📘 Documentation Technique – Projet AP4 / GSB MedOrder

---

1. Vue d'ensemble

Le projet **AP4 / MedOrder** est une application **multiplateforme** (mobile, desktop, web) développée avec **Flutter** pour le frontend et **Node.js (Express.js)** pour le backend. Elle permet à des utilisateurs professionnels (pharmaciens ou praticiens) de **se connecter**, **consulter leurs commandes** et **gérer leur profil**.

---

2. Structure du projet

```
ap4/
├── Backend/            # Serveur Node.js (API REST)
│   ├── app.js          # Point d'entrée du serveur
│   ├── config/         # Connexion à la base de données
│   ├── controllers/    # Logique métier (users, commandes, médicaments)
│   ├── routes/         # Routes de l'API
│   └── package.json    # Dépendances Node.js
├── lib/                # Code source Flutter
│   ├── main.dart       # Point d'entrée Flutter
│   ├── login_page.dart
│   ├── home_page.dart
│   ├── profile_page.dart
│   ├── order_details_page.dart
│   ├── main_navigation_page.dart
│   ├── services/       # Appels API (auth_service.dart)
│   ├── models/         # Modèles de données
│   └── widgets/        # Widgets réutilisables
├── platform folders/   # android/, ios/, web/, windows/, macos/, linux/
├── test/               # Tests unitaires Flutter
├── pubspec.yaml        # Dépendances Flutter
└── README.md
```

---

3. Frontend – Flutter

- **Langage :** Dart
- **Framework :** Flutter
- **Entrée principale :** `lib/main.dart`
- **Navigation :** Routes nommées (`/login`, `/home`, `/profile`)

Écrans :

- `LoginPage` : Authentification (formulaire email + mot de passe)
- `HomePage` : Liste des commandes via `AuthService.getUserOrders`
- `ProfilePage` : Données utilisateur via `AuthService.getUserDetails`
- `OrderDetailsPage` : Détails d'une commande
- `MainNavigationPage` : Navigation avec `BottomNavigationBar`

Services :

- `AuthService` :
  - Login utilisateur
  - Récupération des commandes
  - Récupération des infos utilisateur
  - Utilise le package `http`

Gestion d'état :

- `StatefulWidget` avec `setState`
- `FutureBuilder` pour gérer les appels API asynchrones

Dépendances clés :

- `flutter`
- `http`

- `intl`

---

4. Backend – API Node.js

- **Langage :** JavaScript (Node.js)
- **Framework :** Express.js
- **Entrée principale :** `Backend/app.js`
- **Middlewares :** `cors`, `express.json`

Base de données :

- **Type :** MySQL
- **Connexion :** `Backend/config/db.js`
- **Tables utilisées :**
  `users`, `commandes`, `medicaments`, `lignesCommande`, `commande_details`, `suivi_commandes`

Routes API :

Utilisateurs (authentification)

- `POST /api/users/login` : Connexion (JWT + bcrypt)
- `POST /api/users/register` : Inscription
- `GET /api/users/:userId` : Détails utilisateur

Commandes

- `GET /api/commandes/user/:userId` : Commandes utilisateur
- `POST /api/commandes` : Nouvelle commande

- `GET /api/commande_details/commande/:id` : Détails commande

Suivi commandes

- `GET /api/suivi/commandes/:commandeId/suivi` : Historique
- `POST /api/suivi/suivi_commandes` : Nouveau statut

Médicaments

- `GET /api/medicaments` : Tous les médicaments
- `GET /api/medicaments/:id` : Médicament par ID

Dépendances clés :

- `express`, `cors`, `mysql2`, `bcrypt`, `jsonwebtoken`, `dotenv`

---

5. Lancement du projet

Backend

```
cd Backend
npm install
# Créez un fichier .env avec les infos MySQL
node app.js   # ou npm start
```

Frontend

```
cd ap4
flutter pub get
flutter run   # sur émulateur ou appareil réel
```