DAA - Laboratoire 6

January 13, 2024

Émilie Bressoud **Sacha Butty** **Loïc Herman**

Détails d'implémentation

Gestion des états

L'application utilise un système d'états pour gérer la synchronisation des contacts :

- **SYNCED** : Contact synchronisé avec le serveur
- **CREATED** : Nouveau contact en attente de synchronisation
- **UPDATED** : Contact modifié localement
- **DELETED** : Contact marqué pour la suppression avant la synchronisation

Gestion des communications résesau

Pour simplifier la gestion des communications HTTP, nous utilisons Retrofit. Une librairie créée par Square, les créateurs du client OkHTTP, et qui permet de définir un client à la volée en utilisant simplement une interface Java avec des méthodes annotées par les annotations de Retrofit. L'interface définira des méthodes suspensives qui s'intégreront parfaitement avec la gestion dans le repository décrite plus loin.

Opérations CRUD

Enrollment

Pour la connexion de l'utilisateur, l'application envoie une requête GET à l'endpoint d'enrollment retournant le UUID pour les appels suivants. Cet UUID est stocké par le ViewModel au moyen d'un système de `SharedPreferences`. L'UUID est stocké tel quel dans ce laboratoire, mais il pourrait être intéressant d'utiliser les utilitaires Android pour la cryptographie avec les `EncryptedSharedPreferences` pour plus de sécurité.

Création d'un contact

Lors de la création d'un contact, l'application privilégie l'expérience utilisateur en appliquant immédiatement les modifications en local. Le contact est d'abord sauvegardé dans la base de données locale avec l'état `CREATED`. Une tentative de synchronisation avec le serveur est ensuite effectuée. En cas de succès, l'application met à jour l'identifiant serveur et change l'état en `SYNCED`. Si la synchronisation échoue, le contact conserve son état `CREATED` pour permettre une synchronisation ultérieure.

Modification d'un contact

La modification suit une approche similaire à la création. Les changements sont immédiatement appliqués localement avec l'état `UPDATED`. L'application tente ensuite de synchroniser ces modifications avec le serveur. La réussite de cette opération entraîne le passage à l'état `SYNCED`, tandis qu'un échec maintient l'état `UPDATED` pour une synchronisation future.

Suppression d'un contact

Pour la suppression, le contact est d'abord marqué comme `DELETED` dans la base de données locale. L'application tente ensuite de le supprimer sur le serveur. En cas de succès, le contact est définitivement supprimé de la base locale (hard delete). Si la suppression côté serveur échoue, le contact reste marqué comme `DELETED` pour une tentative ultérieure.

Le marquage `DELETED` permet aussi d'identifier les contacts a ne plus afficher dans l'interface utilisateur.

Stockage local

La persistence des données est gérée via la bibliothèque Room. Nous utilisons une entité Contact qui comprend tous les champs nécessaires ainsi qu'un champ d'état pour la synchronisation. Room facilite la gestion des opérations CRUD et permet une observation réactive des changements via `LiveData`.

Stratégie de synchronisation

Notre application implémente l'approche « local-first » qui priorise la réactivité et l'expérience utilisateur. Les modifications sont appliquées immédiatement en local puis une tentative immédiate de synchronisation est faite avec le serveur. Les conflits sont gérés en conservant les états de synchronisation appropriés, permettant des tentatives ultérieures de synchronisation. En cas d'erreur HTTP, par exemple si le téléphone n'est pas connecté, le statut de synchronisation est gardé à la version la plus appropriée. Une nouvelle tentative de synchronisation sera effectuée si l'utilisateur fait une nouvelle modification sur le contact en question, ou si l'action de refresh est activée.

Gestion des appels réseau

Retrofit2 transforme automatiquement nos appels HTTP en méthodes Kotlin suspensives, permettant une utilisation naturelle avec les coroutines. Les annotations comme `@GET`, `@POST`, `@PUT` et `@DELETE` définissent le type de requête, tandis que `@Header` et `@Path` gèrent les paramètres dynamiques.

Conversion des données

La conversion entre JSON et objets Kotlin est gérée par le convertisseur Gson de Retrofit. Le DTOs assure une séparation claire entre les données réseau et le modèle local :

- **ContactDTO** : Représentation réseau d'un contact
- Mappers de conversion vers/depuis les entités locales

Gson est également configuré avec des adaptateurs pour gérer les dates et les UUIDs.

Gestion des erreurs

Les erreurs permettent de gérer les cas où la synchronisation échoue. Nous utilisons un simple try catch pour intercepter les exceptions et afficher des logs détaillés. Si la synchronisation échoue, l'application conserve le contact en local selon ce que l'utilisateur a effectué. Une nouvelle tentative de synchronisation sera effectuée si l'utilisateur modifie à nouveau le contact ou si l'action de refresh est activée.

Interface utilisateur avec Jetpack Compose

Notre application utilise Jetpack Compose, le toolkit moderne de Google pour la construction d'interfaces utilisateur natives Android. Cette approche déclarative simplifie considérablement le développement UI en comparaison avec les vues XML traditionnelles.

Architecture UI

Nous avons structuré l'interface en utilisant des composants réutilisables afin de facilité la maintenance et l'évolutivité de l'application.

- **ContactTextField** : Champ de saisie personnalisé pour les différentes informations de contact sous forme de TextField
- **ContactDateField** : Sélecteur de date pour la date de naissance
- **ContactPhoneTypeField** : Sélecteur du type de numéro de téléphone
- **EditButtons** : Groupe de boutons pour la gestion des contacts, qui change selon le type d'opération (création ou modification)

State Management

Nous utilisons le state management de Jetpack Compose pour gérer l'état de l'interface utilisateur. Avec `rememberSaveable`, nous conservons l'état du formulaire qui doit survivre aux changements de configuration de l'application, et `remember` est utilisé pour les états qui peuvent être recalculés (par exemple, l'état d'erreur pour les champs de texte).

Pour marquer l'édition d'un contact, nous utilisons une `MutableLiveData` dans le `ViewModel` qui s'occupe de stocker le contact en cours d'édition, ainsi que si l'application doit afficher le formulaire de création ou de modification.