Projet Mobile

Le projet final vise à faire évoluer votre travail sur le TP7 (Retrofit CRUD) vers une application mobile structurée et professionnelle. L'exigence principale est l'adoption stricte de l'architecture MVVM (Model-View-ViewModel). Vous devez démontrer une maîtrise totale de l'intégration des couches : la View (Fragments/Activities) observe l'état du ViewModel, qui interagit avec le Repository, seul responsable de la gestion des données. Concrètement, le projet doit impérativement intégrer un flux d'Authentification complet (Register & Login) avec consommation de l'API via Retrofit, et gestion de la session (stockage du jeton ou du statut) dans la base de données Room ou SharedPreferences. Toutes les opérations de CRUD pour les données principales doivent être fonctionnelles (GET, POST, PUT, DELETE) et mettre à jour l'interface utilisateur de manière réactive (via LiveData ou Flow), confirmant ainsi la séparation claire entre un AuthRepository et un EtudiantRepository (ou autres selon votre sujet) distincts. Ce travail sanctionne votre capacité à construire une application robuste, modulaire et prête pour la production.

NB: veuillez appliquer ces consignes pour toutes vos entités manipulées dans la partie mobile de votre projet

📝 Tableau d'Explication Détaillée des Critères				
Section	**Critère**	**Objectif / Rôle dans l'Architecture**	**Bonnes Pratiques à Vérifier**	**Note**
I. Architecture & Couches (40%)				
I.1. Séparation des Couches	Organisation MVVM	Assurer que le code est maintenable en séparant clairement l'UI, la logique, et les données.	L'Activity/Fragment ne doit appeler que les fonctions du ViewModel et observer les LiveData.	/10
I.2. Couche ViewModel	Isolation de la Vue	Gérer la logique d'affichage et l'état des données pour l'UI.	Utilisation du ViewModelScope pour lancer des coroutines. Utilisation de la ViewModelFactory (ou Hilt/Koin) pour l'instanciation.	/10

I.3. Couche Repository	Source de Vérité Unique	Abstraire les sources de données (API, DB locale) pour le ViewModel.	EtudiantRepository injecté dans EtudiantViewModel. AuthRepository injecté dans AuthViewModel. Le Repository gère la logique "faut-il appeler l'API ou lire le cache Room ?".	/10
I.4. Couche Data	Efficacité et Persistence	Assurer une communication réseau rapide et un stockage local fiable.	Retrofit configuré avec des suspend fun et des corps de requête/réponse clairs (@Body, @GET, etc.). Room utilisé pour la persistance des entités et du jeton (User).	/10
II. Logique Métier & Fonctionnalités (60%)				
II.1. Authentification	Accès Sécurisé	Valider l'identité de l'utilisateur auprès du serveur.	Le LoginRequest et la réponse sont gérés par le AuthRepository via l'API.	/10
II.2. Gestion de Session	Maintien de la Connexion	Permettre à l'utilisateur de rester connecté sans ressaisir ses identifiants.	Le token est stocké dans l'entité Room User ou stocké dans SharedPreferences. La MainActivity utilise authViewModel.authenticatedUser.observe pour conditionner l'affichage.	/10
II.3. CRUD	Manipulation des Données	Fonctionnalité complète pour gérer l'entité principale (Etudiant) et toutes autres entités du projet	Chaque opération CRUD correspond à la bonne méthode HTTP et est gérée asynchronément dans l'EtudiantRepository et autres...	/30
II.4. Gestion des Erreurs	Robustesse de l'Application	Empêcher les plantages et informer l'utilisateur des problèmes.	Les appels Retrofit sont encapsulés dans des blocs try-catch (pour les erreurs réseau) et vérifient response.isSuccessful (pour les erreurs HTTP 4xx/5xx).	/10