A Database of Thrones

OSC: ADA

Baze Podataka

Mogucnosti spremanja podataka na Androidu:

- Internal file storage
- External file storage
- Shared preferences
- Baze podataka

Baze podataka:

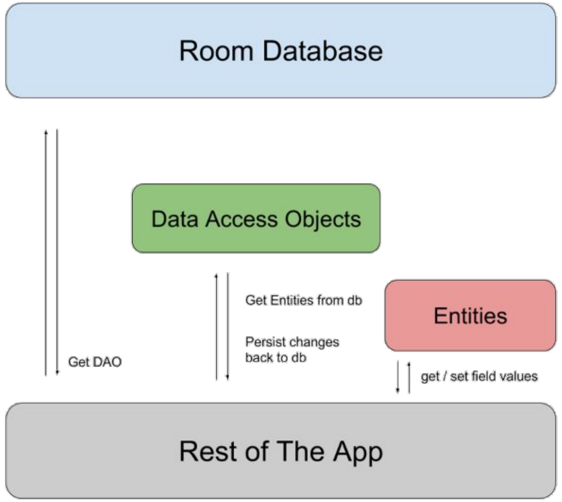
- Bazirane na SQLite-u
- Bazi se moze pristupiti samo iz aplikacije

Room

- abstraknti sloj temeljen na SQLite koji pojednostavnjuje rad s bazama

Room se sastoji od 3 glavne komponente:

- Baze podataka
- Entiteta
- DAO-a (Database access object)



Implementacija Room-a

Implementacija zapocinje u gradle fileovima

```
implementation "android.arch.persistence.room:runtime:$rootProject.roomVersion"
annotationProcessor "android.arch.persistence.room:compiler:$rootProject.roomVersion"
androidTestImplementation "android.arch.persistence.room:testing:$rootProject.roomVersion"

ext {
   roomVersion = '1.0.0'
}
```

Kreiranje entiteta (data modela)

Podaci za ovu aplikaciju su Taskovi, i svaki task je jedan Entitet.

Unutar klase Task potrebni su getteri i setteri za svako polje

```
@Entity(tableName = "task_table")
public class Task implements Serializable {
      //data
}
```

Kreiranje entiteta (data modela)

- Svaka baza treba primarni kljuc, te se to mora posebno naznaciti u data modelu
- Primarni kljuc ne smije biti vrijednosti “null”
- Svaki stupac u tablici moze imati ime po zelji

```
@PrimaryKey
@NonNull
@ColumnInfo(name = "id")
private String mId;
```

Kreiranje DAO-a

- DAO (Data Access Object) je klasa koja sluzi za pristup podacima u bazi
- Room automatski generira neke od cestih querija uz pomoc anotacija
- DAO mora biti interface ili abstraktna klasa

Kreiranje DAO-a

```
@Dao
public interface TaskDao {
    @Insert
    void insert(Task task);
    @Delete
    void delete(Task task);
    @Query("SELECT * from task_table ORDER BY mPriority ASC")
    List<Task> getAllTasks();
}
```

Kreiranje baze

- Room je database layer baziran na SQLite bazi
- Koristi se DAP za querijanje baze
- Room klasa mora biti abstraktna i mora extendati RoomDatabase klasu
- Potrebna je samo jedna instanca baze za cijelu aplikaciju

Kreiranje baze

- Kreirajte public abstract klasu koja extenda RoomDatabase i nazovite ju TaskRoomDatabase

```
public abstract class TaskRoomDatabase extends RoomDatabase{}
```

- Annotate klasu kako bi ista bila Room baza, i deklarirajte entitete koji pripadaju klasi
- Dodajte verziju baze
- Izlistanje entiteta ce kreirati tablice u bazi

```
@Database(entities = {Task.class}, version = 1)
```

- Definirajte DAO

```
public abstract TaskDao taskDao();
```

Kreiranje baze

```
@Database(entities = {Task.class}, version = 1)
public abstract class TaskRoomDatabase extends RoomDatabase {
      public abstract TaskDao taskDao();
}
```

- Ucinite TaskRoomDatabase singletonom kako nebi imali vise instanci baze otvoreno u isto vrijeme

```
private static TaskRoomDatabase INSTANCE;

public static TaskRoomDatabase getDatabase(final Context context) {
  if (INSTANCE == null) {
     synchronized (TaskRoomDatabase.class) {
        if (INSTANCE == null) {
           //create database here
        }
     }
  }
  return INSTANCE;
}
```

Kreiranje baze

```
INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
        TaskRoomDatabase.class, "task_database")
        .allowMainThreadQueries()
        .build();
```

Spremanje podataka u Room bazu

```
private TaskDao mTaskDao;

private void initDao() {
     TaskRoomDatabase database = TaskRoomDatabase.getDatabase(this);
     mTaskDao = database.taskDao();
 }

public void saveTask(){
      …….
      mTaskDao.insert(newTask);
}
```

Citanje podataka iz Room baze

```
private void updateTasksDisplay() {
	List<Task> tasks = mTaskDao.getAllTasks();
	mTaskAdapter.updateTasks(tasks);
	for (Task t : tasks) {
		Log.d(TAG, t.getTitle());
	}
}
```

TypeConverter

```
public class TypeConverterUtil {
  @TypeConverter
  public static TaskPriority fromString(String string) {
     return TaskPriority.valueOf(string);
  }
  @TypeConverter
  public static String fromTaskPriorty(TaskPriority taskPriority) {
     return taskPriority.toString();
  }
}
```

TypeConverter

```
@Database(entities = {Task.class}, version = 1)
@TypeConverters({TypeConverterUtil.class})
public abstract class TaskRoomDatabase extends RoomDatabase {
}
```

Realm

- Realm je data-driven framework koji se koristi za online i offline podatke
- Jednostavna SQL syntax baza koja ne zahtijeva rucno pisanje SQLa vec koristi ciste metode

Implementacija Realm-a

```
dependencies {
   classpath 'com.android.tools.build:gradle:3.1.2'
   classpath "io.realm:realm-gradle-plugin:5.1.0"

apply plugin: 'realm-android'
```

Inicijaliziranje Realm-a

```
public class TaskieApplication extends Application {
  @Override
  public void onCreate() {
     super.onCreate();
     Realm.init(this);
     RealmConfiguration realmConfig = new RealmConfiguration.Builder()
            .name("taskie.realm")
            .schemaVersion(0)
            .build();
     Realm.setDefaultConfiguration(realmConfig);
  }
}
```

Inicijaliziranje Realm-a

AndroidManifest.xml

```
<application
   android:name=".util.TaskieApplication"
</application>
```

Kreiranje entiteta (data modela)

```
public class Task extends RealmObject implements Serializable{
        @Required
        @PrimaryKey
        private String mId;

        mId = UUID.randomUUID().toString();
```

Kreiranje entiteta(data modela)

```java
private String mPriority;

public void saveTaskPriorityEnum(TaskPriority taskPriority) {
        this.mPriority = taskPriority.toString();
}
public TaskPriority getTaskPriorityEnum() {
        return TaskPriority.valueOf(mPriority);
}
public String convertTaskPriorityEnumToString(TaskPriority taskPriority) {
        return String.valueOf(taskPriority.toString());
}
```

Spremanje podataka u Realm bazu

```
private Realm mRealm;

@Override
protected void onCreate(Bundle savedInstanceState) {
        mRealm = Realm.getDefaultInstance();
}
```

Spremanje podataka u Realm bazu

```
mRealm.beginTransaction();
Task newTask = mRealm.createObject(Task.class,
UUID.randomUUID().toString());
newTask.setTitle(title);
newTask.setDescription(description);
newTask.setTaskPriorityEnum(priority);
mRealm.commitTransaction();
```

Citanje podataka iz Realm baze

```
private Realm mRealm;

@Override
protected void onCreate(Bundle savedInstanceState) {
        mRealm = Realm.getDefaultInstance();
}
```

Citanje podataka iz Realm baze

```
private void updateTasksDisplay() {
    RealmResults<Task> tasks = mRealm.where(Task.class).findAll();
    mTaskAdapter.updateTasks(tasks);
    for (Task t : tasks) {
        Log.d(TAG, t.getTitle());
    }
}
```

Brisanje podataka iz Realm baze

```
RealmResults<Task> rows= mRealm.where(Task.class).equalTo("id", id).findAll();
rows.deleteAllFromRealm();
```

Shared Preferences

```
PreferenceManager.getDefaultSharedPreferences(this).edit().putString("MYLABEL", "myStringToSave").apply();



PreferenceManager.getDefaultSharedPreferences(this).getString("MYLABEL", "defaultStringIfNothingFound");
```

Zadaca

1. Kreirati menu u TasksActivity uz pomoc kojega se taskovi mogu filtrirati po prioritetu.

2. Uz pomoc AlertDialoga omoguciti korisniku da dugim klikom moze obrisati taskove.

3. Uz pomoc SharedPreferences spremiti zadnji prioritet koristen od strane korisnika, te isti ponovo staviti kao defaultni pri kreiranju novog taska

4. Kreirati novi data model s kategorijama koje korisnik moze dodavati u bazu podata, te ih kasnije koristiti pri kreiranju novih taskova.

5. (Bonus zadatak, nije obavezan) Kreirati Upotrijebiti ORMLite library umjesto Room/Realm librarija.