DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Entwicklung mobiler Applikationen

WS2022/2023

Jan Brodhaecker | November 2022

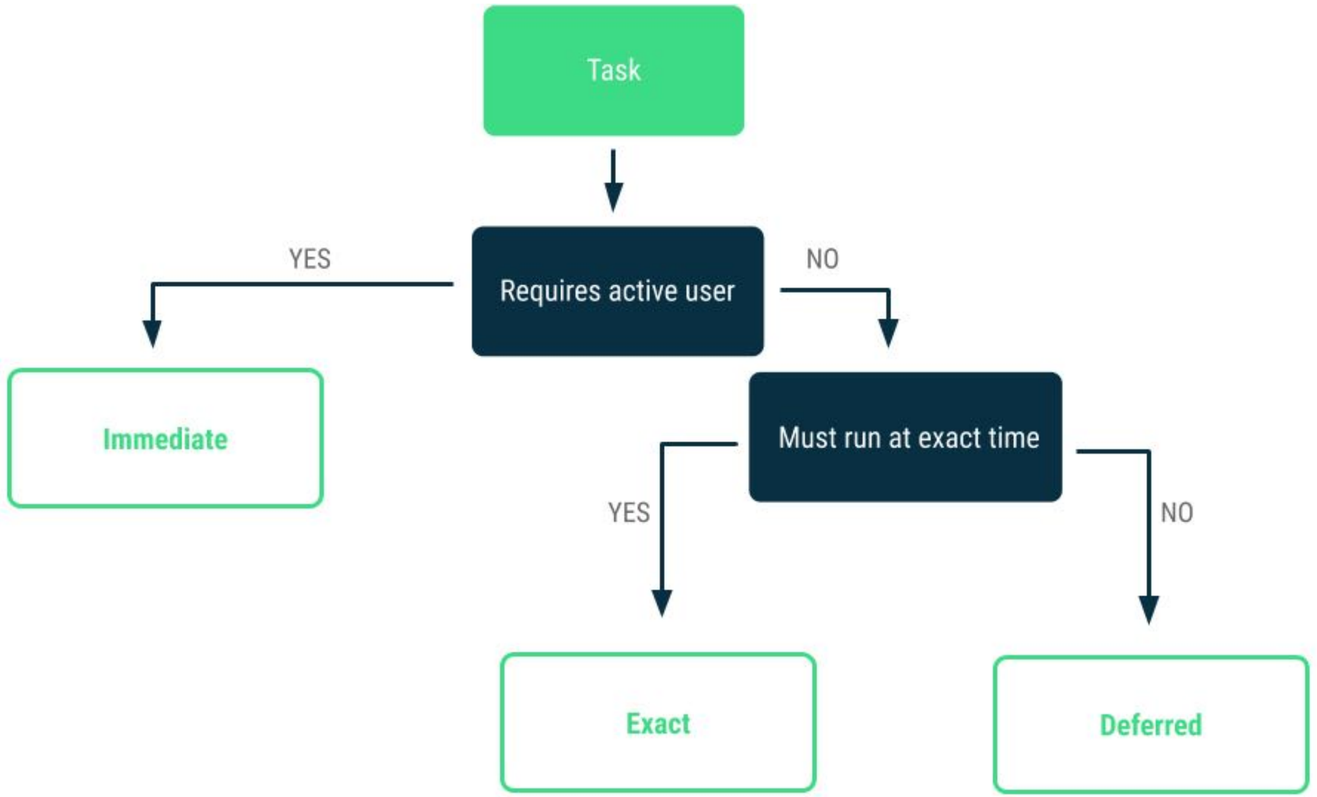
Agenda

- Background Tasks
- Persistenz
- Content-Provider

Background Tasks

- Aufgaben die mehr als ein „paar“ Millisekunden benötigen, sollten in einen Background Tasks ausgelagert werden
- Definition
  - keine Activity, die sichtbar ist
  - kein sogenannter „Foreground“ Service wurde gestartet
- Kategorien
  - immediate (unmittelbar)
  - deferred (verzögert)
  - exact (exakt)

Background Tasks



https://developer.android.com/guide/background

Background Tasks

- Immediate
  - bspw. Web-Request
  - mit Hilfe von Threads oder WorkManager (falls App im Hintergrund)
  - Concurrency Libraries (Guava, RxJava, Kotlin Coroutines)
- Deferred
  - muss nicht zu einem bestimmten Zeitpunkt gestartet werden, wird vom System gesteuert
  - mit Hilfe vom WorkManager
- Exact
  - muss zu einem bestimmten Zeitpunkt ausgeführt werden
  - mit Hilfe von AlarmManager

Background Tasks

- Broadcasts und Broadcast Receiver
  - Nachrichten aus dem System oder von anderen Applikationen empfangen (ähnlich Publish-Subscribe Pattern)
  - kann als Messaging-System verwendet werden um abseits vom Benutzer-Flow zwischen Applikationen zu kommunizieren

Background Tasks

- Broadcasts und Broadcast Receiver
  - Nachrichten aus dem System oder von anderen Applikationen empfangen (ähnlich Publish-Subscribe Pattern)
  - kann als Messaging-System verwendet werden um abseits vom Benutzer-Flow zwischen Applikationen zu kommunizieren

Background Tasks

Broadcast Receiver

Definition / Registrierung im Manifest (bspw. Geofences wenn System startet)

```
...
    <receiver android:name=".BootCompletedReceiver"
        android:enabled="true"
        android:exported="true" >

        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
      </receiver>
...
```

Beispiel Implementierung

```
class BootCompletedReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        Log.d(BootCompletedReceiver::class.simpleName, "onReceive")
    }
}
```

Background Tasks

Broadcast Receiver

- Registrierung am Context möglich
  - so lange gültig, wie der Context selbst
  - Lifecycle Events sind zu beachten (bspw. de-registrieren bei `onPause`)

```
1 val airplaneModeReceiver = AirplaneModeReceiver()
2 val intentFilter = IntentFilter().apply {
3   addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED)
4 }
5 registerReceiver(airplaneModeReceiver, intentFilter)
```

Background Tasks

Broadcast Receiver

- definition von System Broadcasts
  - bspw. USB-Device attached
  - komplette Übersicht: https://developer.android.com/about/versions/11/reference/broadcast-intents-30
- eigene Definition inkl. senden

```
1 val customIntent = Intent().apply {
2   action = "com.dhbw.custom.intent"
3   putExtra("data", "very sensitive information")
4 }
5 sendBroadcast(customIntent)
```

Background Tasks

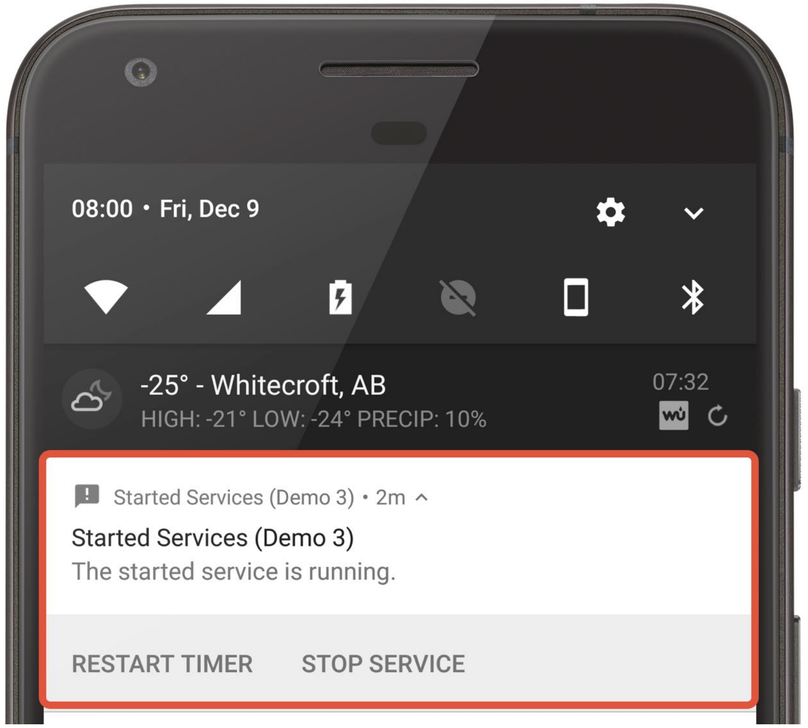
Services

- länger andauernde Operationen im Hintergrund
- kein (direktes) User-Interface
- Kategorien
  - Foreground
  - Background
  - Bound

Background Tasks

Services - Foreground

- zeigen eine Notification in der Status-Leiste an
- weisen den Nutzer auf Background-Aktivität hin
- Beispiele
  - Navigation, die den Wechsel der Richtung ansagt
  - Musik-Player



https://medium.com/@jayd1992/foreground-services-in-android-e131a863a33d

Background Tasks

Services - Background

- Nutzer bemerkt nicht, dass Service im Hintergrund läuft
- keine direkte Interaktion möglich (asynchron)
- Limitierungen beachten https://developer.android.com/about/versions/oreo/background
- Beispiele
  - Downloads (bspw. in einem Browser)
  - Optimierung lokaler Caches

Background Tasks

Services - Bound

- Server im Server-Client Modell
- verschiedene App-Komponenten können sich an Service „binden“ und Requests empfangen und versenden
- ermöglicht synchrone Aufrufe
- stellt Funktionalität bereit, die wiederverwendet werden kann
- für komplexe Funktionalität zwischen Aktivität und Service
- Unterschied zum Foreground-Service: kann vom System beendet werden

Background Tasks

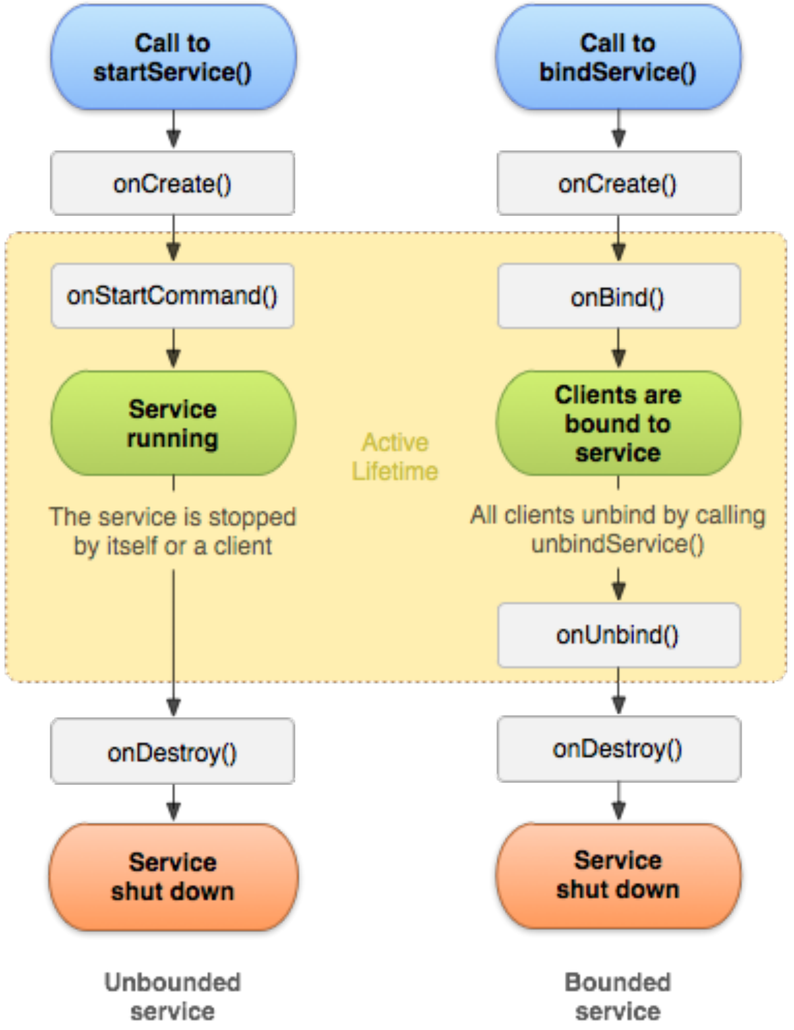
Services - Bound

A started service

The service is created when another component calls `startService()`. The service then runs indefinitely and must stop itself by calling `stopSelf()`. Another component can also stop the service by calling `stopService()`. When the service is stopped, the system destroys it.

A bound service

The service is created when another component (a client) calls `bindService()`. The client then communicates with the service through an `IBinder` interface. The client can close the connection by calling `unbindService()`. Multiple clients can bind to the same service and when all of them unbind, the system destroys the service. The service does *not* need to stop itself.



https://developer.android.com/guide/components/services

Persistenz
Übersicht

	Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
App-specific files	Files meant for your app's use only	From internal storage, `getFilesDir()` or `getCacheDir()` From external storage, `getExternalFilesDir()` or `getExternalCacheDir()`	Never needed for internal storage Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher	No	Yes
Media	Shareable media files (images, audio files, videos)	`MediaStore` API	`READ_EXTERNAL_STORAGE` when accessing other apps' files on Android 11 (API level 30) or higher `READ_EXTERNAL_STORAGE` or `WRITE_EXTERNAL_STORAGE` when accessing other apps' files on Android 10 (API level 29) Permissions are required for **all** files on Android 9 (API level 28) or lower	Yes, though the other app needs the `READ_EXTERNAL_STORAGE` permission	No
Documents and other files	Other types of shareable content, including downloaded files	Storage Access Framework	None	Yes, through the system file picker	No
App preferences	Key-value pairs	Jetpack Preferences library	None	No	Yes
Database	Structured data	Room persistence library	None	No	Yes

Persistenz
Übersicht

- Wann verwende ich was?
  - Wie viel Speicherplatz wird benötigt?
    - interner Speicher ist limitiert
  - Wie verlässlich muss der Zugriff sein?
    - Bei App-Start? Externer Speicher (SD-Karte, Remote-System, …) muss nicht immer verfügbar sein
  - Welche Art von Dateien muss gespeichert werden
    - Mutlimedia-Dateien können auch für andere Apps interessant sein
  - Sollten die Daten nur für die eigene App verfügbar sein?
    - interner Speicher kann nur von der eigenen App verwendet werden

Persistenz

File-API

app-spezifisch

```
val file = File(context.filesDir, filename)
```

externer Speicher (Permissions beachten!)

```
val appSpecificExternalDir = File(context.getExternalFilesDir(), filename)
```

temporäre Dateien sollten im Cache abgelegt werden!

Persistenz

File-API

alle Dateien im Verzeichnis

```
var files: Array<String> = context.fileList()
```

Verzeichnis erzeugen

```
context.getDir(dirName, Context.MODE_PRIVATE)
```

Persistenz

File-API

Datei schreiben

```
val filename = "myfile"
val fileContents = "Hello world!"
context.openFileOutput(filename, Context.MODE_PRIVATE).use {
        it.write(fileContents.toByteArray())
}
```

Datei lesen

```
context.openFileInput(filename).bufferedReader().useLines { lines ->
    lines.fold("") { some, text ->
        "$some\n$text"
    }
}
```

Persistenz

Multimedia-Dateien

Zugriff auf Multimedia-Dateien

```
fun getAppSpecificAlbumStorageDir(context: Context, albumName: String): File? {
    // Get the pictures directory that's inside the app-specific directory on
    // external storage.
    val file = File(context.getExternalFilesDir(
            Environment.DIRECTORY_PICTURES), albumName)
    if (!file?.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created")
    }
    return file
}
```

Persistenz

Multimedia-Dateien

Optimierter Zugriff

```
val projection = arrayOf(media-database-columns-to-retrieve)
val selection = sql-where-clause-with-placeholder-variables
val selectionArgs = values-of-placeholder-variables
val sortOrder = sql-order-by-clause

applicationContext.contentResolver.query(
    MediaStore.media-type.Media.EXTERNAL_CONTENT_URI,
    projection,
    selection,
    selectionArgs,
    sortOrder
)?.use { cursor ->
    while (cursor.moveToNext()) {
        // Use an ID column from the projection to get
        // a URI representing the media item itself.
    }
}
```


https://developer.android.com/training/data-storage/shared/media

Persistenz

Multimedia-Dateien

- **Images**, including photographs and screenshots, which are stored in the `DCIM/` and `Pictures/` directories. The system adds these files to the `MediaStore.Images` table.
- **Videos**, which are stored in the `DCIM/`, `Movies/`, and `Pictures/` directories. The system adds these files to the `MediaStore.Video` table.
- **Audio files**, which are stored in the `Alarms/`, `Audiobooks/`, `Music/`, `Notifications/`, `Podcasts/`, and `Ringtones/` directories, as well as audio playlists that are in the `Music/` or `Movies/` directories. The system adds these files to the `MediaStore.Audio` table.
- **Downloaded files**, which are stored in the `Download/` directory. On devices that run Android 10 (API level 29) and higher, these files are stored in the `MediaStore.Downloads` table. *This table isn't available on Android 9 (API level 28) and lower.*

https://developer.android.com/training/data-storage/shared/media

Persistenz

Shared-Preferences

- für eine *„small collection of key-values“* können *Shared-Preferences* verwendet werden
- maximal zwei Spalten
- `getSharedPreferences()`
  - Preferences für einen Schlüssel, mehrere pro Activity
- `getPreferences()`
  - Preferences ohne Schlüssel

https://developer.android.com/training/data-storage/shared-preferences

Persistenz

Shared-Preferences

```
val sharedPref = activity?.getSharedPreferences(
        getString(R.string.preference_file_key), Context.MODE_PRIVATE)
```

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE)
```


https://developer.android.com/training/data-storage/shared-preferences

Persistenz

Shared-Preferences

Schreiben (apply -> asynchron, commit -> synchron; Threading beachten!)

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
with (sharedPref.edit()) {
    putInt(getString(R.string.saved_high_score_key), newHighScore)
    apply()
}
```

Lesen

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
val defaultValue = resources.getInteger(R.integer.saved_high_score_default_key)
val highScore = sharedPref.getInt(getString(R.string.saved_high_score_key), defaultValue)
```


https://developer.android.com/training/data-storage/shared-preferences

Persistenz
Room Datenbank

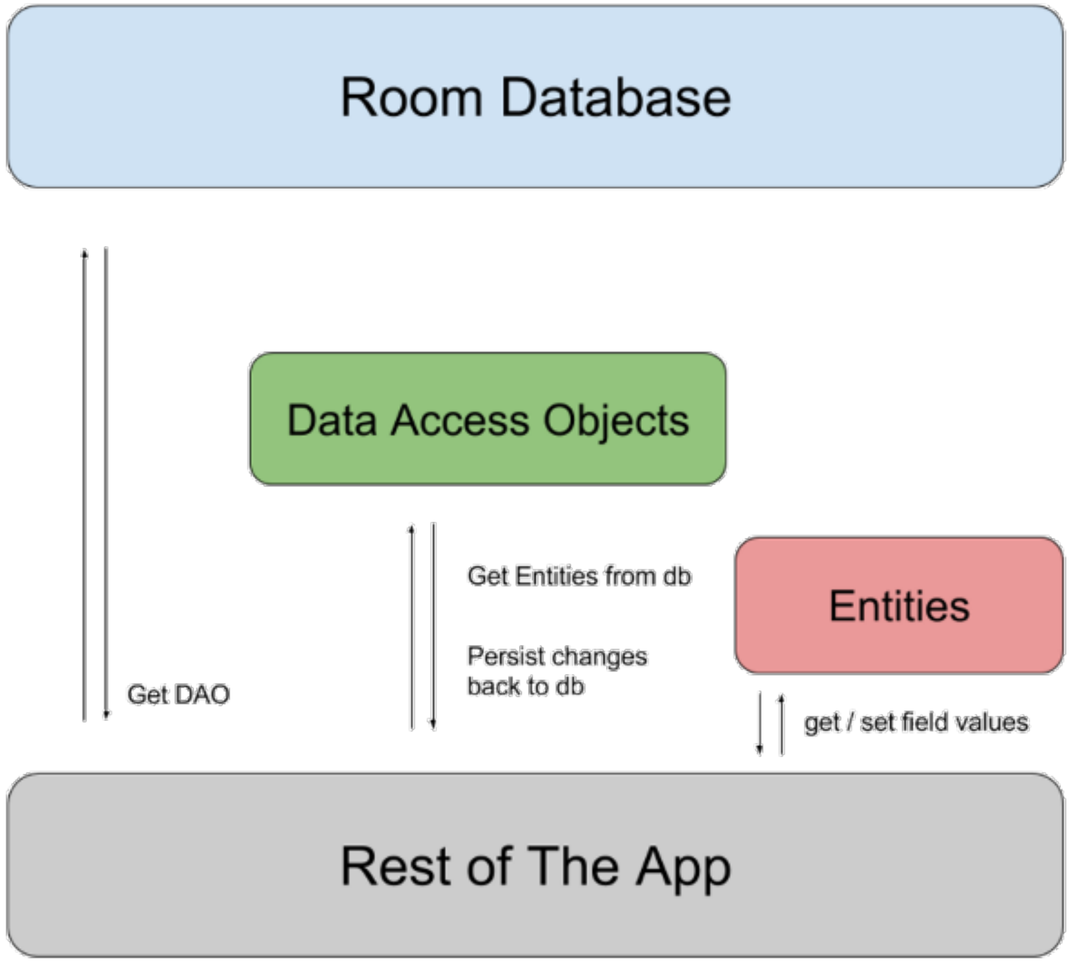
- Room bietet Abstraktionslayer ggü. SQLite Datenbanken
  - Teil von Android Jetpack
  - bevorzugt ggü. SQLite
- hilft dabei größere Datenmengen strukturiert abzulegen
- u.a Support für Offline-Funktionalität: ermöglicht remote Synchronisierung
- muss per Gradle aktiviert werden

https://developer.android.com/training/data-storage/shared-preferences

Persistenz

Room Datenbank

- Database
- Entity
- DAO (Data Access Object)



https://developer.android.com/training/data-storage/room

Persistenz
Room Datenbank

- Entity
  - repräsentiert Objekt innerhalb der Tabelle (Schema)
- Entity - Instanz
  - repräsentiert eine Zeile der Tabelle

```
@Entity
data class User(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```


https://developer.android.com/training/data-storage/room

Persistenz
Room Datenbank

- DAO - beinhaltet Methoden um auf die Datenbank zuzugreifen

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```


https://developer.android.com/training/data-storage/room

Persistenz

Room Datenbank

Datenbank definieren

```
@Database(entities = arrayOf(User::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

Datenbank instanziieren

```
val db = Room.databaseBuilder(
            applicationContext,
            AppDatabase::class.java, "database-name"
        ).build()
```


https://developer.android.com/training/data-storage/room

Persistenz
Room Datenbank

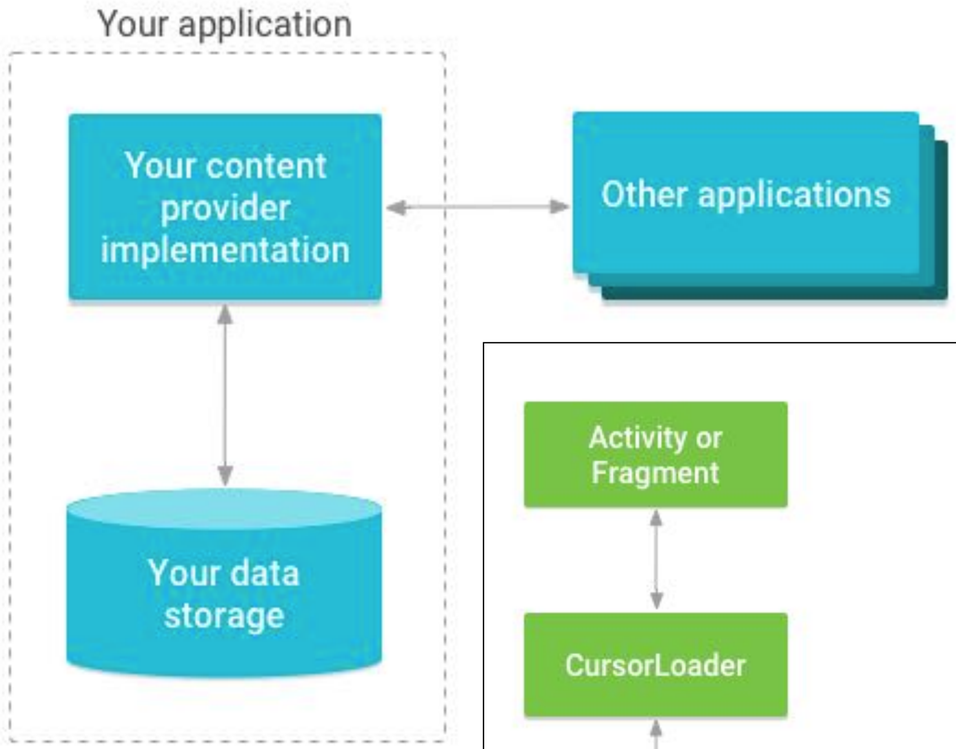
- gute Integration in andere Android-Komponenten (bspw. LiveView)
- kaum Boilerplate Code
- compile-time check der Queries
- erlaubt Schema-Änderungen der Datenbank

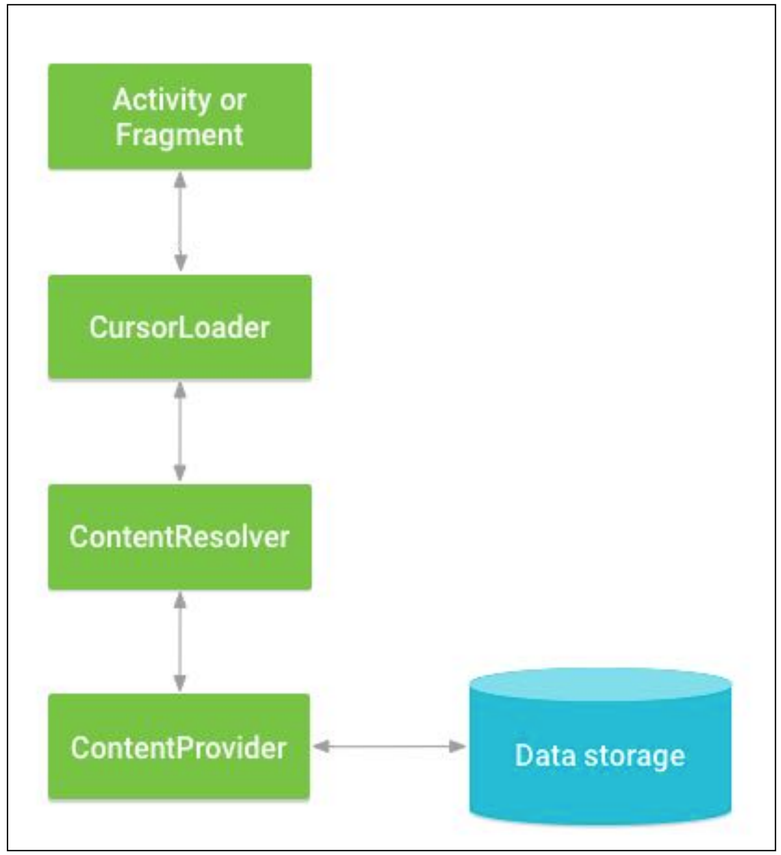
https://medium.com/mindorks/using-room-database-android-jetpack-675a89a0e942

Persistenz

Content-Provider

- erlauben Bereitstellung von komplexen Daten für andere Applikationen
- erlauben sicheren Zugriff und die Modifikation von Daten
- Zugriff in Form einer relationaler Datenbank
- Beispiele: Kontakte, Kalender …





Persistenz

Content-Provider

```
// Queries the user dictionary and returns results
cursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    projection,                         // The columns to return for each row
    selectionClause,                    // Selection criteria
    selectionArgs,                      // Selection criteria
    sortOrder);                         // The sort order for the returned rows
```

query() argument	SELECT keyword/parameter	Notes
`Uri`	`FROM table_name`	`Uri` maps to the table in the provider named *table_name*.
`projection`	`col,col,col,...`	`projection` is an array of columns that should be included for each row retrieved.
`selection`	`WHERE col = value`	`selection` specifies the criteria for selecting rows.
`selectionArgs`	(No exact equivalent. Selection arguments replace ? placeholders in the selection clause.)	
`sortOrder`	`ORDER BY col,col, ...`	`sortOrder` specifies the order in which rows appear in the returned `Cursor`.

https://developer.android.com/guide/topics/providers/content-provider-basics