UserId]`) pour interagir avec la base de données, ce qui aide à prévenir les injections SQL.

•**Exposition de données sensibles :**

•Les messages d'erreur devraient éviter de divulguer des détails techniques qui pourraient aider un attaquant. Par exemple, "Erreur de connexion à la base de données" pourrait être remplacé par un message plus générique.

•**Certificats et HTTPS :**

•Le code ne montre pas directement l'utilisation de HTTPS ou la gestion des certificats. Cependant, pour sécuriser les communications entre le client et le serveur, il est essentiel d'utiliser HTTPS, qui chiffre les données en transit. Cela nécessite un certificat SSL/TLS, qui peut être obtenu auprès d'une autorité de certification (CA) et doit être configuré dans le serveur Web.

SECURITE

- **Améliorations possibles :**
- **Gestion des erreurs :**
  - Personnalisez la gestion des erreurs pour éviter de révéler des informations de débogage sensibles aux clients.
- **Validation des entrées :**
  - Assurez-vous que toutes les entrées reçues par l'API sont validées pour éviter les vulnérabilités.
- **Taux de limitation (Rate Limiting) :**
  - Implémenter des limites de taux pour réduire le risque d'attaques par force brute ou de DDoS.
- **HTTPS :**
  - Assurez-vous que l'application est servie uniquement via HTTPS pour protéger les données en transit.
- **Gestion des sessions :**
  - Utilisez des tokens de session sécurisés et assurez-vous qu'ils expirent après une inactivité ou une durée raisonnable.
- **Headers de sécurité HTTP :**
  - Configurer des en-têtes de sécurité HTTP appropriés comme `Content-Security-Policy`, `X-Frame-Options`, `X-XSS-Protection`, etc.
- **Stockage des mots de passe :**
  - Assurez-vous que les mots de passe stockés dans la base de données sont correctement hashés avec un algorithme moderne et sécurisé comme bcrypt, et envisagez d'utiliser le salage et l'étirement des clés.

TEST UNITAIRE

```
//Mock Produits
jest.mock('../controllers/produitsController.js', () => ({
    GetProduitById: jest.fn((id) => {
        if (id === "efba3820-a23d-40f2-8059-f5a799d2cbde") {
            return { success: true, status: 200, body: { "id": "efba3820-a23d-40f2-8059-f5a799d2cbde", "nom": "Produit Mock Test", "prix": "7.77", "quantite": 7, "description": "C'est le produit Mock Test" }
        } else {
            return { success: false, status: 500, error: `Erreur lors de la récupération du produit ${id} ` };
        }
    })
}));

jest.mock('../controllers/admin/adminProduitsController.js', () => ({
    AddProduitFlutter: jest.fn((nom, prix, quantite, description) => {
        if (nom === 'Produit Mock Test' && prix === '7.77' & quantite === 7 && description === "C'est le produit Mock Test") {
            return { success: true, status: 200, message: "Produit ajouté avec succès" };
        } else {
            return { success: false, status: 500, error: "Erreur lors de l'ajout d'un produit" };
        }
    }),

    EditProduit: jest.fn((id, description) => {
        if (id === "efba3820-a23d-40f2-8059-f5a799d2cbde" && description === "C'était le produit Mock Test") {
            return { success: true, status: 200, message: "Produit modifié avec succès" };
        } else {
            return { success: false, status: 500, error: "Erreur lors de la modification du produit" };
        }
    }),

    DeleteProduit: jest.fn((id) => {
        if (id === "efba3820-a23d-40f2-8059-f5a799d2cbde") {
            return { success: true, status: 200, message: "Produit supprimé avec succès" };
        } else {
            return { success: false, status: 500, error: "Erreur lors de la suppression du produit" };
        }
    })
}));
```

Création des différents Mock liés aux produits pour simulé :

- **La récupération d'un produit en fonction de son Id : GetProduitById**
- **La création d'un produit : AddProduitFlutter**
- **La modification d'un produit en fonction de son Id : EditProduit**
- **La suppression d'un produit en fonction de son Id : DeleteProduit**

TEST UNITAIRE

```
describe('Test produitsRoute GET', () => {
    it("Affichage d'un produit en fonction de son id", async () => {
        const res = await GetProduitById("efba3820-a23d-40f2-8059-f5a799d2cbde")
        expect(res.status).toEqual(200);
        expect(res.body).toEqual({
            "id": "efba3820-a23d-40f2-8059-f5a799d2cbde", "nom": "Produit Mock Test", "prix": "7.77", "quantite": 7, "description": "C'est le produit Mock Test"
        });
    })
});

describe('Test adminProduitsRoute POST', () => {
    it("Création d'un produits sur l'application mobile", async () => {
        const res = await AddProduitFlutter("Produit Mock Test", "7.77", 7, "C'est le produit Mock Test")
        expect(res.status).toEqual(200);
        expect(res.message).toEqual("Produit ajouté avec succès")
    })
});

describe('Test adminProduitsRoute PUT', () => {
    it("Modification d'un produits", async () => {
        const res = await EditProduit("efba3820-a23d-40f2-8059-f5a799d2cbde", "C'était le produit Mock Test")
        expect(res.status).toEqual(200);
        expect(res.message).toEqual("Produit modifié avec succès")
    })
});

describe('Test adminProduitsRoute DELATE', () => {
    it("Suppression d'un produits", async () => {
        const res = await DeleteProduit("efba3820-a23d-40f2-8059-f5a799d2cbde")
        expect(res.status).toEqual(200);
        expect(res.message).toEqual("Produit supprimé avec succès")
    })
});
```

Utilisation des Mock créé précédemment dans des tests unitaires avec la vérification du status et du body ou du message

TEST UNITAIRE

```
1   const request = require('supertest');
2   const app = require('../serveur');
3
4   //Mock des middlewares
5   jest.mock('../middleware/middleware', () => ({
6       WithDBConnection: jest.fn((req, res, next) => next()),
7       Authentificator: jest.fn((req, res, next) => next()),
8       CheckRole: jest.fn((req, res, next) => next()),
9   }));
10
11  describe('Test produitsRoute GET', () => {
12      it("Affichage de tous les produits", async () => {
13          const res = await request(app)
14              .get('/api/produits/getAllProduits');
15          expect(res.status).toEqual(200);
16      });
17  });
18
```

Création des Mock pour les différents middlewares

Grâce au mock, nous simulons les accès et affichons tous les produits avec le status de l'erreur égal à 200.

```
PS C:\Users\remyl\Documents\React\m2l\back> npm test

> back@1.0.0 test
> jest

 PASS  test/mock.test.js
 PASS  test/middleware.test.js
A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --detectOpenHandles to find leaks. Active timers can also cause this, en
sure that .unref() was called on them.

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        6.238 s
Ran all test suites.
```

Confirmation que tous les tests unitaires ont bien fonctionné

ANNEXES

- **A. Glossaire des termes**

- **API (Application Programming Interface) :** Ensemble de règles et de définitions qui permet aux logiciels de communiquer entre eux.

- **JWT (JSON Web Token) :** Méthode standard pour sécuriser les échanges d'informations entre deux parties.

- **HTTPS (Hypertext Transfer Protocol Secure) :** Version sécurisée du HTTP, qui est le protocole de communication utilisé pour le transfert des données sur le Web.

- **SQL (Structured Query Language) :** Langage de programmation conçu pour gérer les données contenues dans un système de gestion de base de données relationnelle.

- **Middleware :** Logiciel qui agit comme un pont entre un système d'exploitation ou une base de données et des applications, surtout sur un réseau.

- **Base de données :** Système organisé pour stocker, gérer et récupérer facilement des données.

- **Token :** Fragment de données cryptographiques qui permet de sécuriser l'authentification et l'échange d'informations.

- **Variable d'environnement :** Variable dynamique sur un ordinateur qui peut affecter le comportement des processus en cours d'exécution.

ANNEXES

•**B. Références externes**

•Pour fournir des informations supplémentaires ou pour valider les informations contenues dans le document, les références suivantes peuvent être consultées :

•**RFC 7519 - JSON Web Token (JWT) :** https://tools.ietf.org/html/rfc7519

•**OWASP Top 10 - Les 10 principaux risques de sécurité des applications Web :** https://owasp.org/www-project-top-ten/

•**Let's Encrypt - Autorité de Certification gratuite, automatisée et ouverte :** https://letsencrypt.org/

•**NIST Special Publication 800-63B - Lignes directrices sur les mots de passe :** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf

ANNEXES

- LEROY Rémy
- WYBIER Antoine