# I.2 - Room (DB)

## Scopo del DB

Lo scopo principale del database in un'app Android è di limitare le richieste network e consentire all'utente di utilizzare l'app anche quando la rete non è disponibile – questo concetto è definito **Offline First**

Un database si utilizza quando è necessario memorizzare dati strutturati e sincronizzati.

Android utilizza una libreria ufficiale per implementare il database, chiamata **Room**

Una implementazione di Room è composta da 3 elementi: **Data Entity, DAO e Database**

## Room setup

### Build.gradle:app

plugins **{**

...  
 id 'kotlin-kapt'   
**}**

//Room  
def room\_version = "2.5.1"  
implementation "androidx.room:room-runtime:$room\_version"  
kapt "androidx.room:room-compiler:$room\_version"

Alla fine del file ( dopo *dependencies{} )* aggiungere a livello root:

// Allow references to generated code  
kapt **{** correctErrorTypes true  
**}**

### Data Entity

Room utilizza le data class per definire tabelle e record

A screen shot of a computer

Description automatically generated with medium confidence

Questo codice definisce un'entità dati Movie. Ogni istanza di Movie rappresenta una riga in una tabella *movie* nel database dell'app.

@Entity definisce una data entity

Il nome della classe Movie sarà il nome della tabella.

I campi di Movie saranno i nomi delle colonne della tabella.

@PrimaryKey è un campo *mandatory* di ogni entity, definisce la primary key nel db.

### Type Converters

Room non converte da solo gli oggetti: è lasciato a noi definire come si vogliono salvare oggetti complessi nel db.

Nel nostro Movie c’è una List<Int> che non puo’ essere salvata automaticamente.

Per adesso, usiamo una soluzione semplice: convertiamo un oggetto in stringa con GSON.

Crea *db/TypeConverters.kt* con questo contenuto

class MovieListTypeConverter {  
 private val gson = Gson()  
 private val movieListType = object : TypeToken<List<Int?>?>() {}.*type* @TypeConverter  
 fun toIntList(data: String?): List<Int> =  
 data?.*let* **{** gson.fromJson(**it**, movieListType) **}** ?: *emptyList*()  
  
 @TypeConverter  
 fun intListToString(data: List<Int?>?): String =  
 gson.toJson(data)  
}

### Data access object (DAO)

Il DAO è un’interfaccia che definirà le funzioni con le quali interagirà il resto dell’app.

Crea un nuovo file *data/db/MovieDB*

A screenshot of a computer

Description automatically generated with medium confidence

Il suo contenuto:

A screen shot of a computer program

Description automatically generated with low confidence

* Le annotazioni Insert, Update e Delete, sono delle abbreviazioni per i relativi metodi di query
* In query è necessario scrivere il codice SQL

### Database

Dentro *MovieDB* aggiungi:

A screen shot of a computer

Description automatically generated with medium confidence

* La classe deve essere annotata con un'annotazione @Database che include un array di entità che elenca tutte le entità di dati associate al database.
* @TypeConverters aggiunge uno o piu’ convertitori per oggetti complessi.
* La classe deve essere una classe astratta che estende RoomDatabase.
* Per ogni classe DAO associata al database, la classe database deve definire un metodo astratto con zero argomenti e restituire un'istanza della classe DAO.

### Implementazione, Modulo Hilt Singleton

Per l’occasione, crea un nuovo modulo Hilt *di/DBModule*

A screen shot of a computer program

Description automatically generated with low confidence

* Installiamo questo modulo nel container Application, rendendolo un singleton. L’istanza di un DB è piuttosto pesante, e avere piu’ copie del DB potrebbe creare problemi di sincronizzazione. Utilizziamo quindi un singolo DB.

Ora sorge un problema: abbiamo fornito all’app il repository MoviesRepositoryImpl tramite provideMovieRepository nel modulo DataModule, nel quale però non è presente provideDB, che appartiene al nuovo DBModule.

DataModule è scoped al viewModel: quindi se vogliamo mantenere il suo scope, non possiamo cambiarlo in singleton. Decidiamo quindi che abbia un riferimento al DBModule.

A DBModule.provideDB aggiungi questa annotation:

A picture containing text, screenshot, font

Description automatically generated

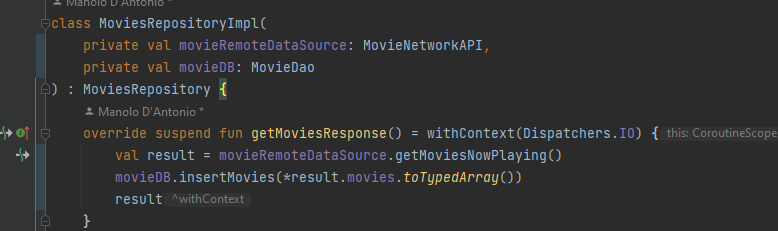
Questo ci permette di richiamare direttamente la iniezione. In DataModule fai questa modifica:

A screen shot of a computer program

Description automatically generated with low confidence

* **@Module(includes = [DBModule::class]):** DataModule ora ha visibilità sulle iniezioni di DBModule, mantenendo comunque il suo ciclo legato al ViewModel
* **@Named("provideDB"):** richiama direttamente l’iniezione

### Repository



* insertMovie ha come input un **vararg**: uno o piu elementi separati da una virgola. Per passare gli elementi della lista, la convertiamo in array. Come parametro di insertMovies non inviamo l’array, ma i singoli elementi contenuti nell’array, grazie al simbolo \*. [Documentazione](https://kotlinlang.org/docs/functions.html#variable-number-of-arguments-varargs)

Dopo aver recuperato la lista di movies dal servizio, facciamo in modo di salvarla nel db prima di ritornarla al viewModel.

Questo apre la strada ad una serie di strategie per far si che l’utente possa utilizzare l’app anche senza connessione a internet.

1. getMoviesResponse potrebbe leggere i dati di una nuova val moviesCache: List<Movie>, che contiene i dati dell’ultima lista inviata al viewModel durante questa sessione. Se moviesCache è vuota, prova prima a scaricare i movies dal servizio, ma se non ce la fa, li carica dal DB.
2. Un’evoluzione potrebbe essere di scaricare i movies dal servizio solo una volta al giorno, a determinati orari: è infatti improbabile che la lista di “nuove uscite” cambi piu’ di una o due volte a settimana.

## Esercizio

Aggiungi moviesCache a MoviesRepositoryImpl e fa si che l’app si comporti come nell’esempio a)

* Basta disattivare il wifi e dati mobili (o modalità aereo) per testare una situazione senza internet
* Intercetta gli errori tramite try\catch
* Opzionale: controlla che la rete sia disponibile prima di effettuare la chiamata alla data source.