Persistance de donnée

SQLITE

Méthode de persistance de données

Préférences partagées (SharedPreferences)

- Sauvegarder les paramètres et l'état de l'application sous la forme de clé/valeurs dans un fichier de préférences partagé
- Une solution simple de persistance de données

SQLite

- SQLite est intégré dans le système android,
- L'API de gestion des bases de données SQLite est définie dans le package **android.database.sqlite**
- androidx.room.Room est une API de mapping objet relationnel qui constitue une couche d'abstraction pour les bases de données sqlite dans Android (depuis 2018)

SharedPreferences

SharedPreferences: une collection locale de persistance de données de type dictionnaire.

Configuration

Ajouter la dépendance suivante dans la section « dependencies » dufichier build.gradle (Module app) :

```
def preference_version  ="1.1.1"

implementation "androidx.preference:preference-ktx:$preference_version"
```

Cliquer sur « Sync now »

Ecriture

```
val sharedPref = sharedPreferences.edit().putString("cle", "valeur")
sharedPref.apply()
```

Lecture

```
val valeur = sharedPreferences.getString("valeur", "vide")
```

SQLite

SQLite est un SGBDR embarqué et open source.

Caratéristiques

- SQLite est implémenté sous la forme d'une bibliothèque écrite en C.
- Utilise un typage dynamique
- Types d'affinités
  - TEXT
  - NUMERIC: la donnée est enregistrée comme un entier ou un réel si c'est possible (sinon elle sera enregistrée comme TEXT)
  - INTEGER: (entier, REAL,TEXT)
  - REAL
  - NONE: la donnée est enregistrée sans conversion.
  - INTEGER PRIMARY KEY: correspond à ROWID adresse de l'enregistrement dans la table.

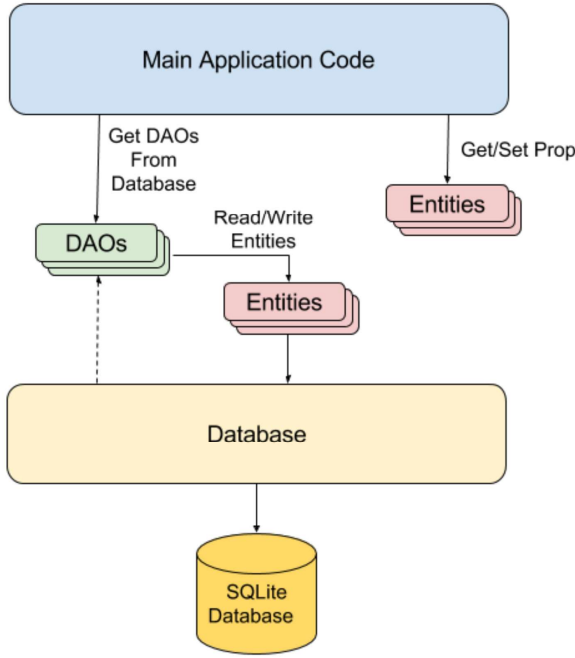
Une base de données est privée et n'est accessible que par l'application qui l'a créée.

Room

Room est une bibliothèque de persistance de données pour la base de donées SQLite intégrée dans le s et elle constituée de 3 composantes:

-

- @Database : Une classe abstraite annotée par Database et qui dérive de RoomDatabase représentela base de données Sqlite, une instance de cette classe est créée par Room.databaseBuilder().
- @Entity: une classe entité qui est annotée par Entity représente une ligne dans une table dont les colonnes sont créées à partir des attributs de cette classe.
- @Dao: cette annotation une classe ou une interface comme un objet d'accès aux données qui définit les opérations CRUD sur une entité



Intégration de room dans un projet

Dans le fichier build.gradle (Module app): ajouter les dépendances:

```
def room_version = "2.2.6"
implementation
"androidx.room:room-runtime:$room_version"
kapt
"androidx.room:room-compiler:$room_version"
```

```
apply plugin: 'kotlin-kapt'
dependencies {

def room_version = "2.2.6"
implementation "androidx.room:room-runtime:$room_version"
kapt "androidx.room:room-compiler:$room_version"
```

@Entity

L’ annotation @Entity possède les attributs suivants:

- tableName, foreingkeys, indices, primarykeys

Les arguments du constructeur et les attributs de la classe deviennent des colonnes dans la table sqlite associée

```
import androidx.room.Entity
import androidx.room.PrimaryKey
@Entity
data class Mot (val nom:String,val type_mot:String="",val indice:Int ){
    @PrimaryKey(autoGenerate = true)
    var id:Long?=null
}
```

@Dao

Définit les méthodes CRUD pour une entité.

```
interface MotDao {
@Query( value: "Select * from Mot")
fun getMots():List<Mot>
@Query( value: "Select * from Mot where id=:idMot")
fun getMot(idMot:Long):Mot
@Insert(onConflict = IGNORE)
public fun addMot(m:Mot):Long
@Update(onConflict = REPLACE)
fun updateMot(m:Mot)
@Delete
fun deleteMot(m:Mot)
    @Query( value: "Select * from Mot where id=:idMot")
    fun getMotLive(idMot:Long):LiveData<Mot>

}
```

@Database

- entities est une collection des entités dans la base de donnéérées par room

  - La classe doit être abstraite et doit dériver de RoomDatabase

  - La méthode abstraite motDao retourne une une instance de type MotDAO, l'implémentation de la méthode est prise en charge par room

```
@Database(entities = arrayOf(Mot::class),version = 1,exportSchema = false)
abstract class MotsDatabase : RoomDatabase() {
    abstract fun motDao():MotDAO
    companion object{
        @Volatile
        private  var instance: MotsDatabase? = null
        fun getInstance(context: Context): MotsDatabase {
            if (instance==null){
                instance= Room.databaseBuilder(context.applicationContext,MotsDatabase::class.java, name: "MotsDB")
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build()
            }
            return instance as MotsDatabase
        }
    }

}
```

La classe Repository

```
class MotRepository(context:Context) {
    private var db=MotsDatabase.getInstance(context)
    private val motDao=db.motDao()
    fun addMot(m: Mot):Long{
        val newId=motDao.addMot(m)
        return  newId
    }
    fun getMots():List<Mot>{
        return motDao.getMots()
    }
    fun getMotsLive():LiveData<List<Mot>>{
        return motDao.getMotsLive()
    }
    fun delete (m:Mot){
        motDao.deleteMot(m)
    }
    fun getMot(id:Long):Mot
    {
        return motDao.getMot(id)
    }

    fun updateMot(m:Mot){

    }
}
```