DOCUMENTATION TECHNIQUE APP M2L

LEROY Rémy
WYBIER Antoine

SOMMAIRE

- Introduction
- Architecture de l'application
- Développement Frontend
- Développement Backend
- Base de donnée
- Tests
- Annexes

INTRODUCTION

La présente documentation technique est destinée à fournir une vue d'ensemble complète et détaillée de l'application de l'entreprise "M2L". Cette plateforme est spécifiquement conçue pour la gestion administrateur du site (gestion de produits etc...).

L'objectif principal de cette documentation est de faciliter le développement, la maintenance et l'évolution de l'application 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 de l'app, 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. La version actuelle de l'app est 1.0.

Cette version initiale comprend toutes les fonctionnalités de base nécessaires pour la gestion et le contrôle sur le site.

Public Cible:

L'app est principalement destiné aux admins M2L.

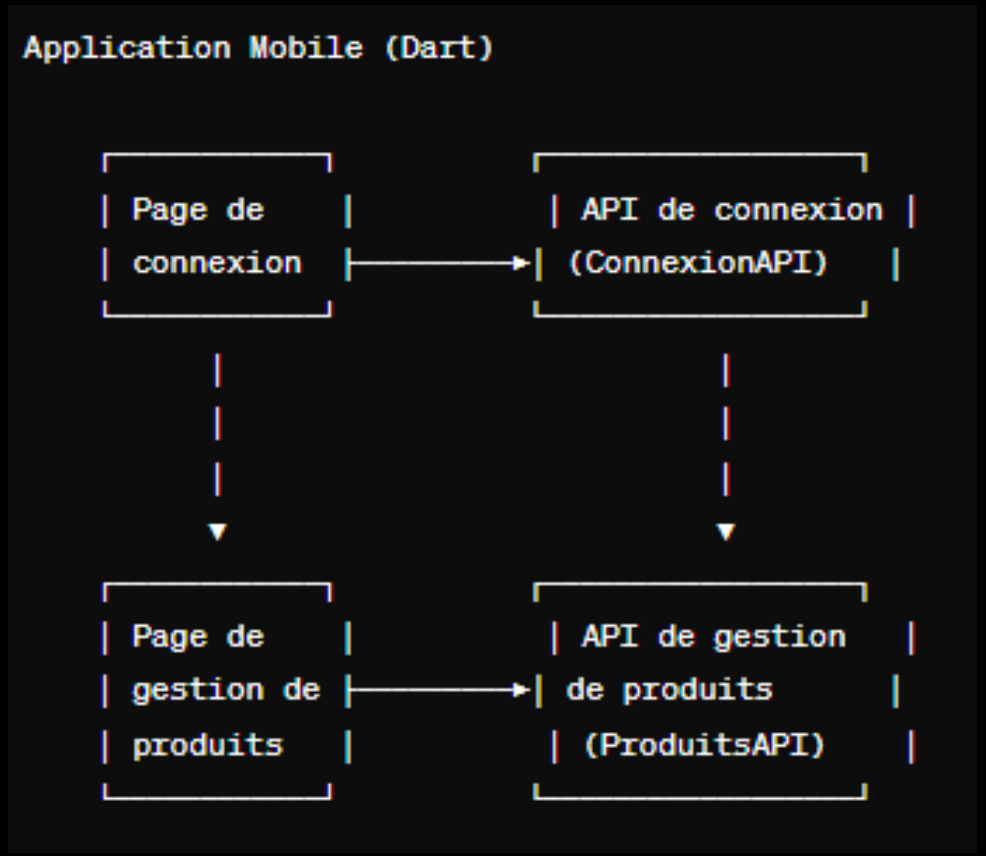
ARCHITECTURE DE L'APPLICATION

Architecture de l'application

La structure de "l'application 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 dart/flutter.

ARCHITECTURE DE L'APPLICATION

- **Page de connexion (Connexion) :**
  - **Interface utilisateur (UI) :** Des champs pour saisir le nom d'utilisateur et le mot de passe, et un bouton pour soumettre les informations.
  - **Logique métier :** Validation des entrées et communication avec une API de backend pour l'authentification.
  - **API de connexion (ConnexionAPI) :** Service backend qui vérifie les identifiants de l'utilisateur.
- **Page de gestion de produits (Produits) :**
  - **Interface utilisateur (UI) :** Liste des produits, boutons ou formulaires pour ajouter, modifier et supprimer des produits.
  - **Logique métier :** Fonctions pour ajouter, mettre à jour et supprimer des produits, ainsi que pour récupérer la liste à jour des produits depuis le backend.
  - **API de gestion de produits (ProduitsAPI) :** Services backend qui gèrent la base de données des produits et les requêtes CRUD (Créer, Lire, Mettre à jour, Supprimer).



DEVELOPPEMENT FRONT END

Page de Connexion

- **Objectif :** Permettre à l'administrateur de s'authentifier pour accéder à la gestion des produits.
- **Composants UI :**
  - **Zone de texte ID Admin :** Champ pour entrer l'identifiant de l'administrateur. Utiliser un `TextField` avec une validation pour s'assurer que le champ n'est pas vide.
  - **Zone de texte Mot de Passe :** Champ pour entrer le mot de passe avec l'attribut `obscureText` à `true` pour cacher le mot de passe. Ajouter également une validation pour ne pas laisser le champ vide.
  - **Bouton Se Connecter :** Un `ElevatedButton` ou un `FlatButton` qui déclenche le processus d'authentification lorsqu'on appuie dessus.
- **Design & Couleurs :**
  - **Couleurs :** Utiliser les couleurs du logo M2L, noir et jaune, pour le thème de la page. Noir pour le texte et les éléments interactifs, jaune pour les boutons et les accents.
  - **Layout :** Les champs de texte doivent être centrés sur la page avec le bouton se connecter en dessous. Utiliser `Column` et `Center` pour organiser les éléments.
- **Fonctionnalités :**
  - **Validation :** Avant de faire une requête à l'API, vérifier que les deux champs sont remplis.
  - **Communication avec l'API :** En cas de clic sur le bouton, envoyer une requête à l'API de connexion et gérer la réponse.

DEVELOPPEMENT FRONT END

Page de Gestion de Produits

- **Objectif :** Afficher la liste des produits existants et permettre l'ajout, la modification et la suppression de produits.
- **Composants UI :**
  - **Liste des Produits :** Utiliser un `ListView.builder` pour afficher la liste des produits. Chaque élément de la liste doit afficher le nom, le prix, la quantité et la description du produit.
  - **Bouton Ajouter (+) :** Un `FloatingActionButton` situé en bas à droite de l'écran pour ajouter un nouveau produit.
  - **Bouton Modifier (crayon) :** Un icône ou bouton pour éditer les détails du produit, placé à côté de chaque produit dans la liste.
  - **Bouton Supprimer (corbeille) :** Un icône ou bouton pour supprimer le produit, situé à côté du bouton modifier.
- **Formulaire d'Ajout/Modification :**
  - **Champs de Saisie :** Des `TextField` pour le nom, le prix, la quantité et la description du produit.
  - **Validation :** S'assurer que tous les champs sont valides avant d'envoyer les données à l'API.
- **Design & Couleurs :**
  - **Couleurs :** Garder la cohérence avec la page de connexion, en utilisant le noir et le jaune du logo M2L.
  - **Disposition :** Une interface claire et intuitive, avec des icônes facilement reconnaissables pour les actions d'édition et de suppression.
- **Fonctionnalités :**
  - **Ajout :** Lorsque l'utilisateur appuie sur le bouton +, ouvrir un formulaire pour entrer les détails du nouveau produit et l'envoyer à l'API après validation.
  - **Modification :** Au clic sur le bouton de modification, afficher un formulaire avec les informations du produit préremplies, permettant à l'utilisateur de les éditer.
  - **Suppression :** Un clic sur l'icône de la corbeille doit déclencher une demande de confirmation, puis supprimer le produit via l'API si confirmée.

DEVELOPPEMENT BACK END

Authentification (API de Connexion)

Endpoint : `/auth/login`

- **Méthode :** POST
- **Body :**
  - **`adminId`** : String (ID de l'administrateur)
  - **`password`** : String (Mot de passe de l'administrateur)
- **Fonctionnement :**
  - Valider les informations d'identification fournies.
  - Générer un token d'authentification si les informations sont valides.
- **Réponses :**
  - **200 OK :** Authentification réussie avec le token en réponse.
  - **401 Unauthorized :** Informations d'identification invalides.
  - **500 Internal Server Error :** Erreur serveur inattendue.

Gestion de Produits

Endpoint : `/products`

- **GET** - Récupérer la liste des produits
  - **Réponses :**
    - **200 OK :** Liste des produits.
    - **500 Internal Server Error :** Erreur serveur inattendue.

  Endpoint : `/products`
- **POST** - Ajouter un nouveau produit
  - **Body :**
    - **`name`** : String
    - **`price`** : Number
    - **`quantity`** : Number
    - **`description`** : String
  - **Réponses :**
    - **201 Created :** Produit ajouté avec succès.
    - **400 Bad Request :** Données manquantes ou invalides.
    - **500 Internal Server Error :** Erreur serveur inattendue.

Endpoint : `/products/:productId`

- **PUT** - Modifier un produit existant
  - **Paramètres :**
    - **`productId`** : String (l'ID du produit dans la base de données)
  - **Body :**
    - **`name`** : String
    - **`price`** : Number
    - **`quantity`** : Number
    - **`description`** : String
  - **Réponses :**
    - **200 OK :** Produit mis à jour avec succès.
    - **400 Bad Request :** Données manquantes ou invalides.
    - **404 Not Found :** Produit non trouvé.
    - **500 Internal Server Error :** Erreur serveur inattendue.

  Endpoint : `/products/:productId`
- **DELETE** - Supprimer un produit
  - **Paramètres :**
    - **`productId`** : String (l'ID du produit dans la base de données)
  - **Réponses :**
    - **200 OK :** Produit supprimé avec succès.
    - **404 Not Found :** Produit non trouvé.
    - **500 Internal Server Error :** Erreur serveur inattendue.

  Sécurité
- Utiliser des middlewares pour gérer l'authentification et la vérification des tokens.
- S'assurer de la sécurisation des données, notamment des mots de passe en utilisant le hachage et le salage.
- Mettre en place une gestion appropriée des erreurs pour donner des réponses claires à l'application front-end.

Base de Données

- **Produits :** Collection avec des documents contenant **`name`**, **`price`**, **`quantity`** et **`description`**.
- **Utilisateurs :** Collection pour les administrateurs avec **`adminId`**, **`passwordHash`**, et potentiellement d'autres informations comme les rôles.

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)).

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

```js
const request = require('supertest');
const app = require('../serveur');

//Mock des middlewares
jest.mock('../middleware/middleware', () => ({
    WithDBConnection: jest.fn((req, res, next) => next()),
    Authentificator: jest.fn((req, res, next) => next()),
    CheckRole: jest.fn((req, res, next) => next()),
}));

describe('Test produitsRoute GET', () => {
    it("Affichage de tous les produits", async () => {
        const res = await request(app)
            .get('/api/produits/getAllProduits');
        expect(res.status).toEqual(200);
    });
});
```

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, ensure 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

LEROY Rémy

WYBIER Antoine