# DOCUMENTATION TECHNIQUE

## Table des matières

# Documentation API

## Package utiliser

- Express.js pour la gestion du serveur HTTP
- Multer pour la gestion des fichiers uploadés
- Bcrypt pour le hashage des mots de passe
- Crypto pour la génération de valeurs aléatoires
- JSON Web Token (JWT) pour l'authentification basée sur des tokens

## Authentification

- POST /user/login : Permet aux utilisateurs de se connecter en fournissant leur pseudo ou email et leur mot de passe. Si les informations sont valides, un token JWT est renvoyé pour authentifier les requêtes ultérieures.
  Dans dart, la fonction de l'api est récupéré, il y est ajouté une vérification vis-à-vis du rôle de l'utilisateur, si il n'es pas admin alors il ne pourra pas se connecter

```dart
class Connexion {
  static String baseUrl = "http://10.0.2.2:3000";

  static Future<List<Map<String, dynamic>>> connection(
      String login, String mdp) async {
    try {
      var url = Uri.parse("$baseUrl/user/connexion");
      var response = await http.post(
        url,
        body: jsonEncode({'login': login, 'mdp': mdp}),
        headers: {'Content-Type': 'application/json'},
      );

      print('Response Status Code: ${response.statusCode}');

      if (response.statusCode == 200) {
        // Convertir la réponse en un seul objet
        var userData = jsonDecode(response.body);
        // Vérifier si l'utilisateur est administrateur
        if (userData['role'] == 'admin') {
          // Encapsuler cet objet dans une liste
          List<Map<String, dynamic>> userDataList = [userData];
          return userDataList;
        } else {
          // Utilisateur n'est pas administrateur, renvoyer une erreur
          throw Exception("Vous n'êtes pas administrateur. Accès refusé.");
        }
      } else {
        // Gérer les autres codes d'état ici
        throw Exception('Server error: ${response.statusCode}');
      }
    } catch (error) {
      throw Exception('Error: $error');
    }
  }
}
```

## Endpoints

1. GET /produit : Récupère tous les produits disponibles.
2. POST /produit/ajouter : Ajoute un nouveau produit avec une image.
3. PUT /produit/modifier/:id : Modifie un produit existant avec une nouvelle image.
4. PUT /produit/modifier/nom/:id : Modifie le nom d'un produit existant.
5. PUT /produit/modifier/stock/:id : Modifie le stock d'un produit existant.
6. PUT /produit/modifier/description/:id : Modifie la description d'un produit existant.
7. PUT /produit/modifier/prix/:id : Modifie le prix d'un produit existant.
8. PUT /produit/supprimer/:id : Supprime un produit en le marquant comme non visible.
9. GET /produit/details/:id : Récupère les détails d'un produit spécifique.

# Documentation Dart et Flutter

*Dart est un langage de programmation développé par Google, utilisé principalement pour créer des applications mobiles, des applications web et des serveurs. Flutter, quant à lui, est un framework open source basé sur Dart, spécialisé dans le développement d'applications multiplateformes. Grâce à son approche de développement par widgets, Flutter permet de créer des interfaces utilisateur réactives et esthétiques, tout en offrant des performances élevées sur différentes plateformes, y compris Android, iOS et le web.*

## Packages Utilisés dans le Projet

| Package | Description | Rôle dans le Projet |
|---|---|---|
| http | Effectue des requêtes HTTP vers le serveur | Communication avec le serveur pour récupérer ou envoyer des données |
| backoffice | Services liés aux produits | Gestion des produits dans l'interface back-office |
| path_provider | Gestion des chemins de fichiers dans le système | Utilisé pour obtenir le chemin de l'image sélectionnée |
| flutter/material.dart | Widgets pour l'interface utilisateur | Création d'éléments visuels dans l'application, tels que des boutons, des textes, des barres d'applications, etc. |
| dart:convert | Fournit des utilitaires pour convertir des données | Utilisé pour encoder et décoder des objets JSON lors des communications avec le serveur |

| Package | Description | Rôle dans le Projet |
|---|---|---|
| flutter/cupertino.dart | Widgets iOS-style pour l'interface utilisateur | Offre une expérience utilisateur similaire à celle d'iOS pour les utilisateurs d'iPhone et d'iPad |

# FONCTIONS

## Fonction : _addProduct() (add.dart)

- Cette fonction est responsable de l'ajout d'un nouveau produit.
- Elle récupère les informations saisies par l'utilisateur, telles que le nom, le stock, le sport, la description et le prix, puis envoie ces données au serveur via une requête HTTP POST.

```
void _addProduct() async {
  String nom = _nomController.text.trim();
  int stock = int.tryParse(_stockController.text.trim()) ?? 0;
  String sport = _sportController.text.trim();
  String description = _descriptionController.text.trim();
  double prix = double.tryParse(_prixController.text.trim()) ?? 0.0;
  try {
    // Call the method to add product with the provided information
    int affectedRows =
        await Add.ajouterProduit(nom, stock, sport, description, prix);

    // If adding the product succeeds
    if (affectedRows > 0) {
      print("Product added successfully!");
      setState(() {
        // Actualiser les données après la mise à jour du produit
      });

      // Rediriger vers la page des produits après la mise à jour
      Navigator.pushReplacementNamed(context, '/productList');
    } else {
      print("Failed to add product!");
      // Handle the case where the product addition fails
    }
  } catch (error) {
    // If an error occurs during product addition
    print("Error adding product: $error");
  }
}
```

## Fonction : _login() (connexion.dart)

- Cette fonction est utilisée pour gérer le processus de connexion de l'utilisateur.
- Elle récupère les informations de connexion saisies par l'utilisateur, envoie ces données au serveur via une requête HTTP POST, puis traite la réponse du serveur pour déterminer si la connexion est réussie ou non.

```dart
void _login() async {
  String login = _pseudoController.text.trim();
  String mdp = _passwordController.text.trim();
  try {
    // Appel de la méthode de connexion avec les informations de l'utilisateur
    List<Map<String, dynamic>>? userData =
        await Connexion.connection(login, mdp);

    // Si la connexion réussit et que des données utilisateur sont retournées
    if (userData != null && userData.isNotEmpty) {
      // Affichez un message de succès ou effectuez d'autres actions en fonction de votre application
      print("Connexion réussie !");

      // Après une connexion réussie, vous pouvez naviguer vers la page des produits
      Navigator.pushReplacementNamed(context, '/productList');
    } else {
      // Si aucune donnée utilisateur n'est retournée, affichez un message d'erreur
      print("Identifiants incorrects !");
    }
  } catch (error) {
    // En cas d'erreur lors de la connexion, affichez un message d'erreur
    print("Erreur lors de la connexion : $error");
  }
}
```

## Fonction : _validatePrice() (ProduitDetailsMod.dart)

- Cette fonction est utilisée pour valider et mettre à jour le prix d'un produit.
- Elle récupère le nouveau prix saisi par l'utilisateur, envoie une requête au serveur pour mettre à jour le prix du produit correspondant, puis actualise l'affichage en conséquence.

```dart
void _validatePrice() {
  double newPrice = double.tryParse(_priceController.text) ?? 0.0;
  Update.updateProductPrice(
    widget.productId,
    newPrice,
  ).then((statusCode) {
    if (statusCode == 200) {
      // Mettre à jour l'affichage
      setState(() {
        _productDetails = Produit.getProductById(widget.productId);
      });
      _showSnackBar('La modification est effective');
    } else {
      // Afficher un message d'erreur
      _showSnackBar('Échec lors de la modification');
    }
  }).catchError((error) {
    // Afficher un message d'erreur
    _showSnackBar('Échec lors de la modification');
  });
```

## Fonction : _validateStock() (ProduitDetailsMod.dart)

- Cette fonction est utilisée pour valider et mettre à jour le stock d'un produit.
- Elle récupère la nouvelle quantité de stock saisie par l'utilisateur, envoie une requête au serveur pour mettre à jour le stock du produit correspondant, puis actualise l'affichage en conséquence.

```dart
  void _validateStock() {
    int newStock = int.tryParse(_stockController.text) ?? 0;
    Update.updateProductStock(
      widget.productId,
      newStock,
    ).then((statusCode) {
      if (statusCode == 200) {
        // Mettre à jour l'affichage
        setState(() {
          _productDetails = Produit.getProductById(widget.productId);
        });
        _showSnackBar('La modification est effective');
      } else {
        // Afficher un message d'erreur
        _showSnackBar('Échec lors de la modification');
      }
    }).catchError((error) {
      // Afficher un message d'erreur
      _showSnackBar('Échec lors de la modification');
    });
  }
}
```

## Fonction : _showSnackBar() (ProduitDetailsMod.dart)

- Cette fonction est utilisée pour afficher un SnackBar avec un message donné.

- Elle est utilisée pour afficher des messages de confirmation ou d'erreur à l'utilisateur dans la page des détails du produit.

```
void _showSnackBar(String message) {
  ScaffoldMessenger.of(context).showSnackBar(SnackBar(
    content: Text(message),
    duration: Duration(seconds: 2),
  )); // SnackBar
}
```

## Fonction : main() (main.dart)

- Cette fonction est le point d'entrée de l'application Flutter.
- Elle configure et lance l'application en définissant la page d'accueil, les thèmes et les routes de navigation.

```
void main() {
  runApp(const MaterialApp(
    home: Home(),
  )); // MaterialApp
}

class Home extends StatelessWidget {
  const Home({Key? key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Backoffice',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      initialRoute: '/', // Set the initial route to '/'
      routes: {
        '/': (context) => AuthService(), // Route for the login page
        '/productList': (context) =>
            AffichageProduit(), // Route for the product page
        '/add': (context) => AddPage(),
      },
    ); // MaterialApp
  }
}
```