Módulo 4:
Persistencia de Datos: Base de datos y Room 01

<title> Objetivos del día </title>

- Recall Día Anterior.
  - Terminar actividad SharedPreferences.
  - Resolución de Dudas..
- Base de datos y Room.
  - Entidades
  - Dao
  - Base de datos
- Utilizar Room en una aplicación.

Repaso Ejercicios: Shared Preferences

Review del ejercicio y Dudas

Luego de este ejercicio, debemos tener más o menos claro cómo:

- Crear un archivo de SharedPreferences
- Guardar Datos
- Leer Datos
- Borrar Datos

¿Qué sucede si giras la pantalla?

Persistencia en Base de datos SQlite

Objetivos

- Entender que es un ORM y cómo interactúa con la base de datos SQlite.
- Conocer Room y en que nos beneficiara.
- Utilizar Room en una app como solución de persistencia de datos, entidades, dato y cliente bbdd.

Persistencia en Base de Datos SQlite

Generalmente para esta tarea se utiliza alguna capa de abstracción por sobre SQlite. Esto se realiza principalmente para aislar posibles problemas con la utilización de este elemento, ya que si lo usamos de forma nativa, no tendremos reporte de errores sobre alguna consulta hasta que se esté ejecutando, es decir los ya conocidos Runtime Error.

A lo largo de la historia del desarrollo Android han aparecido varias alternativas como capas de abstracción, muchos ORM han tenido popularidad y han sido de mucho uso. Por mencionar algunos Realm y GreenDAO facilitaban mucho el desarrollo en de este tipo de características.

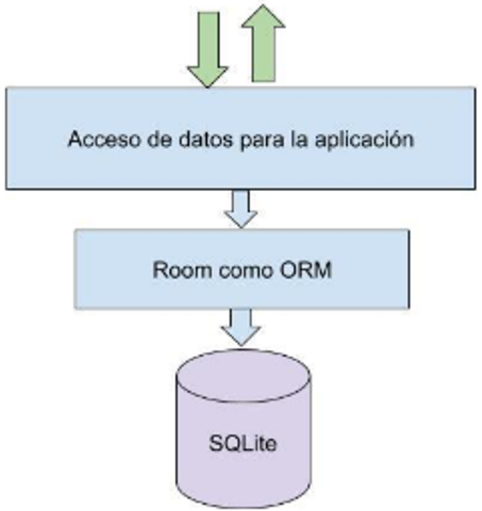
ORM


01
Object Relational Mapper.
02
Código boilerplate en la BBDD y objetos del modelo.
03
Traducción de objetos a tablas y de tablas a objetos.
04
Optimización de código.

Room como alternativa ORM

ORM (Object Relational Mapper)

- Room es recomendado por Google como alternativa de ORM.
- Permite generar el código intermedio para mapear tablas y objetos.



Room

Room proporciona una capa de abstracción por sobre SQlite lo que permite acceder a la base de datos de forma más fácil y sin tantas complicaciones.

Desde su lanzamiento, Google está abogando por el uso de Room en los proyectos, sin embargo, también es posible utilizar SQlite directo u otras alternativas como las mencionadas anteriormente.

Durante este curso vamos a revisar y utilizar Room como alternativa de persistencia de datos estructurados.

Una de las principales ventajas de Room es su compatibilidad con Live Data.

Principales componentes de Room

- Database
- Entity
- DAO

Entity

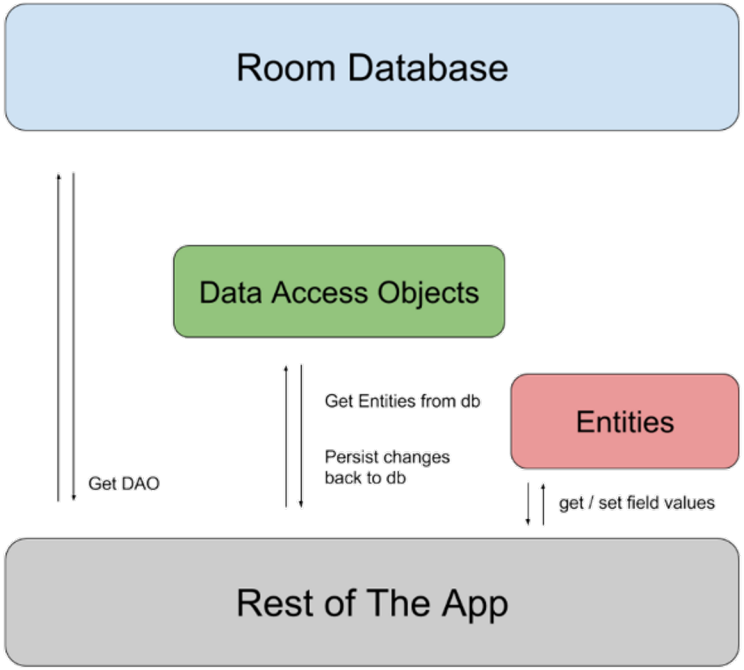
Mapea directamente a las tablas de la base de datos.

Database

Contenedor que actúa como único punto de acceso a la base de datos.

DAO
(Data Access Object)

Objeto que permite interactuar con la base de datos.



Añadir Room a un proyecto

Puedes añadir Room a un proyecto agregando esto en tu build.gradle

No olvidar añadir el plugin de kapt para manejar las anotaciones de Room.

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

```
id 'kotlin-kapt'
```

```
dependencies {
    def room_version = "2.3.0"

    implementation "androidx.room:room-runtime:$room_version"
    kapt "androidx.room:room-compiler:$room_version"

    // optional - Kotlin Extensions and Coroutines support for Room
    implementation "androidx.room:room-ktx:$room_version"

    // optional - Test helpers
    testImplementation "androidx.room:room-testing:$room_version"
}
```

Entity

Una entidad simplemente representa una tabla al interior de la base de datos.

Regularmente utilizamos nuestros POJOS (Data class) y le añadiremos la anotación @Entity

- **@Entity**: Indica a room que esta clase es un entity.
- **tableName**: Indica el nombre de la tabla que almacena este entity.
- **@ColumnInfo**: Dice que esto es una columna, con name “id”, de la tabla
- **@PrimaryKey**: Indica que es una clave primaria, autoGenerate = true: Indica que se maneje de forma automática este atributo.
- **@Ignore**: Nos permite anotar campos que no queremos que se almacenen en la base de datos.

```
@Entity(tableName = "task_table")
data class
Task(@PrimaryKey(autoGenerate = true)
@NonNull val idTask: Int,
        val task:String,
        val creationDate:String,
        val finishDate:String
        )
```

Dao

Contendrá los métodos utilizados para acceder a la base de datos. Documentación.

Es una interface la cual luego es utilizada por Room para construir su instancia. Utiliza **@anotaciones**

- **@Dao**: Indica que esto es un DAO, Room implementará todos sus métodos
- **@Query**: Nos permite ejecutar queries SQL con chequeo de errores en Compile Time.
- **@Insert**: Indica que este método hará un insert a la base de datos.
- **@Update:** Genera métodos para actualizar objetos.
- **@Delete** Nos permite eliminar objetos o colecciones completas de la base de datos.

```
@Dao
interface TaskDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertTask(task: Task)

    @Insert
    fun insertMultipleTask(list: List<Task>)

    @Update
    fun updateTask(task: Task)

    @Delete
    fun deleteOneTask(task: Task)

    @Query("SELECT * FROM task_table ORDER BY idTask
ASC")
    fun getAllTask(): LiveData<List<Task>>


}
```

Database

Esta contendrá un holder para la base de datos, y servirá como el principal acceso a las capas de persistencia de datos. La podemos reconocer porque siempre tendrá la anotación **@Database**, también debe cumplir las siguientes condiciones:

- Debe ser una clase abstracta y heredar de RoomDatabase
- Debe incluir una lista de las entidades asociadas con la base de datos.
- Debe contener un método abstracto que no tiene argumentos y que retorne los **@dao** que hayamos creado.

Database

La clase que contendrá nuestra base de datos sera asi. ->

- Este componente une las entities con los Daos en Room.
- Indica a Room que es una database.
- Entregamos un arreglo de entities. Además, le decimos que versión de base de datos se está utilizando.
- Los métodos definidos son del tipo abstract, nuestra clase hereda de RoomDatabase.

```
@Database(entities = [Task::class], version = 1, exportSchema
= false)
abstract class TaskDataBase : RoomDatabase(){

    abstract fun getTaskDao(): TaskDao

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

        fun getDatabase(context: Context): TaskDataBase {
            val tempInstance =
                INSTANCE
            if (tempInstance != null) {
                return  tempInstance
            }
            synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    TaskDataBase::class.java,
                    "task_database")
                    .build()
                INSTANCE = instance
                return instance
            }
        }
    }
}
```

Repaso consultas SQL

```
@Query("SELECT * FROM user")

@Query("SELECT * FROM user WHERE uid IN
(:userIds)")


@Query("SELECT * FROM user WHERE first_name
LIKE :first AND " +"last_name LIKE :last
LIMIT 1")
```

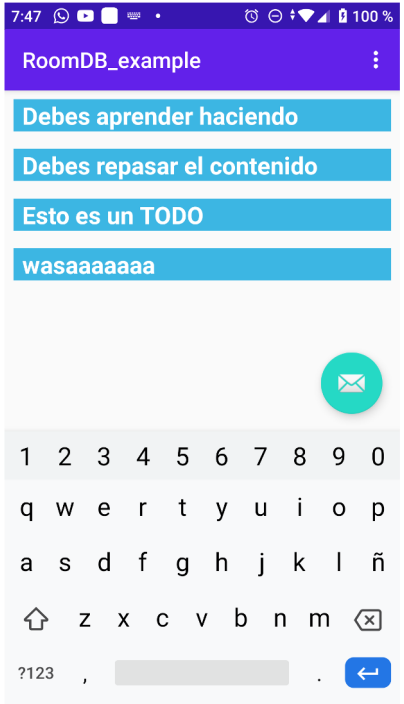
Añadir Room a un proyecto

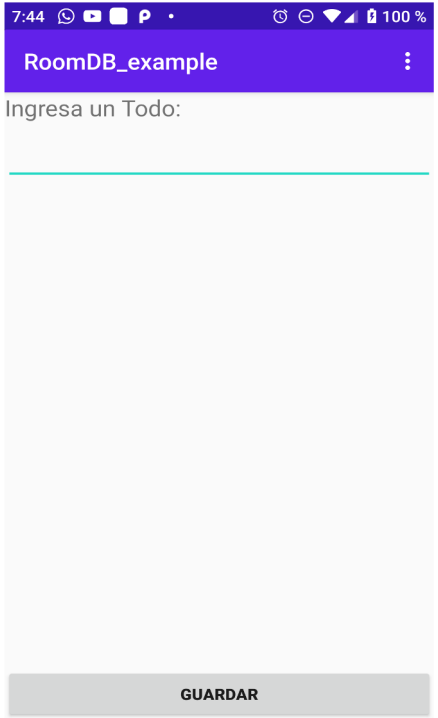
Ejercicio: Implementar Room + RV+ ¿? (To-do App)

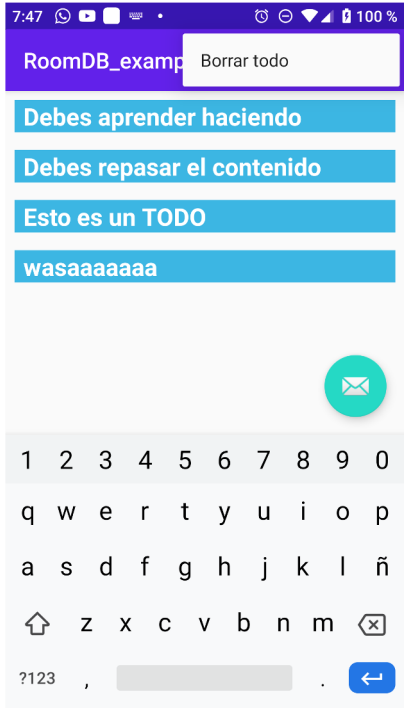
Vamos a crear una app e implementar Room database.

Además consolidar algunos conocimientos por ejemplo:

- RecyclerView
- Consultas SQL
- ¿ MVP o MVVM ?
- Callbacks
- Fragmentos



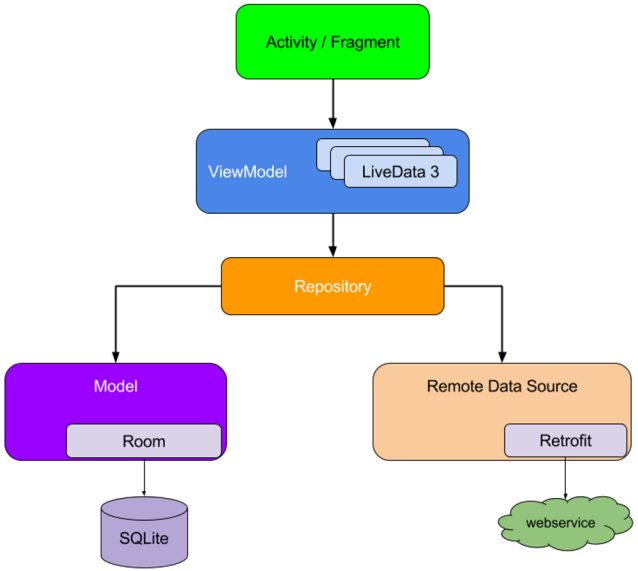




Que tipo de arquitectura vamos a construir.

Esta imagen representa el patrón de arquitectura recomendada por google.

En esta ocasión vamos a construir un patrón similar pero utilizando MVVM y solo tendremos datos locales.



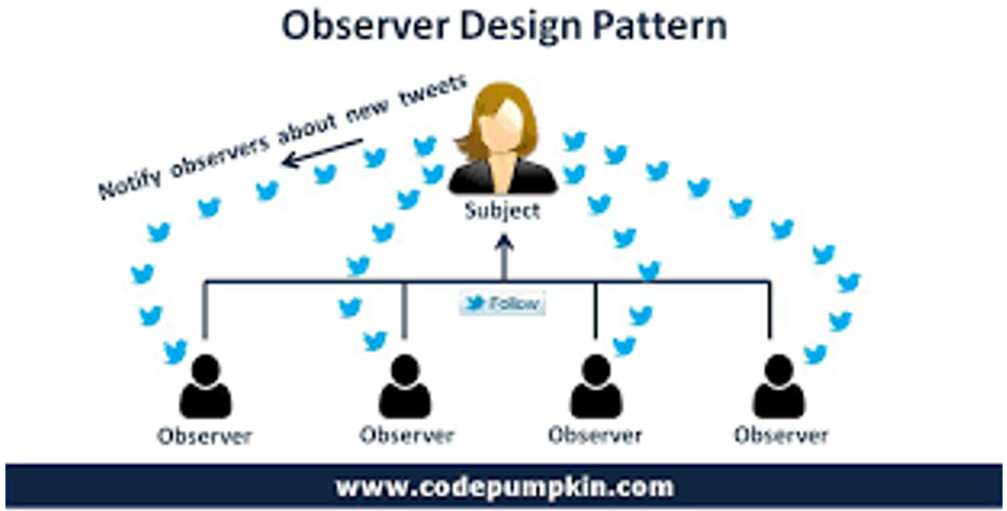
Extras que vamos a revisar

Para construir la app indicada, vamos a tener que repasar conceptos pasados y conocer algunos nuevos elementos por ejemplo:

- Patrón Observador y LiveData
- Arquitectura recomendada
- Repositorio.
- Coroutines.

Patrón observador

Es un patrón de diseño que define una dependencia de tipo uno a muchos entre objetos. Cuando el **objeto observado** cambia de estado, todos **los dependientes son notificados.**



en.wikipedia.org/wiki/Observer

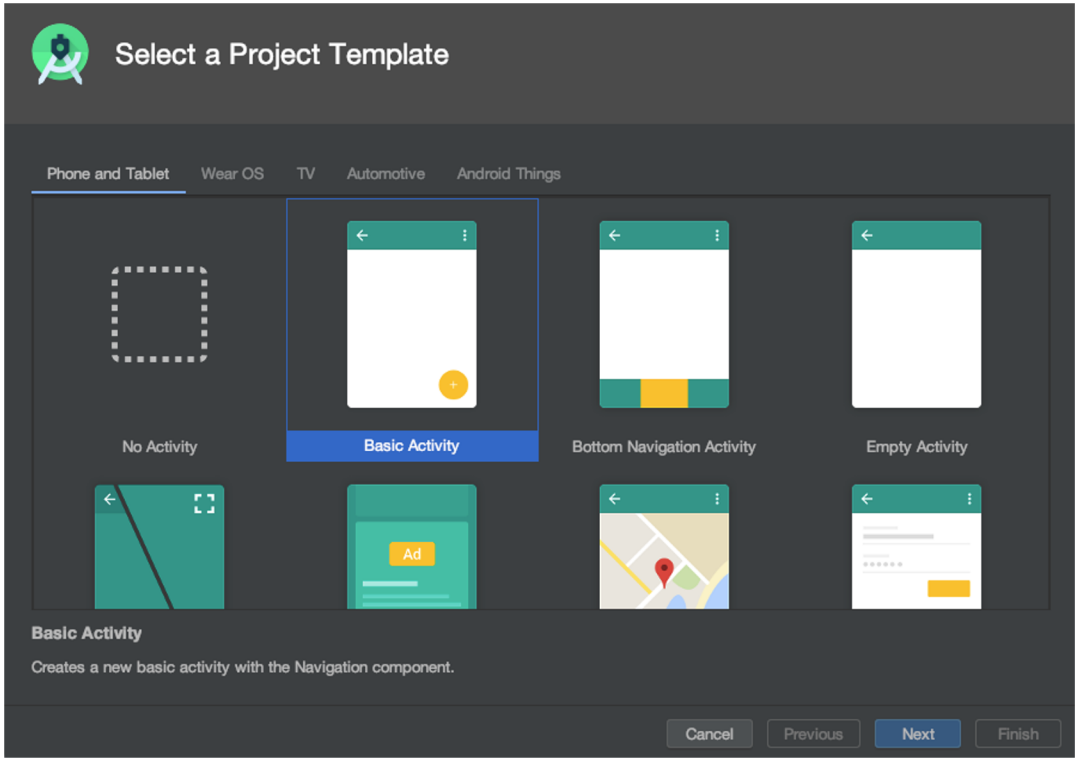
LiveData

Es parte de Android JetPack, y según su definición oficial es una clase de contenedor de datos observable. A diferencia de una clase observable regular, **Live Data** está optimizada para ciclos de vida, lo que significa que respeta el ciclo de vida de otros componentes de las apps, como actividades, fragmentos o servicios.

Aparte de esta y otras características, estamos revisando live data por la fácil implementación que tiene con Room.

Primero crearemos un nuevo proyecto

1. Selecciona un “Basic Activity” como base del proyecto



Añadir dependencias

Añadir dependencias:

- View Binding
- Room DB
- Biblioteca de navegación (revisar si está implementada)

```
// Room
def room_version = "2.3.0"
implementation "androidx.room:room-runtime:$room_version"
annotationProcessor"androidx.room:room-compiler:$room_version"
testImplementation "androidx.room:room-testing:$room_version"
```