Développement Android
Laboratoire n°6

Application communicante

20.12.2023

Introduction

Ce laboratoire consiste au développement d'une application *Android*, basée sur une architecture *MVVM* et mettant en œuvre une base de données locale synchronisée avec un service web. Nous allons en particulier voir la mise en place d'une interface permettant la création, la modification et la suppression de *Contacts* dans une base de données locale, ainsi que la synchronisation de celle-ci avec un serveur *REST*. Le laboratoire peut être réalisé soit avec des vues « traditionnelles » soit avec des vues générées avec *Jetpack Compose*.

Manipulations

1. Mise en place

Pour ce laboratoire nous vous fournissons le code d'une application basée sur une architecture *MVVM*, dans un état proche de celui que vous avez mis en place dans un laboratoire précédent. La première étape consiste donc à ouvrir le projet fourni dans *Android Studio*.

Il est également possible de réaliser ce laboratoire en mettant en place une interface basée sur *Jetpack Compose*, vous êtes libre de choisir l'une ou l'autre approche, nous vous fournissons le code de base des deux versions. Si vous utilisez l'approche *Compose*, veuillez vous référer aux slides de présentation du laboratoire.

2. Interface de création et d'édition d'un Contact

Cette étape consiste en la mise en œuvre d'une interface utilisateur permettant d'éditer et de modifier les *Contacts* présents dans la base de données locale. Dans le code fourni, vous remarquerez que lorsque l'on clique sur un *Contact* existant ou sur le *FloatingActionButton* un *Toast* apparait indiquant l'action à réaliser. L'application fournie consiste en une unique *Activité* accueillant un *Fragment* qui liste les contacts existants, nous souhaitons ajouter un second *Fragment* permettant l'édition. La Fig. 1 présente un exemple d'interface possible pour la création, l'édition/suppression d'un *Contact*.

Vous veillerez dans votre mise en œuvre à vous assurer que l'état de l'*Activité* et des *Fragments* soient cohérents lors d'une rotation de l'écran. Afin de se concentrer sur les manipulations essentielles, il n'est pas nécessaire que le champ *birthday* soit éditable par l'utilisateur dans ce laboratoire.

Hint : Il est conseillé d'utiliser une *LiveData* contenant soit *null*, soit le *Contact* à éditer

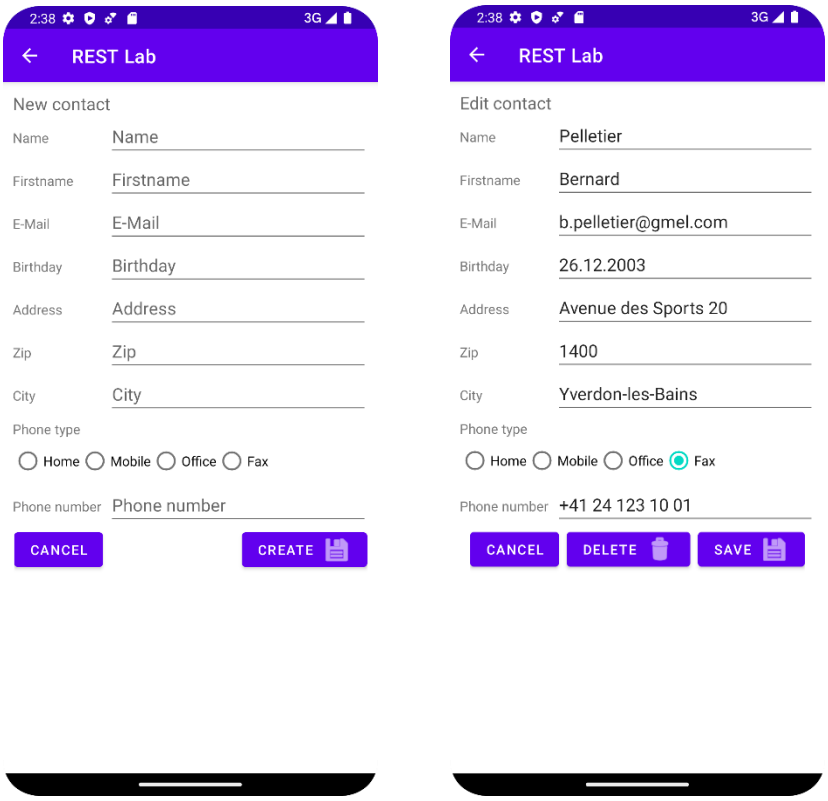

Figure 1 - Exemple d'interface d'édition d'un contact (à g. nouveau contact, à d. contact existant)

3. Préparation des entités pour la synchronisation

Nous souhaitons mettre en place un mécanisme simple de synchronisation des contacts avec un serveur : un unique client mobile et le serveur ne modifie pas les contacts. Il s'agit dans cette étape de modifier l'entité *Contact* en lui ajoutant les propriétés nécessaires à la mise en place de la synchronisation. Bien que nous modifions le schéma de la base de données, il n'est pas nécessaire ici de prévoir la migration des données, en cas d'incompatibilité vous pouvez désinstaller l'app et la relancer.

4. Communication avec le serveur

Pour ce laboratoire, nous avons mis en place un serveur offrant un service REST. Le serveur est accessible sur *Internet* à l'adresse https://daa.iict.ch/, sur cette page vous trouverez une interface affichant quelques logs du serveur, en particulier les appels à l'*API*. Pour garantir que les données ne soient pas modifiées par d'autres, chaque appel à l'API sera accompagné d'un « token », un *UUID* représentant le propriétaire d'un jeu de données. Les *UUIDs* sont attribués par le serveur avec un appel au endpoint */enroll* qui va créer un jeu de données (3 contacts) avec des valeurs par défaut et retourner un nouvel *UUID* ; pour ajouter, modifier ou supprimer des données il faudra ajouter cet *UUID* dans les en-têtes de chaque requête au endpoint */contacts*. Il est également possible d'utiliser l'*UUID* attribué

comme filtre dans l'interface du serveur pour voir uniquement vos requêtes. L'*UUID* est retourné sous la forme d'un simple texte, les contacts sont retournés sous la forme de tableaux ou d'objets *json*.

L'*API* mise à disposition est la suivante :

- ENROLL - Création d'un nouveau jeu de données et attribution d'un token
  GET https://daa.iict.ch/enroll
  Headers : Ø
  Payload : Ø
  Réponse : 420e384b-7082-464e-9afe-25f6e6319301

- CONTACTS – Obtenir tous les contacts
  GET https://daa.iict.ch/contacts
  Headers : X-UUID: 420e384b-7082-464e-9afe-25f6e6319301
  Payload : Ø
  Réponse : Tableau json contenant tous les contacts associés à l'UUID

- CONTACTS – Obtenir un contact spécifique
  GET https://daa.iict.ch/contacts/34
  Headers : X-UUID: 420e384b-7082-464e-9afe-25f6e6319301
  Payload : Ø
  Réponse : Objet json contenant le contact « 34 » pour autant qu'il appartienne à l'UUID indiqué

- CONTACTS – Créer un nouveau contact
  POST https://daa.iict.ch/contacts/
  Headers : X-UUID: 420e384b-7082-464e-9afe-25f6e6319301
  Content-Type: application/json
  Payload : Objet json du contact (id = null)
  Réponse : Objet json contenant le contact avec l'id attribué

- CONTACTS – Modifier un contact
  PUT https://daa.iict.ch/contacts/34
  Headers : X-UUID: 420e384b-7082-464e-9afe-25f6e6319301
  Content-Type: application/json
  Payload : Objet json du contact à modifier (id = 34)
  Réponse : Objet json contenant le contact modifié, erreur si pas bon UUID

- CONTACTS – Supprimer un contact
  DELETE https://daa.iict.ch/contacts/34
  Headers : X-UUID: 420e384b-7082-464e-9afe-25f6e6319301
  Payload : Ø
  Réponse : Ø, erreur si pas bon UUID

Voici un exemple d'un contact au format *json* :

```
{
    "id": 9070,
    "name": "Pelletier",
    "firstname": "Bernard",
    "birthday": "2003-12-26T23:00:00.000+00:00",
    "email": "b.pelletier@heig-vd.ch",
    "address": "Avenue des Sports 20",
    "zip": "1400",
    "city": "Yverdon-les-Bains",
    "type": "FAX",
    "phoneNumber": "+41 24 123 10 01"
}
```

A l'exception de l'id, tous les champs sont des chaines de charactères, *type* peut prendre une valeur parmis une énumération (HOME, OFFICE, MOBILE, FAX), *birthday* est une représentation textuelle d'un *Calendar* au format ISO : `"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"`.

4.1 Implémentation de l'inscription (enrollment)

Comme première fonctionnalité « communiquante » nous vous conseillons de débuter par la mise en place de l'inscription (enrollment), cette fonctionnalité sera accessible à l'utilisateur avec l'entrée du menu ( ) et consistera à supprimer toutes les données locales, à obtenir un nouvel *UUID* auprès du serveur et la récupération de tous les contacts correspondants à cet *UUID* et à leur stockage en local. Il faudra en particulier implémenter les coroutines permettant de réaliser les GET sur */enroll* et */contacts*. L'*UUID* devra être accessible pour les prochaines requêtes vers */contacts*, également en cas de redémarrage du smartphone. Vous détaillerez votre approche et les solutions mises en place dans votre rapport. Vous supprimerez dans l'application fournie l'ajout automatique de données lors de la création de la DB.

4.2 Création, modification et suppression de contacts

Pour l'étape suivante, nous allons mettre en œuvre les opérations CRUD pour un *Contact* individuel, il s'agit, sur l'application mobile, d'une création d'un nouveau contact, d'une modification ou encore d'une suppression. Ces opérations individuelles seront immédiatement appliquées en base de données locale, en particulier elles devront être immédiatement visibles dans l'app. Votre *Repository* essayera ensuite de l'appliquer sur le endpoint correspondant du serveur, en cas de succès l'objet local sera mis-à-jour, sinon l'objet restera dans un état « dirty » en local. Il s'agit de mettre en place une politique « best-effort », si la requête n'abouti pas il n'y a pas besoin de la replanifier automatiquement. Le point 4.3 consiste à effectuer une synchronisation intégrale des contacts.

Vous allez devoir utiliser les champs supplémentaires (de l'étape 3) de votre entité locale pour permettre la mise en œuvre du protocole de synchronisation avec le serveur. Vous détaillerez votre solution et sa mise en œuvre dans votre rapport, en particulier celle-ci doit permettre d'effectuer une synchronisation ultérieure si la première ne réussit pas, également l'utilisateur doit pouvoir apporter des modifications supplémentaires à un objet qui n'a pas pu être synchronisé avec le serveur.

Hint : N'hésitez pas à couper l'accès aux données de votre smartphone ou de l'émulateur pour faire échouer volontairement les requêtes de synchronisation (mode avion).

4.3 Synchronisation de tous les contacts

Il s'agit ici de mettre en œuvre une fonctionnalité permettant de lancer manuellement la synchronisation de toutes les entrées « dirty » de la base de données locale, celle-ci pourra être lancée par l'utilisateur en appuyant sur le menu (). Vous détaillerez votre approche et votre solution dans votre rapport.

Durée / Evaluation

- 6 périodes
- A rendre le lundi **29.01.2023** à **23h55** au plus tard.
- Pour rendre votre code, nous vous demandons de bien vouloir zipper votre projet Android Studio en veillant à bien supprimer les dossiers build (à la racine et dans app/) pour limiter la taille du rendu. Vous remettrez également un document **pdf** comportant les explications sur l'implémentation de votre solution.
- Merci de rendre votre travail sur *CyberLearn* dans un zip unique. N'oubliez pas d'indiquer vos noms dans le code, sur vos réponses et de commenter vos solutions.