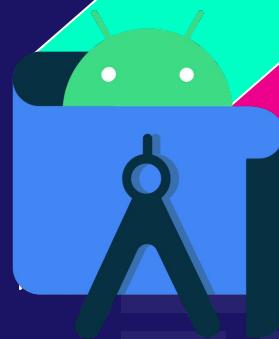


#11

Android Studio



Android Studio

" Data disappears on reboot,
How can we work with data more efficiently? "

INNEHÅLLSFÖRTECKNING

01

Översikt

03

ROOM &
Persisting Data

02

Dependencies,
Threads &
Project Setup

04

Uppgifter
&
Övningar



01

ÖVERSIKT



```
curl_easy_setopt(comm, CURLOPT_URL, url);  
if (curl_easy_perform(comm) != CURLE_OK)  
{  
    fprintf(stderr, "Failed to set URL [%s]\n", errorbuf);  
    return 1;  
}  
  
curl_easy_setopt(comm, CURLOPT_FOLLOWLOCATION,  
                 1);  
if (curl_easy_perform(comm) != CURLE_OK)  
{  
    fprintf(stderr, "Failed to set redirect option [%s]\n", errorbuf);  
    return 1;  
}  
  
curl_easy_setopt(comm, CURLOPT_WRITEFUNCTION,  
                 curl_write_callback);  
if (curl_easy_perform(comm) != CURLE_OK)  
{  
    fprintf(stderr, "Failed to set writer [%s]\n", errorbuf);  
    return 1;  
}
```

ROOM

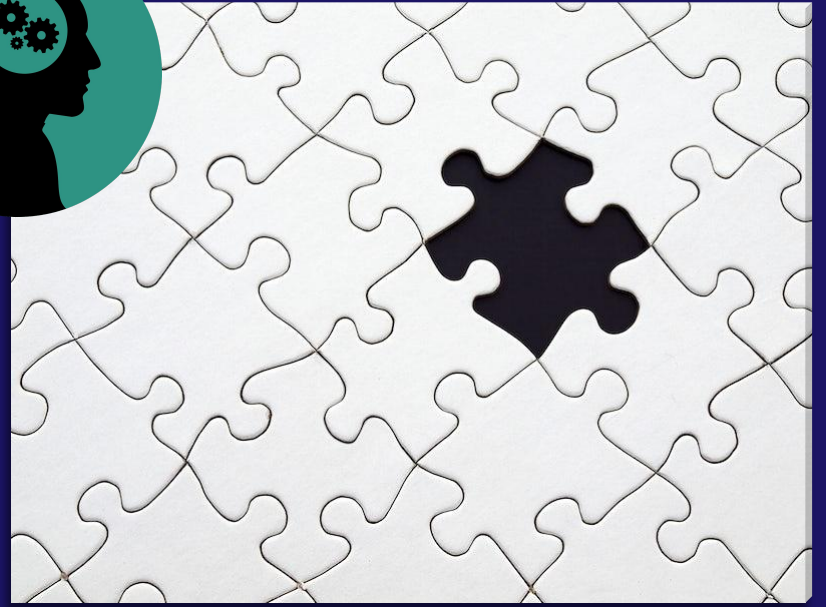


ROOM DB

- Spara ner data lokalt inom mobila enheter
- Byggt upp på sqLITE
- Enkel uppsättning!



Frågor?





02

Dependencies, Threads
&
Project Setup

Dependencies



Dependencies



Copy paste the following:

```
implementation "androidx.room:room-runtime:2.5.0"
androidTestImplementation "androidx.room:room-testing:2.5.0"

// To use Kotlin annotation processing tool (kapt)
kapt "androidx.room:room-compiler:2.5.0"

// To use Kotlin Symbol Processing (KSP)
kapt "androidx.room:room-compiler:2.5.0" // DUPLICATE???

implementation "androidx.lifecycle:lifecycle-runtime-ktx:2.5.1"
```

Apply 'kapt'

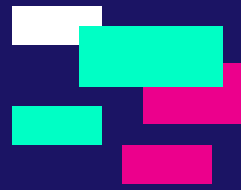


Copy paste the following:

```
plugins {  
    id 'com.android.application'  
    id 'org.jetbrains.kotlin.android'  
  
}  
  
apply plugin: 'kotlin-kapt'
```

Detta är överst på filen, om vi inte lägger till detta kommer inte 'kapt' dependencies att fungera.

" If you are using a newer version of Android Studio (3.0 or higher), it is recommended to use kapt instead of annotationProcessor for Room annotation processing. "





```
curl_easy_setopt(comm, CURLOPT_URL, url);  
if (curl_easy_perform(comm) != CURLE_OK)  
    fprintf(stderr, "Failed to set URL [%s]\n", errorbuf);  
else;  
    curl_easy_setopt(comm, CURLOPT_FOLLOWLOCATION,  
        1);  
    curl_easy_setopt(comm, CURLOPT_WRITEFUNCTION,  
        write_callback);  
    curl_easy_setopt(comm, CURLOPT_WRITEDATA, &data);  
    curl_easy_perform(comm);  
    if (curl_easy_getinfo(comm, CURLINFO_RESPONSE_CODE, &code) != CURLE_OK)  
        fprintf(stderr, "Failed to set writer [%s]\n", errorbuf);  
    else  
        printf("Response code: %d\n", code);  
}
```

ROOM?





`“The Room persistence library provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite. “`

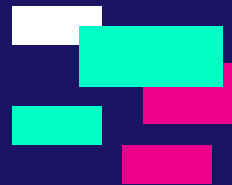
- Android.developers

developers 

ROOM

"Apps that handle non-trivial amounts of structured data can benefit greatly from persisting that data locally."

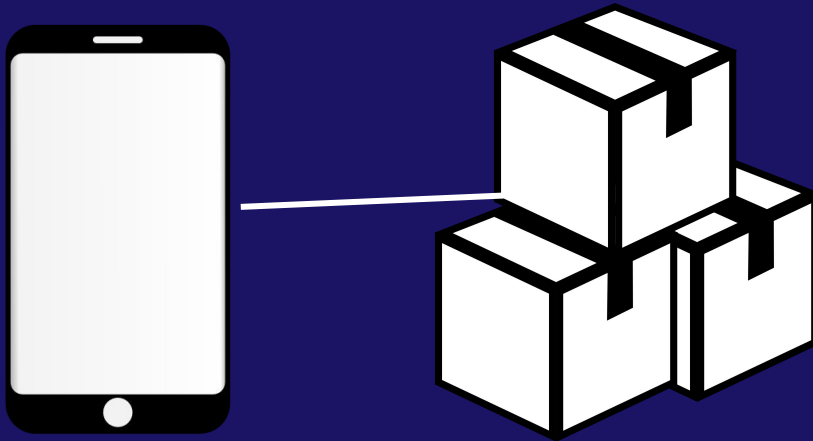
The most common use case is to cache relevant pieces of data so that when the device cannot access the network, the user can still browse that content while they are offline."

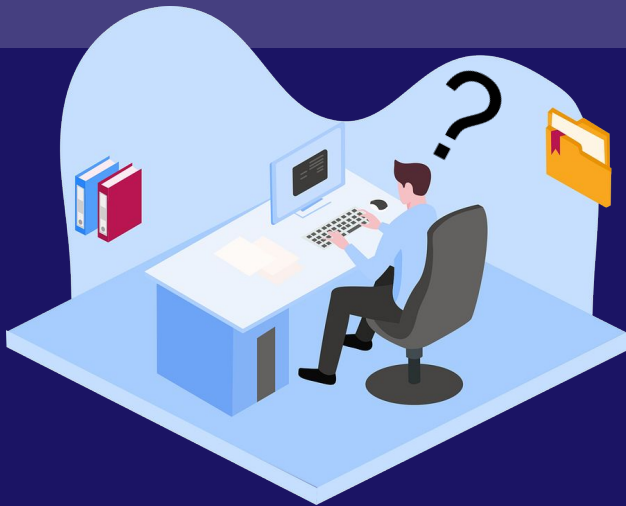


ROOM DB SQLITE Info



Maximum Database Size: 140 tb, but it will depend on your device disk size





Problem

Vad är 'ROOM DB'?



Solution

Room DB är ett 'persistent' bibliotek som tillhandahåller ett abstraktionslager över SQLite-databas för Android-appar.



Threads





Threads...



IO Threads

Main Threads

Dessa är något vi kommer arbeta med...

Men vad är dessa?

Stockholm!

När man pratar om 'Threads', handlar det om hur data bearbetas och används genom applikationen.

Låt oss ta Stockholm som en reflektion inom vår analogi!



THE UI

Vad kan vi säga om Stockholm i samband med 'Threads'... ?

- Skyltar
- Dörrar
- Affärer
- Nyheter, reklam
- Saker vi kan interagera med!
- Saker som fångar vår uppmärksamhet

Enkelt sagt - något som liknar ett interface.

(Låter lite som knappar, textvyer och inputs)



Main Thread

Vad kan vi säga om gatorna?

- Fotgängare
- Cyklister
- Mopedister
- Mindre fordon
- Enklare / Smala vägar

Trafik inom lokala vägar kan bli
överbelastade och skapa långsam trafik

*(Låter som operationer såsom knapptryck och
uppdatering av text på vår textvy)*



I/O Thread

Vad kan vi säga om motorvägarna?

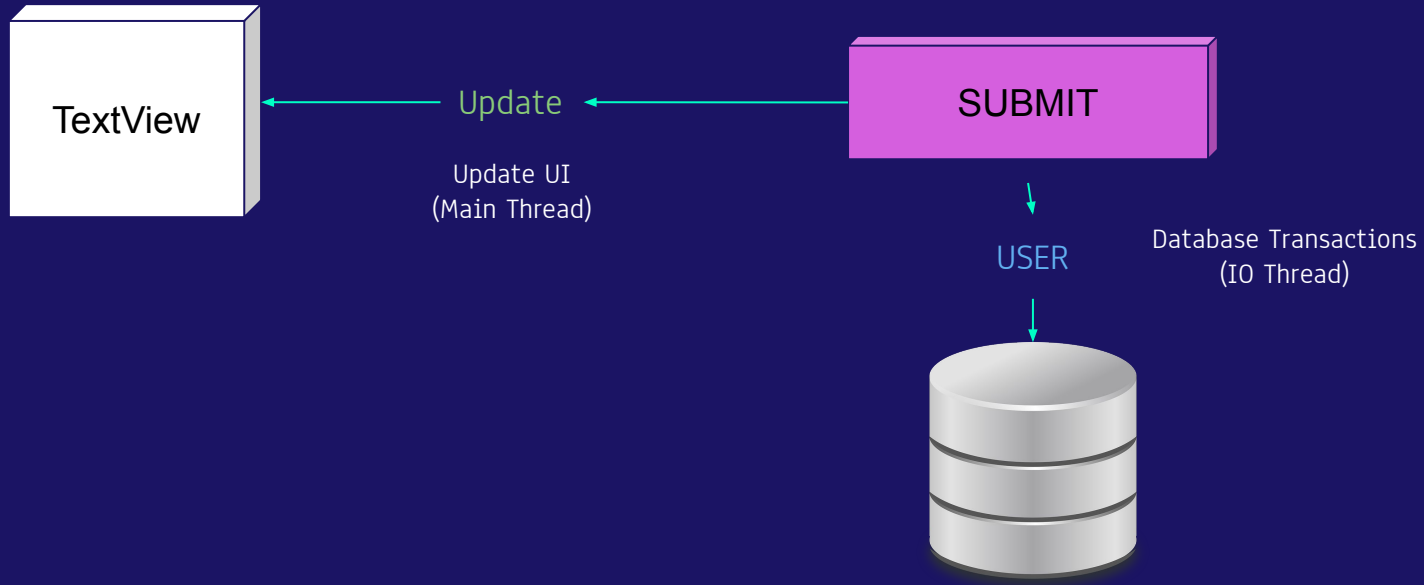
- Flertal vägar
- Mindre belastning
- Blockerar ej trafiken inom staden
- Tyngre och större trafik kan ske

För att avlasta trafiken från staden kan vi använda oss av motorvägarna för att bryta ut trafiken

(Inputs / outputs, transactions, dessa låter som något tyngre t.ex. databas transaktioner)



THREADS



Threads pt.1

- **Main Thread/UI Thread:** This is the thread responsible for handling user interface interactions and rendering. It is also sometimes called the "UI thread" because it is the only thread that is allowed to modify the user interface directly.
- **Background Thread:** This is a thread that is used for any task that takes a long time to complete and should not block the Main Thread. Examples of tasks that might be performed on a background thread include database queries, network requests, and image processing.
- **Worker Thread:** This is a type of background thread that is created and managed by the system for performing specific tasks. For example, the Android system has a specific type of worker thread called a "HandlerThread" that can be used to perform background tasks in a specific order.

Threads pt.2

- **IO Thread:** This is a thread that is optimized for performing input/output (IO) operations, such as reading or writing files or making network requests. In the context of Android development, the IO Thread is often used in conjunction with the Room database library to perform database queries.
- **Service Thread:** This is a thread that is used for running background services, which are long-running processes that run independently of the user interface. Services are often used for tasks such as playing music, downloading files, or monitoring sensor data.
- **Timer Thread:** This is a thread that is used for scheduling tasks to run at specific times or intervals. It is often used in conjunction with the Timer class in Java.



Problem

Vad är ett 'Thread'?

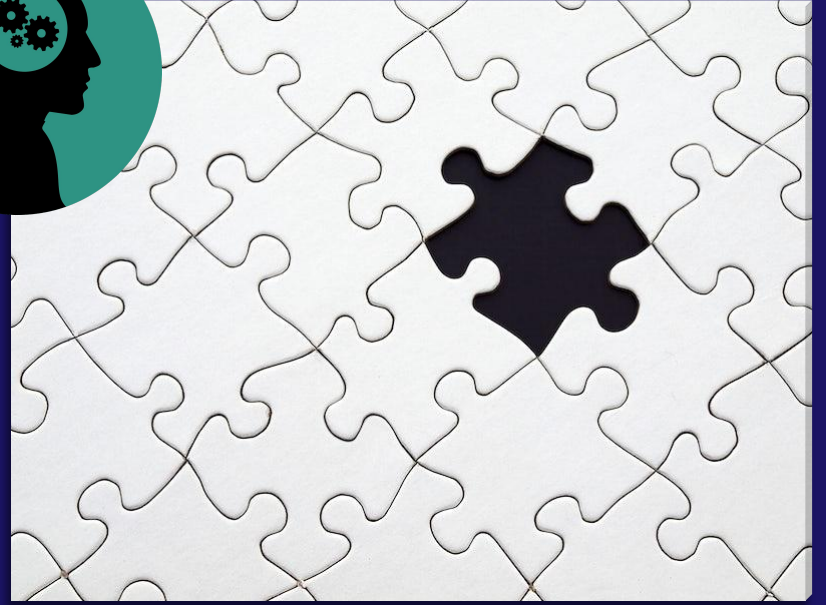


Solution

Ett "Thread" är inte mycket mer än instruktioner som är oberoende av andra "threads" (instruktioner)



Frågor?



External Database?

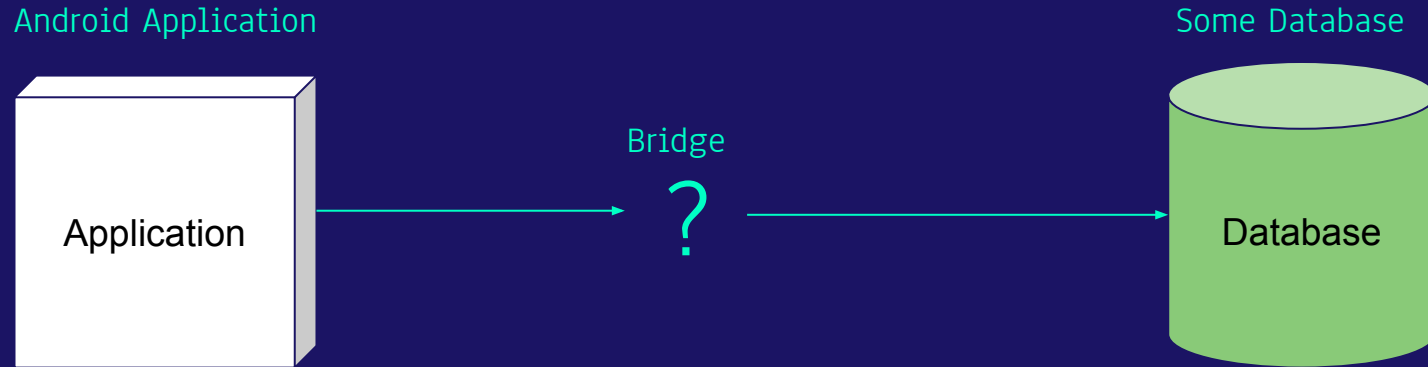


Project Structure

Vi vet redan att vi kommer arbeta med en 'lokal' databas.

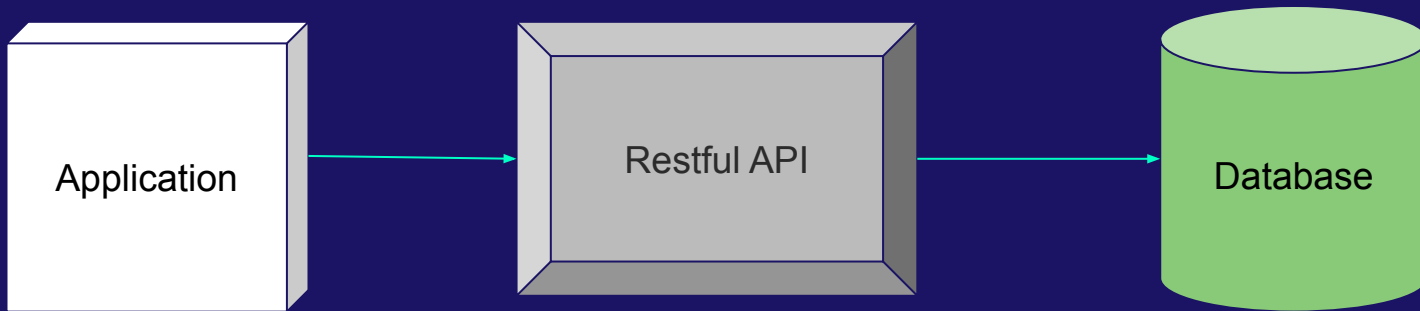
Men ibland vill man göra mer.. Därför kommer jag inkludera alternativ här lite kort!

Project Structure



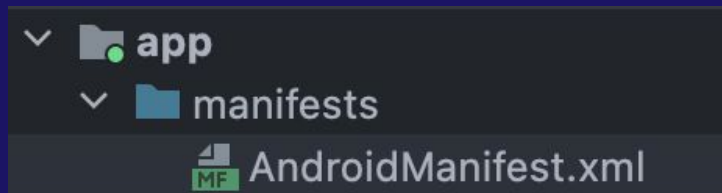
Ett väldigt vanligt problem är att man saknar en viss 'brygga'

Project Structure



Här kan man då använda sig av t.ex. Node.js eller annan 'server side' programvara som kan agera som kommunikator

Permissions



```
<uses-permission android:name="android.permission.INTERNET" />
```

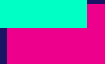
Eftersom 'internet' inte anses vara en 'farlig permission', så behövs inte en begäran under start av applikation.

Detta steg är inte nödvändigt!

Project Structure

Återigen, vi kommer inte arbeta med detta - men vill ändå berätta att detta finns som möjlighet.

Kanske om man väljer köra på detta som examensarbete!





03

ROOM & PERSISTING DATA

Entity



Vad är ett Entity?

Ett Entity är ett objekt som skall bevaras (persist) inom en databas.

User Entity



KRAV LISTA PÅ ENTITETER



Entity

Model

@Entity

Markerar klassen som ett 'Entitets objekt'

@PrimaryKey

Det finns åtminstone ett fält med primär nyckel

@Ignore / transient

Om ett fält inte markerats med dessa, så ska fältet vara publikt eller med publik get/set

Denna annotation läggs till om det finns ett fält vi inte vill lägga in i databasen.

Supported Data types

Supported Data types

- Int
- Long
- Short
- Byte
- Float
- Double
- Boolean
- String
- ByteArray
- Date
- BigDecimal
- BigInteger

No Arrays

Det finns begränsningar, här är en lista av datatyper som stöds 'out of the box'



```
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class User(
    var name: String,
    var age: Int
) {

    @PrimaryKey(autoGenerate = true)
    var id: Long? = null
}
```

Ett objekt

Märk av att vi har markerat 'id' som en **primär nyckel**.

Detta kommer då kännetecknas som en **identifikation** för
vårt objekt.

Två objekt kan ha samma värden, men de har olika idn't!

Märk av att vi har en konstruktor, men den inkluderar inte
ett id.

Vi markerar den med **autoGenerate = true** så att ROOM
sköter id åt oss!



User DAO





```
package com.example.deleteme.user

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.Query

@Dao
interface UserDao {

    @Insert
    fun insertUser(user: User)

    @Query("SELECT * FROM User")
    fun getAllUsers(): List<User>

}
```

UserDAO

DAO - Data Access Object

Är en abstraktion av ett objekt som ligger närmre databasen.

Detta så att vi gömmer fula queries från vår kod!

Här definierar vi: Queries!

@Insert = Inbyggd och färdig query för att spara data
@Query() = Bygg din egen query!

Database Class



```
@Database(entities = [User::class], version = 1)
abstract class AppDatabase : RoomDatabase() {

    abstract fun userDao(): UserDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getInstance(context: Context):
AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance =
Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "my-app-db"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Database

Ni behöver ej veta vad denna kod gör i nära detalj.

Detta kör igång ett 'Singleton pattern'
Är instansen tillgänglig? Om inte: Skapa en ny.

Synchronized säger till att bara en 'Thread' kan komma åt
detta kod block.

@Volatile säger till att variabeln INSTANCE kommer aldrig
'cache', och förändringar på denna variabel kommer vara
synlig inom andra 'threads'.

Så genom applikationens livscykel förser vi att bara EN
instans körs.

Multipla instanser vs Object med max en instans.

User Repository



```

package com.example.deleteme.user

import com.example.deleteme. AppDatabase
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class UserRepository (private val appDatabase: AppDatabase, private val
coroutineScope: CoroutineScope) {

    fun insertUser (user: User) {
        appDatabase.userDao ().insertUser (user)
    }

    fun getUsers (): List<User> {
        return appDatabase.userDao ().getAllUsers ()
    }

    fun performDatabaseOperation (dispatcher: CoroutineDispatcher,
databaseOperation: suspend () -> Unit) {
        coroutineScope.launch(dispatcher) {
            databaseOperation ()
        }
    }
}

```

UserRepository

Här definierar vi alla våra funktioner!

'insertUser' från userDao()

Jag har inkluderat coroutines här så att ni ej behöver oroa er över Threads!

Main Activity Fetch & Insert



MainActivity

```
val db = AppDatabase.getInstance(applicationContext)

userRepository = UserRepository(db, lifecycleScope)

println(applicationContext.getDatabasePath("my-app-db"))
```

onCreate()

Inom onCreate, så kan vi sätta upp vår databas, repository och förbereda lite data!

MainActivity

```
btnInsertUser.setOnClickListener {  
    userRepository.performDatabaseOperation (Dispatchers.IO) {  
        userRepository.addUser(  
            User("Benny", "123", "ADMIN")  
        )  
    }  
}
```

onClick

Här så kallar vi på repository,
performDatabaseOperation {

```
    insert (User Benny)  
}
```

```
// FETCH
btnFetchUsers.setOnClickListener {

    userRepository.performDatabaseOperation (Dispatchers.IO)
{
    val usersList = userRepository.getAllUsers ()

    println (usersList)

userRepository.performDatabaseOperation (Dispatchers.Main) {
    tvUsers.text = usersList.toString ()
    }
}
}
```



onClick

Om ni vill arbeta med både databas
transaktioner och uppdatera UI

Kör på separata 'Threads'

TypeConverter (Array Support) ADVANCED

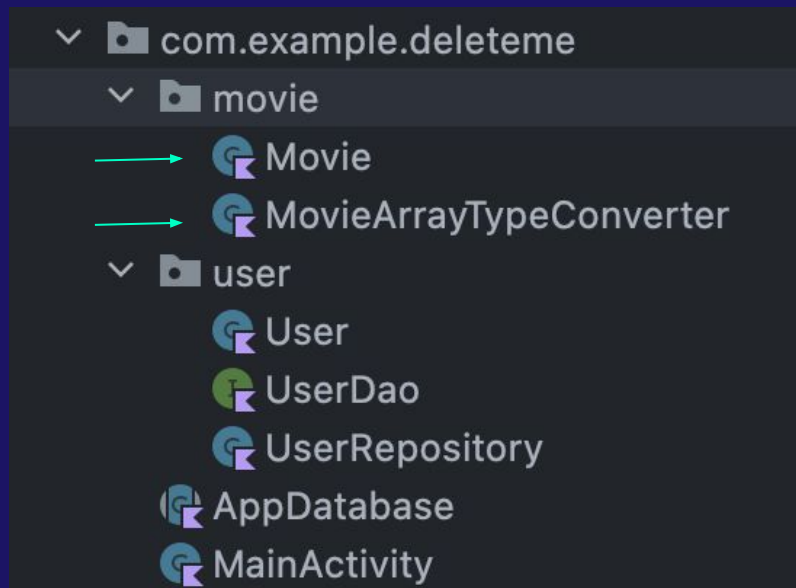




Dependency Requirement

```
implementation 'com.squareup.retrofit2:converter-gson:2.9.0' // GSON
```

Project Structure



Vi vill att användare ska kunna ha en lista av filmer med sig!

Vi behöver objektet
Och en konverterare

Movie Object

Enkel String som variabel

Viktiga här är att veta att det är ett objekt.
Vi behöver INTE markera detta som ett entitets objekt
eftersom 'Konverterare' kommer sköta detta åt oss.

▼ com.example.deleteme

▼ movie

→ Movie

```
package com.example.deleteme.movie
```

```
data class Movie(  
    val name: String,  
) {  
}
```

▼ com.example.deleteme

▼ movie

Movie

→ MovieArrayTypeConverter

```
package com.example.deleteme.movie

import androidx.room.TypeConverter
import com.google.gson.Gson

class MovieArrayTypeConverter {

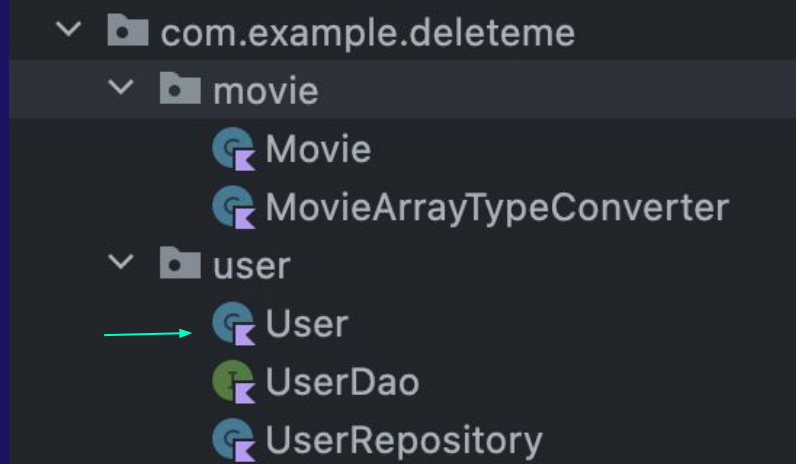
    @TypeConverter
    fun fromString(value: String): Array<Movie> {

        return Gson().fromJson(value,
Array<Movie>::class.java)
    }

    @TypeConverter
    fun fromMovieArray(array: Array<Movie>): String {
        return Gson().toJson(array)
    }
}
```

Array Converter

Byt ut <Movie> till er datatype



Lägg till array

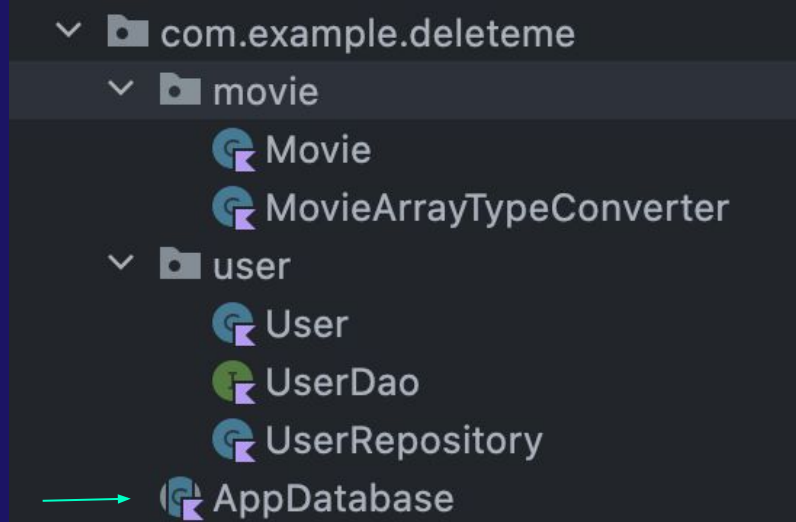
Nu accepterar vi en array med objekt!

Ni kommer få en varning när ni gör detta kring 'equals' och 'hashCode'.

Tillämpa overrides och låt den generera automatiskt.

```
@Entity
data class User(
    var name: String,
    var age: Int,

    @TypeConverters(MovieArrayTypeConverter::class)
    var movies: Array<Movie>
) {
```



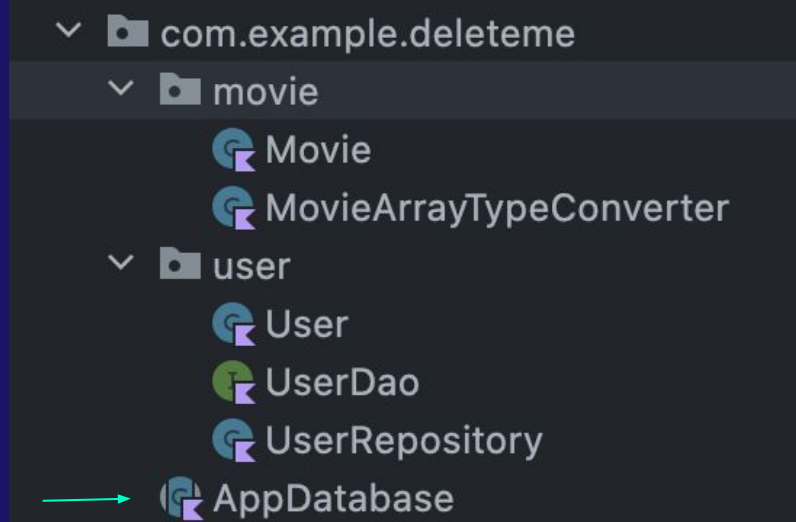
Database!

Lägg till support för din nya klass nu!
Glöm inte ändra versionen också!

Annars får ni följande error:

Looks like you've changed schema but forgot to update
the version number. You can simply fix this by increasing
the version number

```
@Database(entities = [User::class], version = 2)
@TypeConverters(MovieArrayTypeConverter::class)
abstract class AppDatabase : RoomDatabase() {
```



MIGRATION ERROR

Om ni får problem med migration, skriv så här:

Detta kommer **ta bort all tidigare data!**
(helst vill man undvika detta vid produktion)

Under debugging är detta OK!

```
fun getInstance(context: Context): AppDatabase {  
    return INSTANCE ?: synchronized(this) {  
        val instance = Room.databaseBuilder(  
            context.applicationContext,  
            AppDatabase::class.java,  
            "my-app-db"  
        )  
        .fallbackToDestructiveMigration() ←  
        .build()  
        INSTANCE = instance  
        instance  
    }  
}
```


FETCH & POST

Lägg till er array!
Testa appen!



```
val arrayOfMovies: Array<Movie> = arrayOf(  
    Movie("Star wars"),  
    Movie("Back to the Future")  
)  
  
binding.btnCreateUser.setOnClickListener {  
    userRepository.performDatabaseOperation (Dispatchers.IO) {  
        db.userDao().insertUser (User ("Benny", 15,  
arrayOfMovies))  
    }  
}
```

```
I/System.out: [User(name=Benny, age=15, movies=[Movie(name=Star wars), Movie(name=Back to the Future)]]]
```



04

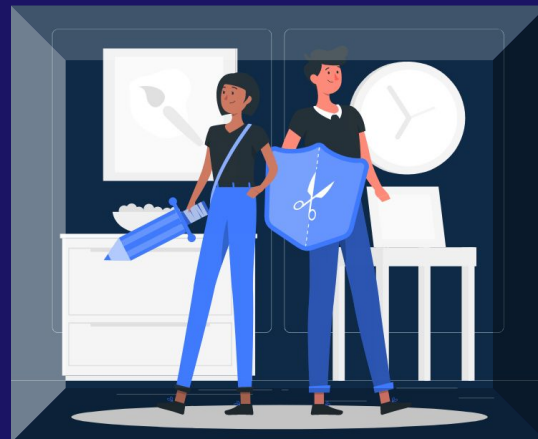
Uppgifter
&
Eget Arbete

Välkommen till första uppgiften!

Uppgifterna är till för att testa dina färdigheter och kunskaper för att både öva och repetera på det vi har arbetat med under föreläsningarna.

Dessa är **INTE** obligatoriska.
Men är starkt rekommenderat att arbeta med.

Uppgifter



MINNS DU?

```
// Vad är skillnaden på IO & Main  
Threads?
```

```
// Vad är ett singleton pattern?
```

```
// Vilka regler finns gällande Entity?
```

```
// Vilken sorts kod skrivs inom DAO?
```

```
// Vad är ett 'Repository'?
```

```
// När används TypeConverter?
```

```
1          // -Uppgift #1- //
2
3  /* INSTRUCTIONS
4
5      Skapa ett nytt projekt!
6
7      Döp projektet till: Lektion_11_uppgifter
8
9      Lägg till dependencies för ROOM + Livscykel
10
11     Skapa följande klass:
12         AppDatabase
13
14
15     Copy/paste:a koden på nästa slide
16 */
17
18 // HINT & Examples
19 hint(" Slide #9 och #10 ")
20
21
22
23
```

Uppgift #1

Kom igång enkelt med uppgift #1

```

1  package com.example.YOUR_APP_NAME
2
3  import android.content.Context
4  import androidx.room.Database
5  import androidx.room.Room
6  import androidx.room.RoomDatabase
7  import androidx.room.TypeConverters
8
9  @Database(entities = [Student::class], version = 1)
10 abstract class AppDatabase : RoomDatabase() {
11
12     abstract fun studentDao(): StudentDao
13
14     companion object {
15         @Volatile
16         private var INSTANCE: AppDatabase? = null
17
18         fun getInstance(context: Context): AppDatabase {
19             return INSTANCE ?: synchronized(this) {
20                 val instance = Room.databaseBuilder(
21                     context.applicationContext,
22                     AppDatabase::class.java,
23                     "my-app-db"
24                 )
25                     .build()
26                 INSTANCE = instance
27                 instance
28             }
29         }
30     }
31 }

```

CODE

COPY PASTE

Denna kod är finurlig på det sätt att det är 'singleton pattern'.

@Volatile pekar på 'Threads' och att data inte ska 'cache'a.

Mycket av denna kod behöver man ej kunna. Men databas namnet är viktigt 'my-app-db'

```

1          // -Uppgift #2- //
2
3      /* INSTRUCTIONS
4
5          Skapa ett nytt package:
6              "student"
7
8          Inom detta package skapar du:
9              data class      Student
10             interface StudentDao
11             class           StudentRepository
12
13          Ge er 'Student' några variabler inom
14          konstruktorn.
15
16          Exempelvist: Name, Age, Grade
17
18      */
19      // HINT & Examples
20      hint("
21          Packages hjälper oss att strukturera appen så att
22          allt inte ligger inom en och samma fil!
23          ")

```

Uppgift #2

Uppsättning här är viktigt.
Nästa uppgift är att definiera ett entitet!

```

1          // -Uppgift #3- //
2
3      /* INSTRUCTIONS
4
5         Gör nu om Student till ett Entitets objekt!
6
7         Annotationer som måste användas:
8
9         @Entity
10        @PrimaryKey
11
12    */
13
14    // HINT & Examples
15    hint("
16    Slide #38
17    Klass nivå och på ID nivå.
18
19    Försök att exkludera 'id' inom konstruktorn.
20
21    Inom primaryKey() ← sätt autoGenerate = true
22    Om ni vill att id generering skall ske per
23    automatik inom ROOM.
24    ")

```

Uppgift #3

Entiteter, objekt som skall sättas in via ORM, in mot vår databas.


```

1          // -Uppgift #4- //
2
3      /* INSTRUCTIONS
4
5          Inom StudentDao interface
6          Markera på klassnivå: @Dao
7
8          Nu kan du skapa queries med annotationer!
9          Prova med olika annotationer:
10         @Insert
11         @Delete
12         @Update
13         @Query
14
15         Bara genom att hålla över dessa så kommer ni
16         få förslag på hur man kan skriva sin kod!
17
18     */
19
20     // HINT & Examples
21     hint("
22     Slide #40
23     ")

```

Uppgift #4

Med annotationer, styr vi hur vi vill
manipulera data.
Detta är ett jätteviktigt steg!

```

1          // -Uppgift #5- //
2
3      /* INSTRUCTIONS
4
5          Navigera nu mot Student Repository
6
7          Copy / paste följande kod:
8
9      fun performDatabaseOperation(
10     dispatcher: CoroutineDispatcher,
11     databaseOperation: suspend () -> Unit) {
12
13         coroutineScope.launch(dispatcher) {
14             databaseOperation()
15         }
16     }
17
18     // HINT & Examples
19     hint("
20     Argumenten är viktiga för att den skall kunna
21     hålla reda på vilket 'Thread' dessa operationer
22     skall köras på! Detta utökar prestandan!
23     ")

```

Uppgift #5

Prestanda inom apputveckling är väldigt viktigt. Här utför vi förberedelser inför Activity Main!

Mer kod finns på nästa sida

```

1  package com.example.deleteme.user
2
3  import com.example.deleteme.AppDatabase
4  import kotlinx.coroutines.CoroutineDispatcher
5  import kotlinx.coroutines.CoroutineScope
6  import kotlinx.coroutines.Dispatchers
7  import kotlinx.coroutines.launch
8
9  class UserRepository(private val appDatabase: AppDatabase,
10 private val coroutineScope: CoroutineScope) {
11
12     fun insertUser(user: User) {
13         appDatabase.userDao().insertUser(user)
14     }
15
16     fun getUsers(): List<User> {
17         return appDatabase.userDao().getAllUsers()
18     }
19
20     fun performDatabaseOperation(dispatcher:
21 CoroutineDispatcher, databaseOperation: suspend () -> Unit)
22 {
23     coroutineScope.launch(dispatcher) {
24         databaseOperation()
25     }
26 }
27

```

CODE

Märk av att vi har 'insertUser' och även 'getUser' här.

Om ni vill ha med Delete, inkludera detta här med!

```

1          // -Uppgift #6- //
2
3      /* INSTRUCTIONS
4
5          Navigera in till Main onCreate()
6
7          Copy / paste:a följande kod:
8
9          val db = AppDatabase.getInstance(applicationContext)
10         userRepository = UserRepository(db, lifecycleScope)
11
12     */
13
14     // HINT & Examples
15     hint("
16         Detta steg är intressant för att vi
17         passar nu in 'lifeCyclescope' så att den är
18         Med på om livscykeln störs eller inte.
19
20         App context är också med!
21     ")
22
23

```

Uppgift #6

```
1 // -Uppgift #6.5- //
```

```
2
```

```
3 /* INSTRUCTIONS
```

```
4
```

```
5 binding.btnFetchUser.setOnClickListener {
```

```
6     userRepository.performDatabaseOperation(Dispatchers.Main)
```

```
7     {
```

```
8         // Update Main
```

```
9     }
```

```
10     userRepository.performDatabaseOperation(Dispatchers.IO) {
```

```
11         // Update IO
```

```
12     }
```

```
13 }
```

```
14 */
```

```
15
```

```
16 // HINT & Examples
```

```
17 hint("
```

```
18     Main är till för UI förändringar
```

```
19     IO är till för tyngre transaktioner
```

```
20 ")
```

```
21
```

```
22
```

```
23
```

Uppgift #6.5

```
1          // -Uppgift #7- //
2          TOUGH NUT
3      /* INSTRUCTIONS
4
5          Försök nu pusha in en array med böcker
6          som en student kan ha!
7
8      */
9      // HINT & Examples
10     hint("
11     Guide: Slide #49
12     ")
13
14
15
16
17
18
19
20
21
22
23
```

Uppgift #7

Good luck CHAMP!

THANKS!

Do you have any questions?
kristoffer.johansson@sti.se

sti.learning.nu/

CREDITS: This presentation template was created by
Slidesgo, including icons by Flaticon, and
infographics & images by Freepik.

You can also contact me VIA Teams (quicker response)
Du kan också kontakta mig VIA Teams (Snabbare svar)