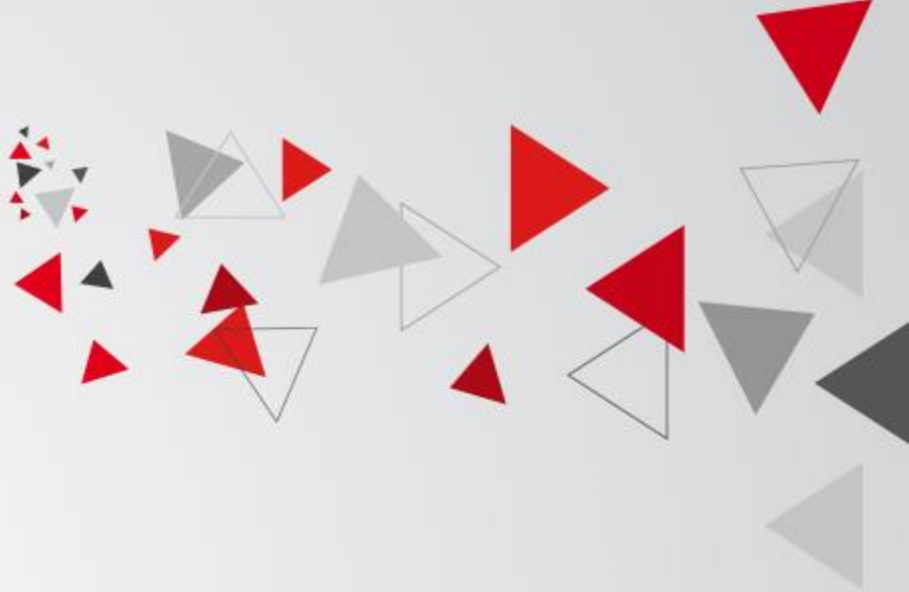
esprit

Se former autrement



Développement Mobile

Cours : Data storage

Année universitaire 2019/2020

cdio

CONCEIVE DESIGN IMPLEMENT OPERATE

Enropean Accreditation of Engineering Programmes EUR-ACE®

Délivré par Cti Commission des Titres d'Ingénieur

CONFÉRENCE DES GRANDES ÉCOLES

Stockage de données

- Il existe essentiellement quatre manières différentes de stocker des données dans une application Android:
  - Shared Preferences
  - Internal Storage
  - External Storage
  - SQLite database / Room

Shared Preferences

- On l'utilise pour sauvegarder des données primitives dans des paires clé-valeur.
- Les valeurs peuvent être l'un des éléments suivants: boolean, float, int, long ou string.
- En interne, la plateforme Android stocke les Shared Preferences d'une application dans un fichier xml situé dans un répertoire privé.
- Une application peut avoir plusieurs fichiers de Shared Preferences.
- Les Shared Preferences sont trop restrictives. Vous souhaiterez peut-être conserver des objets Java ou des images. Ou vos données doivent logiquement être conservées en utilisant la hiérarchie familière du système de fichiers.

Internal Storage

- La méthode de stockage de données interne est spécifiquement destinée aux situations dans lesquelles vous devez stocker des données sur le système de fichiers du périphérique, mais vous ne voulez pas qu'une autre application (même l'utilisateur) lise ces données.
- Les données stockées à l'aide de la méthode de stockage interne sont totalement privées pour votre application et sont supprimées du périphérique lors de la désinstallation de votre application.

External Storage

- Pour enregistrer (et / ou lire) des fichiers sur la mémoire de stockage externe de l'appareil, votre application doit demander l'autorisation WRITE_EXTERNAL_STORAGE. Si vous souhaitez uniquement lire à partir du stockage externe sans écrire, demandez l'autorisation READ_EXTERNAL_STORAGE.
- L'autorisation WRITE_EXTERNAL_STORAGE accorde les deux accès en lecture / écriture. Cependant, à partir d'Android 4.4, vous pouvez réellement écrire dans un dossier de stockage externe «privé» sans demander WRITE_EXTERNAL_STORAGE.
- Le dossier «privé» peut être lu par d'autres applications et par l'utilisateur. Toutefois, les données stockées dans ces dossiers ne sont pas analysées par le scanner de supports. Ce dossier app_private se trouve dans le répertoire Android / data et est également supprimé lors de la désinstallation de votre application.

SQLite database / Room

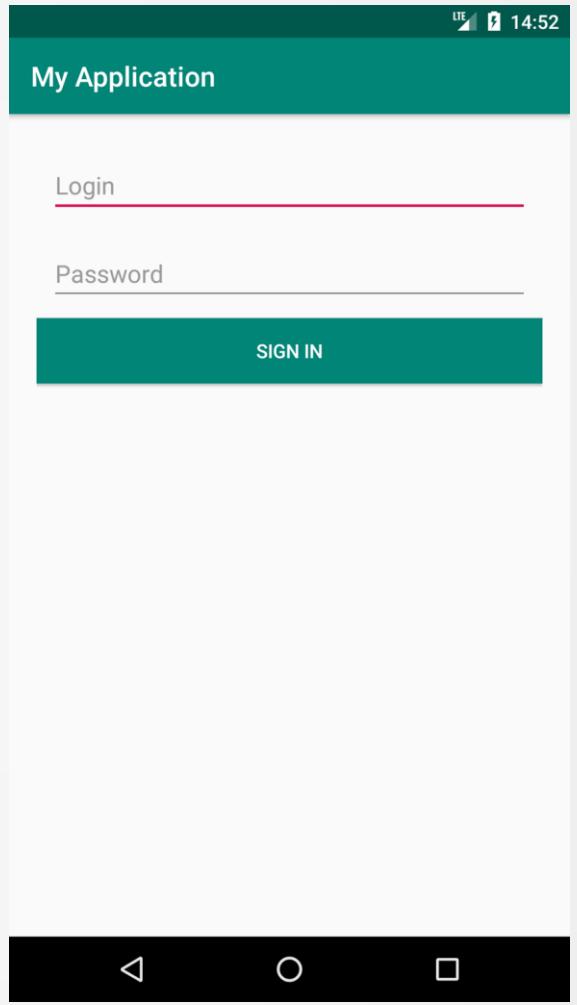
- Android prend en charge les applications qui utilisent des bases de données SQLite pour le stockage de données.
- Les bases de données créées sont spécifiques à chaque application et sont disponibles pour toutes les classes de l'application, mais pas pour les applications externes.
- Room est une couche de base de données au-dessus de SQLite. Elle prend en charge les tâches banales de gestion qu'on utilise avec une classe d'assistance (SQLiteOpenHelper).
- Room utilise DAO pour émettre des requêtes à la base de données SQLite en se basant sur les fonctions appelées.
- *DAO:* Un mappage des requêtes SQL aux fonctions. Lorsque vous utilisez un DAO, votre code appelle les fonctions et les composants s'occupent du reste.

Shared Preferences

Shared Preferences

- Si nous avons une application qui demande une authentification pour chaque ouverture, l'utilisateur doit entrer le login et le mot de passe a chaque fois.
- Pour aider l'utilisateur a mieux utiliser notre application, on va sauvegarder le login et le mot de passe dans un fichier SharedPreferences. Une fois l'utilisateur s'authentifie, son login et son mot de passe seront enregistrer dans ce fichier.
- Ensuite, chaque fois l'utilisateur ouvre l'application, celle-ci récupère automatiquement le login et le mot de passe dans l'interface d'authentification.

Shared Preferences



- Commencer par créer un nouveau projet
- Modifier activity_main.xml en ajoutant deux EditText et un Bouton.
- Dans MainActivity.java on va ajouter un objet de SharedPerferences et une variable String qui contient le nom du fichier de l'objet SharedPreferences

```
private SharedPreferences mPreferences;
public static final String sharedPrefFile = "tn.esprit.myapplication";
```

- Et dans la méthode onCreate on va instancier notre objet:

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
```

La méthode getSharedPreferences() ouvre le fichier au nom donné (sharedPrefFile) avec le mode MODE_PRIVATE.

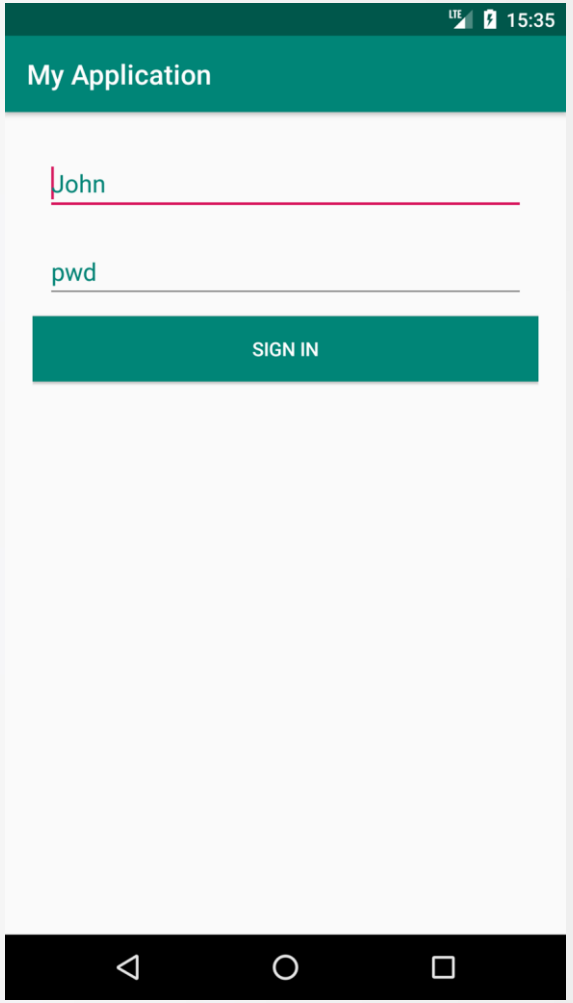
Shared Preferences

- Pour récupérer des données de notre objet mPreferences on utilise l'une des méthodes "get" tel que getInt() ou getString(). Dans la méthode onCreate :

```
mLogin.setText( mPreferences.getString("login","") );
mPassword.setText( mPreferences.getString("password","") );
```

Notez que la méthode getString() prend deux arguments: un pour la clé et l'autre pour la valeur par défaut si la clé est introuvable.

NB : mLogin et mPassword sont deux variable de type EditText supposer déjà créer.



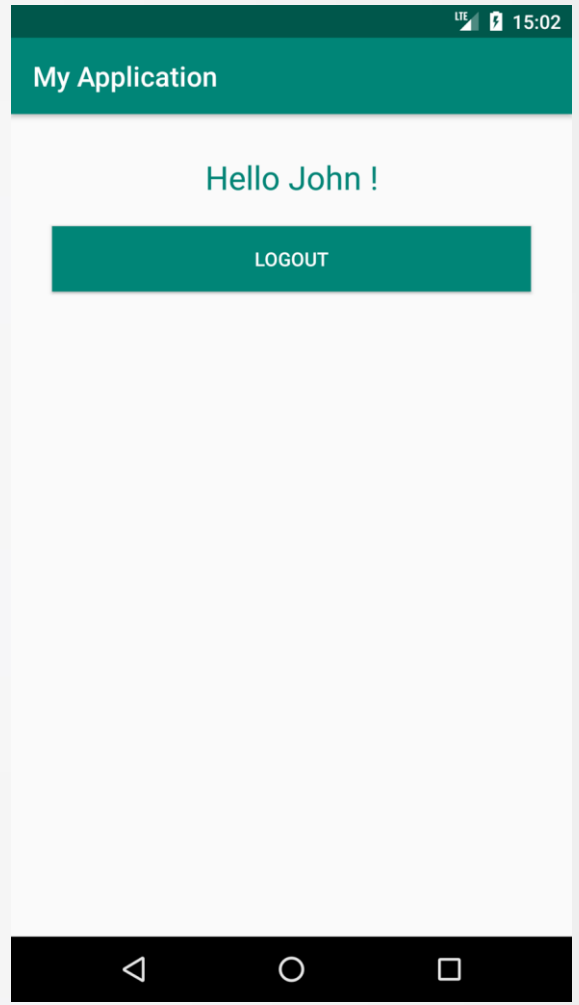
Shared Preferences

- Pour sauvegarder des données de notre objet mPreferences on utilise SharedPreferences.Editor. Dans la méthode onClick du bouton :

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
preferencesEditor.putString("login", mLogin.getText().toString());
preferencesEditor.putString("password", mPassword.getText().toString());
preferencesEditor.apply();
```

La méthode apply() enregistre les préférences de manière asynchrone à partir du thread d'interface utilisateur. SharedPreferences.Editor dispose également d'une méthode commit() pour enregistrer les préférences de manière synchrone. La méthode commit() est déconseillée car elle peut bloquer d'autres opérations.

Shared Preferences



- Le click sur le bouton SIGN IN permet d'enregistrer le login et le mot de passe aussi il ouvre une nouvelle activité qui contient un simple TextView et un bouton.
- Le clic sur le bouton LOGOUT permet de supprimer les préférences enregistrer.
- Dans le code java de la deuxième activité on ajoute une variable:

```
private SharedPreferences mPreferences;
```

- Ensuite dans le code du bouton LOGOUT :

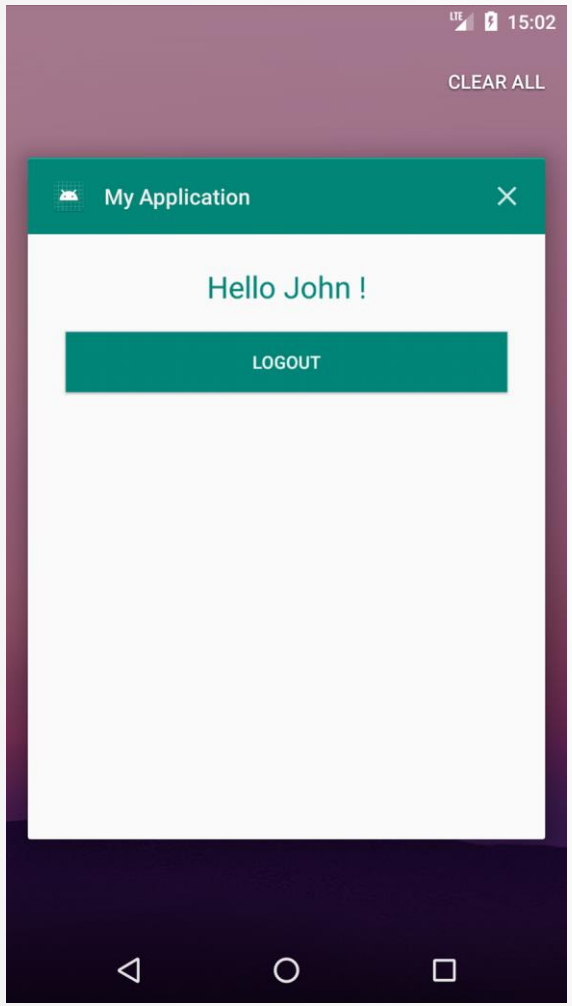
```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
preferencesEditor.clear();
preferencesEditor.apply();
finish();
```

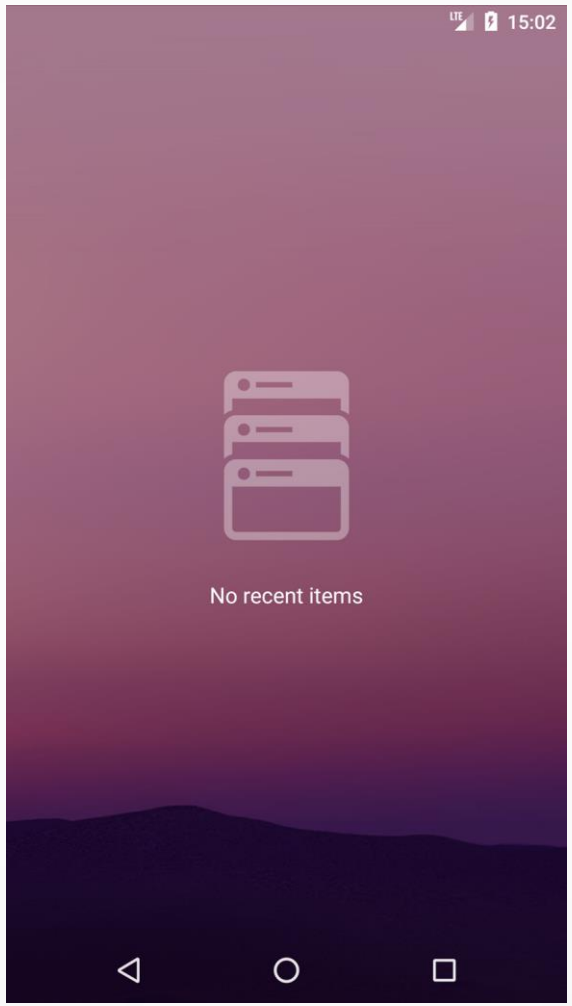
NB: La méthode clear() supprime toutes les préférences partagées

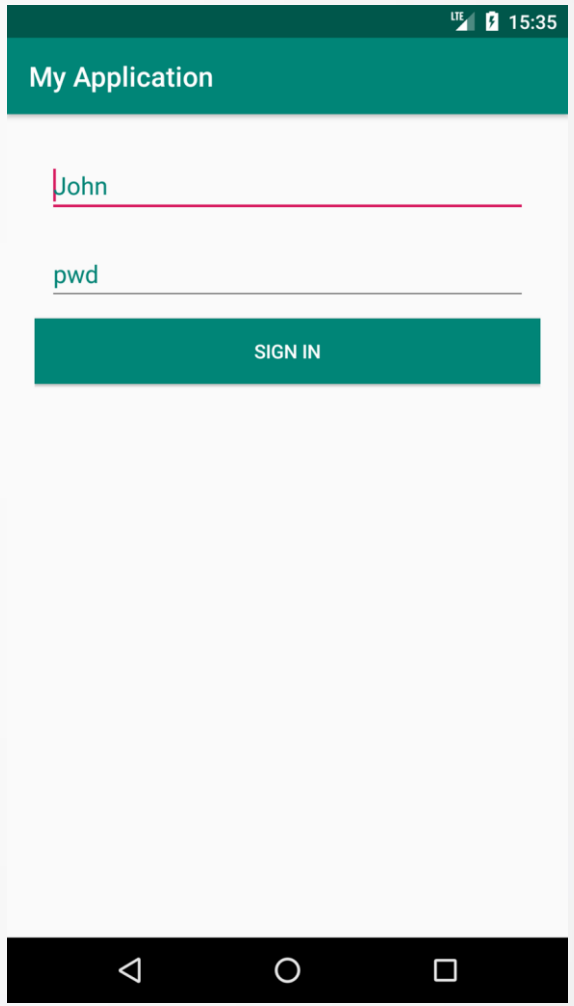
Shared Preferences

- Pour tester, n’oublier pas de forcer la fermeture de l'application en utilisant l'une des méthodes suivantes:
  - Dans Android Studio, sélectionnez **Exécuter> Arrêter l'application.**
  - Sur l'appareil, appuyez sur le bouton Recents (le bouton carré dans le coin inférieur droit). Glissez la carte de l'application pour quitter l'application ou cliquez sur le X dans le coin droit de la carte.

Shared Preferences

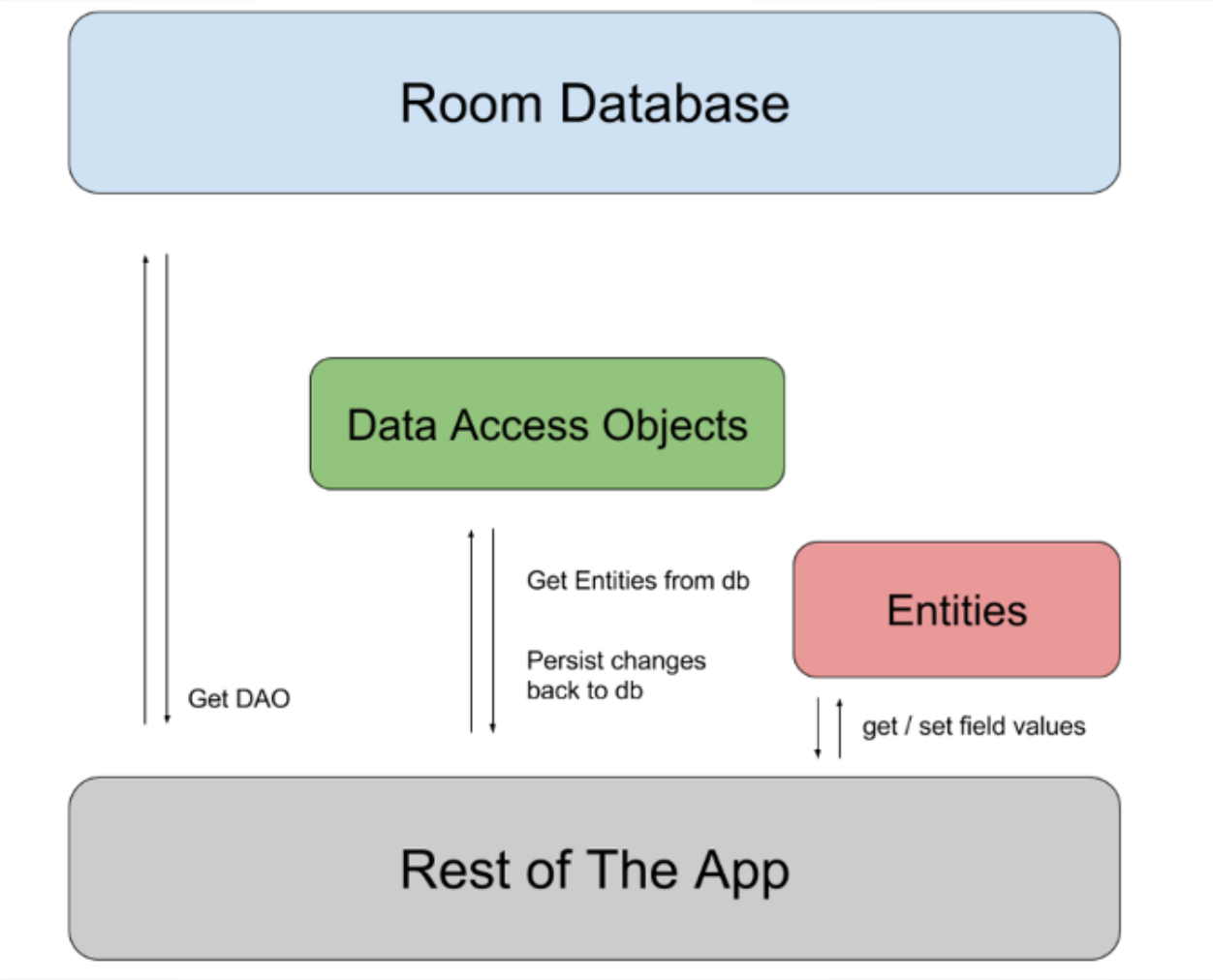






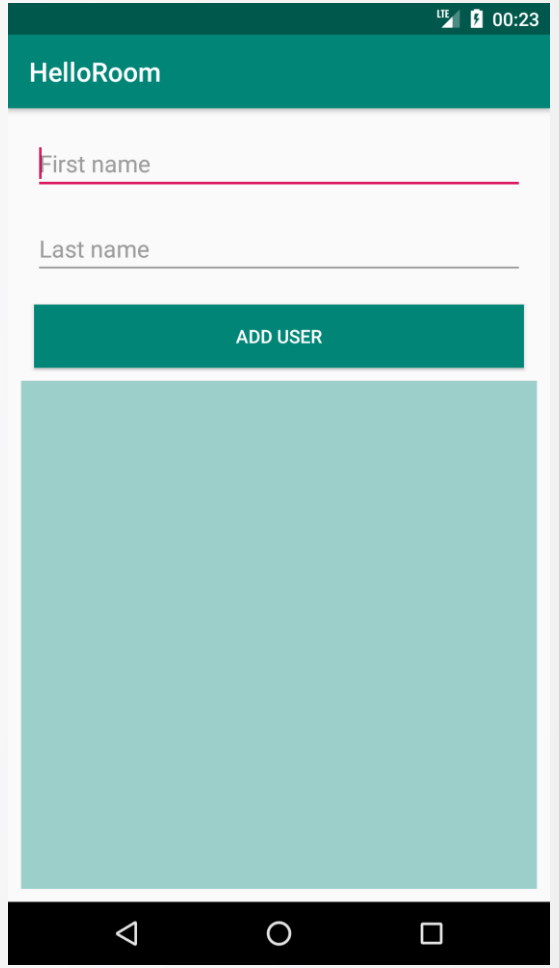

Room database

Architecture Room



Room database

- Créer un nouveau projet intitulé HelloRoom
- Modifier activity_main.xml comme le montre la figure :
  - Deux EditText
  - Un Bouton
  - Un RecyclerView

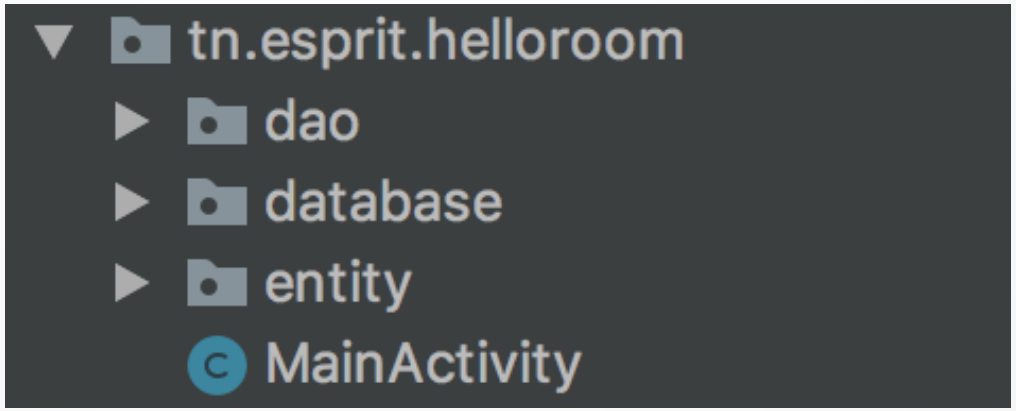


Room database

- Pour utiliser Room il est nécessaire d'importer ces dépendances

```
implementation 'androidx.room:room-runtime:2.2.0'
annotationProcessor 'androidx.room:room-compiler:2.2.0'
```

- Ensuite, préparer les packages nécessaires



Room database

- Ajouter une class User dans le package entity
- Ajouter une annotation a cette class pour dire que c'est une table

```
@Entity(tableName = "user_table")
public class User { }
```

- Ajouter les attributs avec leur annotations comme suit

```
@PrimaryKey(autoGenerate = true)
private int uid;
@ColumnInfo(name = "first_name")
private String firstName;
@ColumnInfo(name = "last_name")
private String lastName;
```

N'oublier pas les getters et le setters

Room database

- Dans le package Dao créer une interface UserDao avec l'annotation @Dao

```
@Dao
public interface UserDao { }
```

- Ajouter les méthodes suivant

```
@Insert
void insertOne(User user);

@Delete
void delete(User user);

@Query("SELECT * FROM user_table")
List<User> getAll();
```

Room database

- Dans le package database, créer une class abstraite AppDataBase qui hérite de RoomDatabase avec l'annotation @Database

```
@Database(entities = {User.class}, version = 1, exportSchema = false)
public abstract class AppDataBase extends RoomDatabase { }
```

- Nous allons maintenant appliquer le patron de conception singleton dont l'objectif est de restreindre l’instanciation de cette class à un seul objet

Room database

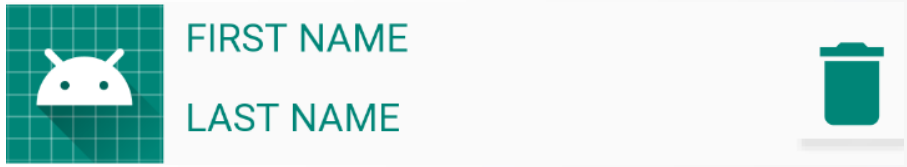
```
private static AppDataBase instance;
public abstract UserDao userDao();
public static AppDataBase getAppDatabase(Context context) {
   if (instance == null) {
      instance = Room.databaseBuilder(context.getApplicationContext(),  AppDataBase.class, "room_test_db")
                        .allowMainThreadQueries()
                        .build();
   }
   return instance;
}
```

NB: la méthode allowMainThreadQueries() n'est pas conseiller car il pourrait verrouiller l'interface utilisateur pendant une longue période **si** la base contienne beaucoup de données.

Il faut utiliser les AsyncTask qui sont des taches asynchrones qui s'exécutent sur un thread en arrière-plan

Room database

- Créer maintenant une layout single_row pour avoir un affichage comme suit



- Ajouter un adaptateur UsersAdapter pour notre RecyclerView avec une List<User> users;

Room database

- Dans MainActivity.java ajouter les variables suivantes :

```
private AppDataBase database ;
private UsersAdapter usersAdapter;
private List<User> userList = new ArrayList<User>();
```

- Dans la méthode onCreate nous allons récupérer l'instance et récupérer la liste des utilisateurs dans la base:

```
database = AppDataBase.getAppDatabase(this);
userList = database.userDao().getAll();
usersAdapter = new UsersAdapter(getApplicationContext(), userList);
```

Room database

- Pour ajouter un nouveau utilisateur, voici un exemple du click

Ajouter:

```
User user = new User(mFirstName.getText().toString(),
mLastName.getText().toString());
database.userDao().insertOne(user);
usersAdapter.notifyChange(database.userDao().getAll());
```

- La méthode notifyChange permet de mettre à jour l'affichage de la vue avec la nouvelle liste

```
public void notifyChange(List<User> users){
        this.users = users;
        this.notifyDataSetChanged();
}
```

Room database

- Le click sur le bouton supprimer permet d'effacer l'utilisateur de la base et de la liste de l'adaptateur.

```
final User singleItem = users.get(position);
holder.btn_delete.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                AppDataBase.getAppDatabase(mContext).userDao().delete(singleItem);
                UserAdapter.this.notifyChange(AppDataBase.getAppDatabase(mContext).userDao().getAll());
        }
});
```

Room database