Manuel technique de l'application mobile

Sommaire

1) CRUD des produits

Ajout d'un produit :
On donne le nom, le prix, la description, le stock, l'action et l'image sur le formulaire.

```dart
class AddFormHomePage extends StatelessWidget {
  final String title;
  final TextEditingController nomController = TextEditingController();
  final TextEditingController prixController = TextEditingController();
  final TextEditingController descriptionController = TextEditingController();
  final TextEditingController stockController = TextEditingController();
  final TextEditingController actionController = TextEditingController();
  final TextEditingController imageController = TextEditingController();
```

Cette méthode envoie une requête POST pour ajouter un produit dans le tableau. Si la requête est réussie l'utilisateur est redirigé vers la page d'accueil sinon une exception est levé et affichée dans la console

```dart
// Fonction pour ajouter un produit
Future<void> addProduct(String nom, double prix, String description,
    int stock, String image, BuildContext context) async {
  // Ajout de BuildContext context
  final url = Uri.parse('http://localhost:8000/api/produit/addProduct');

  try {
    final response = await http.post(
      url,
      body: jsonEncode({
        'nom': nom,
        'prix': prix,
        'description': description,
        'stock': stock,
        'image': image
      }),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
    );

    if (response.statusCode == 200) {
      // Si la requête est réussie, vous pouvez rediriger l'utilisateur vers la page d'accueil ou faire d'autres actions nécessaires.
      Navigator.pushNamed(context, '/homePage');
    } else {
      throw Exception('Failed to add product');
    }
  } catch (e) {
    print('Error adding product: $e');
  }
}
```

Récupération des produits :
Cette méthode envoie une requête GET pour récupérer la liste des produits depuis l'API. Elle retourne une liste de « Map<String, dynamic> » représentant les produits.
En cas d'erreur lors de la récupération des produits, une exception est lancée.

```dart
// GET products
Future<List<Map<String, dynamic>>> recupererProduits() async {
  final url =
      Uri.parse('http://localhost:8000/api/produit/allProducts'); //10.74.1.203

  try {
    final response = await http.get(url);

    if (response.statusCode == 200) {
      final List<dynamic> data = json.decode(response.body);
      return data.cast<Map<String, dynamic>>();
    } else {
      throw Exception('Failed to load products');
    }
  } catch (e) {
    throw Exception('Failed to load products: $e');
  }
}
```

Il contient une liste « products » qui stocke les produits récupérés depuis l'API.
Elle appelle « _loadProducts() » pour charger les produits au démarrage de la page.
« _loadProducts() » appelle « recupererProduits() » pour récupérer les produits, puis met à jour l'état (products) de la page avec les produits récupérés.

```dart
// GET products
  List<Map<String, dynamic>> products = [];
  @override
  void initState() {
    super.initState();
    _loadProducts(); // Appel à la méthode pour charger les produits
  }

  // Méthode pour charger les produits
  void _loadProducts() {
    recupererProduits().then((List<Map<String, dynamic>> produits) {
      setState(() {
        products.clear();
        products.addAll(produits);
      });
    }).catchError((error) {
      print('Error loading products: $error');
    });
  }
```

Suppression d’un produit :
« deleteProduct(String id) » est une méthode asynchrone qui envoie une requête DELETE pour supprimer un produit de l'API. Elle prend l'id du produit à supprimer en paramètre. Si la suppression réussit, elle met à jour l'état (products) en supprimant le produit de la liste. Ensuite, elle appelle « _loadProducts() » pour recharger la liste des produits après la suppression.

```dart
// DELETE product
  Future<void> deleteProduct(String id) async {
    final url =
        Uri.parse('http://localhost:8000/api/produit/deleteProduct/$id');

    try {
      final response = await http.delete(url);

      if (response.statusCode == 200) {
        // Produit supprimé avec succès, actualisez l'affichage des produits
        setState(() {
          products.removeWhere((product) => product['id'] == id);
        });

        // Actualisez les produits après avoir supprimé le produit
        _loadProducts();
      } else {
        throw Exception('Failed to delete product');
      }
    } catch (e) {
      print('Error deleting product: $e');
    }
  }
```

2) Connexion / Déconnexion

Connexion :
« _token » : Variable statique privée qui contient le jeton d'authentification. Elle est initialisée à null.
« token » (Getter) : Méthode statique qui permet de récupérer le jeton d'authentification. En l'appelant avec « Auth_Token.token », on peut obtenir la valeur actuelle du jeton.
« setToken » : Méthode statique qui permet de définir le jeton d'authentification. Elle prend en paramètre un nouveau jeton (« token ») et le stocke dans la variable « _token ».

```dart
// Classe qui permet de stocker le token de l'utilisateur connecté
You, 5 seconds ago | 1 author (You)
class Auth_Token {
  static String? _token; // Le token d'authentification, initialisé à null

  // Méthode pour récupérer le token
  static String? get token => _token;

  // Méthode pour définir le token
  static void setToken(String? token) {
    _token = token;
  }

  // Méthode pour supprimer le token (déconnexion)
  static void removeToken() {
    _token = null;
  }
}
```

Il récupère d'abord le nom d'utilisateur (username) et le mot de passe (password) à partir des contrôleurs _usernameController et _passwordController.
Il utilise le package http pour effectuer une requête POST asynchrone vers l'URI spécifiée (http://localhost:8000/api/utilisateur/login).
La requête inclut des en-têtes spécifiant Content-Type comme application/json; charset=UTF-8, et un corps (body) encodé en JSON contenant le nom d'utilisateur (mail) et le mot de passe (mdp).

```dart
final String username = _usernameController.text;
final String password = _passwordController.text;

final response = await http.post(
  Uri.parse('http://localhost:8000/api/utilisateur/login'),
  headers: <String, String>{
    'Content-Type': 'application/json; charset=UTF-8',
  },
  body: jsonEncode(<String, String>{
    'mail': username,
    'mdp': password,
  }),
);
```

Si statusCode est 200 (HTTP OK), cela signifie une authentification réussie.
Il décode ensuite le corps de la réponse (probablement contenant les données de l'utilisateur) du format JSON. Il extrait les valeurs token et isAdmin de la réponse décodée. Si isAdmin vaut 1, cela indique que l'utilisateur est un administrateur. Un message de réussite est affiché à l'aide d'un SnackBar. Le jeton (token) reçu est stocké en utilisant Auth_Token.setToken(token). L'utilisateur est ensuite redirigé vers la page /home en utilisant Navigator.pushReplacementNamed(context, '/home'). Si isAdmin n'est pas 1, cela signifie que l'utilisateur n'est pas un administrateur, et un message d'erreur est affiché à l'aide d'un SnackBar. Si statusCode n'est pas 200, cela indique une authentification échouée.

```dart
if (response.statusCode == 200) {
  // Connexion réussie, récupérer les données de réponse
  final responseData = jsonDecode(response.body);
  final token = responseData['token'];
  final isAdmin = responseData['isAdmin'];

  if (isAdmin == 1) {
    // L'utilisateur est un administrateur, procédez à la connexion
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Connexion réussie!'),
      ), // SnackBar
    );

    // Stockage du token dans la variable globale
    Auth_Token.setToken(token);
    // Redirection vers la page HomePage après une connexion réussie
    Navigator.pushReplacementNamed(context, '/home');
  } else {
    // L'utilisateur n'est pas un administrateur, afficher un message d'erreur
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text(
            'Seuls les administrateurs sont autorisés à se connecter.'), // Text
      ), // SnackBar
    );
  }
} else {
  // Échec de la connexion
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(
      content: Text('Échec de la connexion'),
    ), // SnackBar
  );
}
```

Déconnexion :
« removeToken » : Méthode statique qui permet de supprimer le jeton d'authentification (en le définissant comme null), ce qui simule la déconnexion de l'utilisateur.

```dart
// Méthode pour supprimer le token (déconnexion)
static void removeToken() {
  _token = null;
}
```

« onPressed » : Propriété du « ElevatedButton » qui spécifie l'action à effectuer lorsque le bouton est pressé. Dans ce cas, lorsque le bouton est pressé, la méthode « removeToken() » de « Auth_Token » est appelée pour supprimer le jeton d'authentification, simulant ainsi une déconnexion. Ensuite, « Navigator.pushReplacementNamed(context, '/') » est utilisé pour naviguer à l'écran de connexion en remplaçant l'écran actuel.
« child » : Propriété du « ElevatedButton » qui spécifie le widget enfant à afficher dans le bouton. Ici, un texte 'Déconnexion' est affiché comme contenu du bouton.

```dart
ElevatedButton(
  onPressed: () {
    // Suppression du token lors de la déconnexion
    Auth_Token.removeToken();
    // Retour à l'écran de connexion
    Navigator.pushReplacementNamed(context, '/');
  },
  child: const Text('Déconnexion'),
), // ElevatedButton
```

3) Test Unitaires

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

describe('Test Route GET', () => {
    it('Recuperation Produits', async () => {
        const res = await request(app)
    .get('api/produits/allProducts')
    expect(res.status).toEqual(200);
    });
})

describe('Test Route POST', () => {
    it('Création d\'un produit', async () => {
        const res = await request(app)
        .post('/api/produits/addProduct')
        .send({
            nom: "Produit",
            prix: 60,
            description: "description",
            image:"image.png",
            stock: 1
        })
        expect(res.status).toEqual(200);
        expect(res.body).toEqual(1);
    });
});
```

```
describe('Test Route PUT', () => {
    it('Modification d\'un produit', async () => {
        const res = await request(app)
        .post('/api/produits/editProduct')
        .send({
            id: 1,
            nom: "Produit modifié "
        })
        expect(res.status).toEqual(200);
    });
});

describe('Test Route DELETE', () => {
    it('Suppression d\'un produit', async () => {
        const res = await request(app)
        .post('/api/produits/deleteProduct')
        .send({
            id: 1
        })
        expect(res.status).toEqual(200);
    });
});
```

Les tests unitaires servent à vérifier le bon fonctionnement de nos routes en asynchrone, je présente ici comment ceux-ci fonctionne pour la gestion des produits.