# Documentation Module Messagerie – Hivemind Academy Mobile

## 1. AuthService

### 1.1 Gestion des tokens

- **Ancien fonctionnement :** Les tokens (access_token, refresh_token) étaient stockés directement dans `FlutterSecureStorage` sans couche d'abstraction.
- **Nouveau fonctionnement :**
  - Accès au stockage des tokens encapsulé dans une classe dédiée `TokenStorage`.
  - Implémentation d'une vérification automatique d'expiration des tokens avant chaque appel API.
  - Méthode unique `getValidAccessToken()` garantissant le retour systématique d'un token valide, avec rafraîchissement automatique si nécessaire.

### 1.2 Gestion des utilisateurs

- **Ancien fonctionnement :** Les informations utilisateur étaient stockées via `UserService1` et la désérialisation `User.fromJson`.
- **Nouveau fonctionnement :**
  - Introduction d'un `UserRepository` pour une meilleure séparation des responsabilités.
  - `AuthService` se concentre uniquement sur l'authentification, déléguant la persistance des données utilisateur.
  - Ajout d'un singleton `CurrentUser` qui conserve les informations utilisateur en mémoire post-login, évitant ainsi les accès répétitifs à `FlutterSecureStorage`.

### 1.3 Login & Inscription

- **Ancien fonctionnement :** Les méthodes `login`, `registerUser`, `registerParent` retournaient un booléen et une Map, sans gestion fine des erreurs.

- **Nouveau fonctionnement :**
    - `login()` retourne un objet typé `LoginResponse` structuré comme suit :

```
class LoginResponse {
  final bool success;
  final String? message;
  final User? user;
}
```

    - Meilleure gestion des erreurs avec `try/catch` et gestion des codes HTTP.
    - Inscription externalisée dans `AuthRepository` pour alléger le service principal.

### 1.4 Rafraîchissement automatique des tokens

- **Ancien fonctionnement :** Méthode `refreshToken()` appelée manuellement avant certains appels API.
- **Nouveau fonctionnement :**
    - Mise en place d'un middleware HTTP (via Dio ou `http_interceptor`) qui rafraîchit automatiquement le token expiré, rendant ce processus transparent pour l'appelant.

### 1.5 Déconnexion

- **Ancien fonctionnement :** Suppression simple des tokens et du userId dans `FlutterSecureStorage`.
- **Nouveau fonctionnement :**
    - La méthode `logout()` supprime également l'instance `CurrentUser`.
    - Possibilité d'émettre un évènement `AuthEvent.loggedOut` via un stream pour notifier l'ensemble de l'application, facilitant l'intégration avec Bloc ou Provider.

---

## 2. WebSocketService

### 2.1 Connexion WebSocket

- Implémentation robuste de la connexion WebSocket, intégrant l'authentification via JWT.

### 2.2 Envoi de messages

- Implémenter pour les trois acteurs : Élève, Parent, Enseignant.

### 2.3 Réception de messages

- Implémenter pour les trois acteurs : Élève, Parent, Enseignant.

### 2.4 Gestion des erreurs et stabilité

- Gestion améliorée des erreurs WebSocket (timeout, reconnexion automatique) pour garantir la continuité du service.

---

## 3. ChatScreen (pour les trois acteurs : Élève, Parent, Enseignant)

### 3.1 Gestion WebSocket

- **Ancien fonctionnement :** Un seul callback `onPrivateMessageReceived` limité.
- **Nouveau fonctionnement :**
  - Utilisation de la méthode `addMessageListener()` du `WebSocketService` permettant plusieurs listeners simultanés, offrant davantage de flexibilité.
  - Connexion WebSocket initialisée avec le `jwtToken` et `userId`.
  - Déconnexion explicite via `_webSocketService.disconnect()` dans la méthode `dispose()` pour libérer proprement les ressources.

### 3.2 Modèle de message

- Passage d'une construction manuelle en Map JSON vers l'utilisation d'une classe `ChatMessage` fortement typée pour améliorer la lisibilité et la maintenabilité.

### 3.3 Gestion des messages dans la liste

- L'ajout des messages reçus se fait maintenant en début de liste (`_messages.insert(0, ...)`) afin de gérer un affichage inversé typique des applications de messagerie.
- Optimisation de la compatibilité avec `ListView.builder(reverse: true)`.

### 3.4 Interface utilisateur (UI)

- Titre dynamique affichant le nom du contact avec une animation clignotante (`FadeTransition`) lors de la réception d'un nouveau message.
- Esthétique améliorée : bulles de message avec un rayon de bordure augmenté (18 au lieu de 16) pour une meilleure lisibilité.

### 3.5 Chargement et affichage des messages

- Les messages sont récupérés via `messagerieService.getPrivateMessages()`, puis inversés (`messages.reversed`) pour afficher les plus récents en haut de la liste.

### 3.6 Gestion des erreurs

- Affichage amélioré des erreurs avec un message explicite (`Text('Error: $_error')`) plutôt qu'un simple widget vide, facilitant le débogage.