WuberEat

WuberEat est une application de démonstration et d'apprentissage développée avec le framework Flutter. Inspirée de l'application UberEat, elle met en avant des concepts fondamentaux pour le développement d'applications mobiles. Elle dispose d'une interface utilisateur intuitive et implémente des fonctionnalités clés telles que la navigation entre pages, l'authentification des utilisateurs, et la gestion des données locales via SQLite.

Objectifs de l'application

- Illustrer la conception d'interfaces utilisateur modernes et réactives.
- Implémenter un système de gestion d'utilisateurs avec une base de données locale.
- Servir de base pour l'apprentissage des bases de Flutter et SQLite.

Fonctionnalités principales

1. **Page d'accueil** : Permet de naviguer vers les différents produits disponibles.
2. **Page de détails du produit** : Offre des informations supplémentaires sur chaque produit et permet de l'ajouter au panier.

3. **Système d'authentification** : Gestion des connexions et inscriptions via SQLite.
4. **Base de données locale** : Stockage des informations utilisateur.

Documentation : Système d'Authentification et Gestion des Utilisateurs

Cette section documente le système d'authentification et la gestion des utilisateurs de l'application **WuberEat**. Ce système repose sur trois composantes principales : les pages de connexion et d'inscription, la gestion de la base de données SQLite, et le modèle de données JSON des utilisateurs.

---

1. Système d'Authentification

Page de Connexion : `login.dart`

La page de connexion permet aux utilisateurs d'accéder à leur compte existant en renseignant leur nom d'utilisateur et leur mot de passe.

- **Fonctionnalités principales** :
  - Validation des champs de formulaire pour vérifier que les informations sont complètes.
  - Envoi des informations à la base de données pour vérification.
  - Affichage d'un message d'erreur si les informations sont incorrectes.
  - Possibilité de naviguer vers la page d'inscription.
- **Extrait de code** :

```
Future<void> login() async {
  var response = await db.login(Users(userName: username.t
ext, userPassword: password.text));
  if (response == true) {
    print("login successful");
  } else {
    setState(() {
```

```
      _isLoginTrue = true;
    });
  }
}
```

- **Interface utilisateur** :
  - Champs pour le nom d'utilisateur et le mot de passe.
  - Bouton permettant de basculer entre l'affichage et le masquage du mot de passe.
  - Lien vers la page d'inscription.

Page d'Inscription : `signup.dart`

La page d'inscription permet de créer un compte utilisateur.

- **Fonctionnalités principales** :
  - Validation des champs, incluant la vérification de la correspondance des mots de passe.
  - Enregistrement des informations dans SQLite.
  - Redirection vers la page de connexion après l'inscription.
- **Extrait de code** :

```
Future<void> signup() async {
  final db = DatabaseHelper();
  db.signup(Users(userName: username.text, userPassword: p
assword.text)).whenComplete(() {
    Navigator.push(context, PageRouteBuilder(
      pageBuilder: (context, animation1, animation2) => Lo
ginScreen(),
    ));
  });
}
```

- **Interface utilisateur** :

- Champs pour le nom d'utilisateur, le mot de passe, et la confirmation du mot de passe.
- Indicateurs visuels pour les mots de passe non concordants.
- Bouton pour naviguer vers la page de connexion.

---

2. Gestion des Utilisateurs avec SQLite

Initialisation de la Base de Données

La classe `DatabaseHelper` gère l'interaction avec SQLite. La base de données est initialisée à l'aide de la méthode `initDB`.

- **Structure de la table `users`** :
  - `userId` : Identifiant unique (clé primaire).
  - `userName` : Nom d'utilisateur unique.
  - `userPassword` : Mot de passe.
  - `createdAt` : Date et heure de création.
- **Extrait de code** :

```
Future<Database> initDB() async {
  final databasePath = await getDatabasesPath();
  final path = join(databasePath, databaseName);

  return openDatabase(path, version: 1, onCreate: (db, ver
sion) async {
    await db.execute('CREATE TABLE users (
      userId INTEGER PRIMARY KEY AUTOINCREMENT,
      userName TEXT UNIQUE,
      userPassword TEXT,
      createdAt TEXT DEFAULT CURRENT_TIMESTAMP
    )');
  });
}
```

Connexion

La méthode `login` vérifie les informations d'authentification d'un utilisateur en interrogeant SQLite.

- **Extrait de code** :

```
Future<bool> login(Users user) async {
  final Database db = await initDB();
  var result = await db.rawQuery("SELECT * FROM users WHER
E userName = ? AND userPassword = ?", [user.userName, use
r.userPassword]);
  return result.isNotEmpty;
}
```

Inscription

La méthode `signup` ajoute un nouvel utilisateur dans la base de données.

- **Extrait de code** :

```
Future<int> signup(Users user) async {
  final Database db = await initDB();
  return db.insert('users', user.toMap());
}
```

---

3. Modèle de Données : `users.dart`

La classe `Users` définit le modèle pour les informations utilisateur. Elle est conçue pour faciliter la conversion entre les objets Dart et les structures JSON ou SQLite.

Structure de la Classe

- **Attributs** :
  - `userId` : Identifiant unique.
  - `userName` : Nom d'utilisateur.

- `userPassword` : Mot de passe.

Méthodes Clés

1. `fromMap` : Crée une instance de `Users` à partir d'un `Map`.

```
factory Users.fromMap(Map<String, dynamic> json) => Users(
    userId: json["userId"],
    userName: json["userName"],
    userPassword: json["userPassword"],
);
```

2. `toMap` : Convertit une instance de `Users` en `Map`.

```
Map<String, dynamic> toMap() => {
    "userId": userId,
    "userName": userName,
    "userPassword": userPassword,
};
```

Cette section documente l'interface utilisateur de l'application **WuberEat**, avec une analyse des pages principales et des modèles de données qui alimentent ces pages. L'interface est conçue pour offrir une expérience utilisateur fluide, moderne et intuitive.

---

4. Page Principale : `home.dart`

La page principale constitue le point d'entrée de l'application. Elle présente plusieurs sections : recherche, navigation par catégories, et recommandations personnalisées.

Barre de navigation (AppBar)

- **Description** : La barre de navigation affiche le titre de l'application et un menu d'options.
- **Fonctionnalités** :

- Titre centré.
- Menu d’options pour accéder à des fonctionnalités supplémentaires (ex. : compte utilisateur, commandes, aide).

Barre de recherche

- **Description** : Champ de texte permettant de rechercher des produits.
- **Fonctionnalités** :
  - Icône de recherche et filtre visuel.
  - Conception moderne avec ombrages et bordures arrondies.
- **Extrait de code :**

```
Container _search() {
  return Container(
    decoration: BoxDecoration(
      boxShadow: [
        BoxShadow(
          color: Colors.grey.withOpacity(0.5),
          spreadRadius: 1,
          blurRadius: 8,
        ),
      ],
    ),
    child: TextField(
      decoration: InputDecoration(
        hintText: 'Are you Hungry?',
        prefixIcon: SvgPicture.asset('assets/icons/Search.
svg'),
        suffixIcon: SvgPicture.asset('assets/icons/Filter.
svg'),
        border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(15),
        ),
      ),
```

```
    ),
  );
}
```

Section des catégories

- **Description** : Affiche une liste horizontale de catégories, chaque élément étant représenté par une icône et un nom.
- **Extrait de code :**

```
Column _categoriesSection() {
  return Column(
    children: [
      const Text(
        'Nearby Restaurants',
        style: TextStyle(fontSize: 20, fontWeight: FontWei
ght.bold),
      ),
      ListView.separated(
        scrollDirection: Axis.horizontal,
        itemCount: categories.length,
        itemBuilder: (context, index) {
          return Container(
            child: SvgPicture.asset(categories[index].icon
path),
          );
        },
      ),
    ],
  );
}
```

Section des recommandations

- **Description** : Carrousel horizontal affichant des produits tendance.

- **Extrait de code :**

```
Column _recommended() {
  return Column(
    children: [
      const Text(
        'Recommended For You',
        style: TextStyle(fontSize: 20, fontWeight: FontWei
ght.bold),
      ),
      ListView.separated(
        scrollDirection: Axis.horizontal,
        itemCount: tendance.length,
        itemBuilder: (context, index) {
          return Container(
            child: Text(tendance[index].name),
          );
        },
      ),
    ],
  );
}
```

Navigation vers la page de détails

- **Description** : Chaque produit affiché dans les recommandations redirige vers une page de détails lors de l'interaction.
- **Extrait de code :**

```
GestureDetector _orderNowButton(BuildContext context, int
index) {
  return GestureDetector(
    onTap: () {
      var selectedTrend = tendance[index];
      Navigator.push(
```

```
        context,
        PageRouteBuilder(
          pageBuilder: (context, animation, secondaryAnima
tion) =>
              ProductPage(selectedTrend: selectedTrend),
        ),
      );
    },
  );
}
```

---

5. Page de Détails : `product.dart`

La page des détails fournit des informations complètes sur un produit sélectionné.

Composants principaux

1. **Image du produit** :
   - Icône SVG avec une bordure arrondie.
   - **Extrait de code :**

     ```
     Container(
       decoration: BoxDecoration(
         border: Border.all(color: Color.fromARGB(255, 194,
     194, 194), width: 2),
         borderRadius: BorderRadius.circular(50),
       ),
       child: SvgPicture.asset(selectedTrend.iconpath, heigh
     t: 300, width: 300),
     );
     ```

2. **Informations textuelles** :
   - Nom, description, temps de livraison, et prix du produit.
   - **Extrait de code :**

```
Text(selectedTrend.name, style: const TextStyle(fontSiz
e: 30, fontWeight: FontWeight.bold));
Text(selectedTrend.texte, style: const TextStyle(fontSi
ze: 20, fontStyle: FontStyle.italic));
```

3. **Bouton « Like »** :
   - Interaction dynamique pour signaler un intérêt pour le produit.
   - **Extrait de code :**

```
LikeButton(
  size: 90,
  likeBuilder: (bool isLiked) {
    return Container(
      child: Text('I WANT IT', style: TextStyle(color:
Colors.black, fontSize: 17)),
    );
  },
);
```

---

6. Modèles de Données : Catégories et Tendances

Modèle des Catégories (``)

- **Attributs principaux** :
  - `name` : Nom de la catégorie (ex. Burger, Pizza).
  - `iconpath` : Chemin de l'icône SVG.
  - `boxColor` : Couleur associée.
- **Extrait de code :**

```
static List<CategoryModel> getCategories() {
  return [
    CategoryModel(
```

```
      name: 'Burger',
      iconpath: 'assets/icons/burger.svg',
      boxColor: Color.fromARGB(255, 200, 247, 185),
    ),
  ];
}
```

Modèle des Tendances (``)

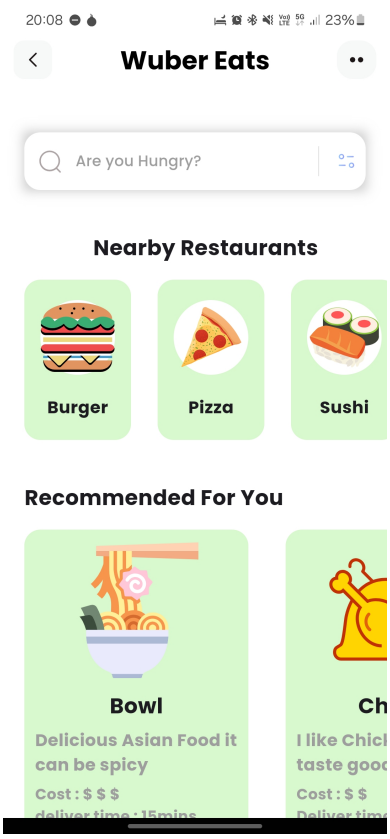
- **Attributs principaux** :
  - `name` : Nom de la tendance.
  - `description` : Brève description.
  - `price` : Coût.
  - `deliverTime` : Temps de livraison estimé.
- **Extrait de code :**

```
static List<tendanceModel> getTendance() {
  return [
    tendanceModel(
      name: 'Bowl',
      iconpath: 'assets/icons/bowl.svg',
      price: 'Cost : \$ \$ \$',
      deliverTime: 'deliver time : 15mins',
      boxColor: Color.fromARGB(255, 200, 247, 185),
      description: 'Delicious Asian Food',
      texte: 'A bowl of rice, cooked with various ingredie
nts.',
    ),
  ];
}
```

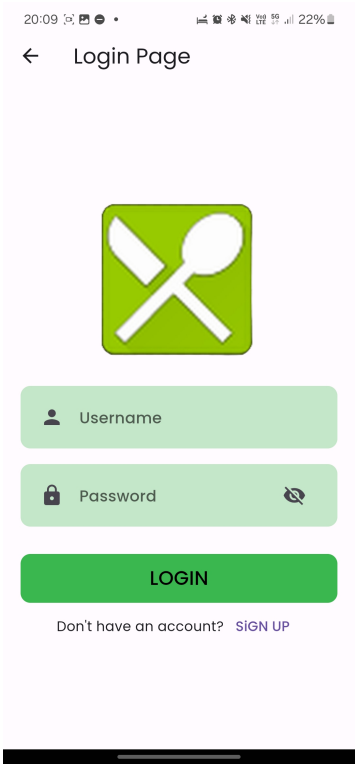
Conclusion

L’interface utilisateur de l’application WuberEat est conçue pour être à la fois intuitive et esthétique. Les modèles de données bien structurés permettent une gestion fluide des informations. Ensemble, ces éléments offrent une expérience utilisateur cohérente et engageante.

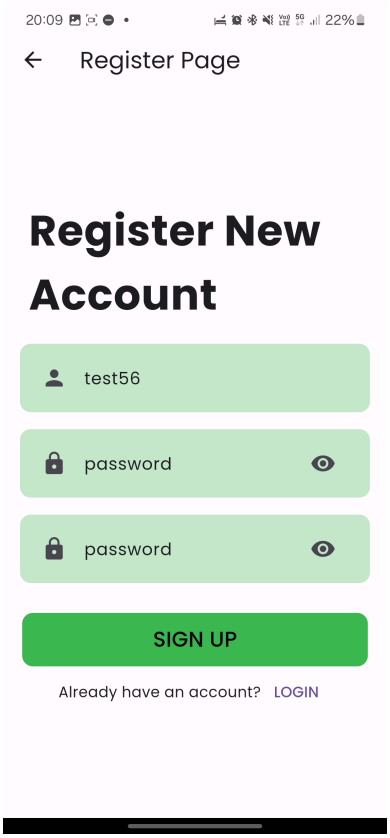
Capture d’écran de l’application :



Menu Principale



Page de connexion



Page d’inscription