Laboratorijska vježba 9

Perzistencija podataka u Android aplikacijama

Zadatak 1

Omogućiti da se klikom na odgovarajuće dugme u detaljima o filmu, film doda u bazu podataka kao omiljeni film. Film se treba prikazivati u listi omiljenih filmova.

U prvom koraku dodat ćemo odgovarajuće dependency-e u gradle.

```
implementation("androidx.room:room-runtime:2.3.0")
annotationProcessor "androidx.room:room-compiler:2.3.0"
implementation("androidx.room:room-ktx:2.3.0")
kapt("androidx.room:room-compiler:2.3.0")
```

Za potrebe `kapt` dodat ćemo i `plugin`:

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

Proširit ćemo klasu `Movie` s odgovarajućim anotacijama:

```
@Entity
data class Movie (
        @PrimaryKey @SerializedName("id") var id: Long,
        @ColumnInfo(name = "title") @SerializedName("title")  var title: String,
        @ColumnInfo(name = "overview") @SerializedName("overview")  var overview:
String,
        @ColumnInfo(name = "release_date") @SerializedName("release_date")   var
releaseDate: String,
        @ColumnInfo(name = "homepage") @SerializedName("homepage")   var homepage:
String?,
        @ColumnInfo(name = "poster_path") @SerializedName("poster_path") var
posterPath: String?,
        @ColumnInfo(name = "backdrop_path") @SerializedName("backdrop_path")  var
backdropPath: String?
)
```

Kreirat ćemo `MovieDAO` s metodama za dobavljanje svih filmova i za upis filmova u bazu (za sada):

```
@Dao
interface MovieDao {
    @Query("SELECT * FROM movie")
    suspend fun getAll(): List<Movie>
    @Insert
    suspend fun insertAll(vararg movies: Movie)
}
```

Kreirat ćemo `AppDatabase` klasu. U sklopu iste ćemo dodati i companion objekat koji će instancirati bazu te će se na taj način održati singleton pattern.

```
@Database(entities = arrayOf(Movie::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun movieDao(): MovieDao
    companion object {
        private var INSTANCE: AppDatabase? = null
        fun getInstance(context: Context): AppDatabase {
            if (INSTANCE == null) {
                synchronized(AppDatabase::class) {
                    INSTANCE = buildRoomDB(context)
                }
            }
            return INSTANCE!!
        }
        private fun buildRoomDB(context: Context) =
            Room.databaseBuilder(
                context.applicationContext,
                AppDatabase::class.java,
                "cinaeste-db"
            ).build()
    }
}
```

Napomena: Context ćemo proslijeđivati kroz odgovarajuće metode ili instancirati unutar ViewModel-a.

Bazu možemo podesiti da blokira glavnu nit. Međutim, ovo se ne savjetuje u praksi i na prethodnim vježbama smo radili da je uvijek bolje koristiti coroutine i prebaciti sve na IO nit.

Definišimo odgovarajuće metode za upis filma i dobavljanje filmova u/iz baze podataka unutar `MovieRepository` klase.

```
suspend fun getFavoriteMovies(context: Context) : List<Movie> {
    return withContext(Dispatchers.IO) {
        var db = AppDatabase.getInstance(context)
        var movies = db!!.movieDao().getAll()
        return@withContext movies
    }
}
suspend fun writeFavorite(context: Context,movie:Movie) : String?{
    return withContext(Dispatchers.IO) {
        try{
            var db = AppDatabase.getInstance(context)
            db!!.movieDao().insertAll(movie)
            return@withContext "success"
        }
        catch(error:Exception){
            return@withContext null
        }
    }
}
```

U sklopu `MovieListViewModel-a` ćemo definisati metodu za poziv metode iz repozitorija.

```
fun getFavorites(context: Context, onSuccess: (movies: List<Movie>) -> Unit,
                    onError: () -> Unit){
        scope.launch{
           val result = MovieRepository.getFavoriteMovies(context)
            when (result) {
                is List<Movie> -> onSuccess?.invoke(result)
                else-> onError?.invoke()
            }
        }
    }
```

U sklopu `MovieDetailViewModel`-a ćemo definisati odgovarajuću metodu za upis filma:

```
fun writeDB(context: Context, movie:Movie, onSuccess: (movies: String) -> Unit,
                onError: () -> Unit){
    scope.launch{
        val result = MovieRepository.writeFavorite(context,movie)
        when (result) {
            is String -> onSuccess?.invoke(result)
            else-> onError?.invoke()
        }
    }
}
```

U `FavoritesMovieFragment` izvršit ćemo u `onCreate` poziv metode iz ViewModel-a i implementirati metode `onSuccess` i `onError`:

```
context?.let {
    movieListViewModel.getFavorites(
        it,onSuccess = ::onSuccess,
        onError = ::onError)
}
```

Odgovarajuće metode:

```
fun onSuccess(movies:List<Movie>){
      favoriteMoviesAdapter.updateMovies(movies)
}
fun onError() {
    val toast = Toast.makeText(context, "Error", Toast.LENGTH_SHORT)
    toast.show()
}
```

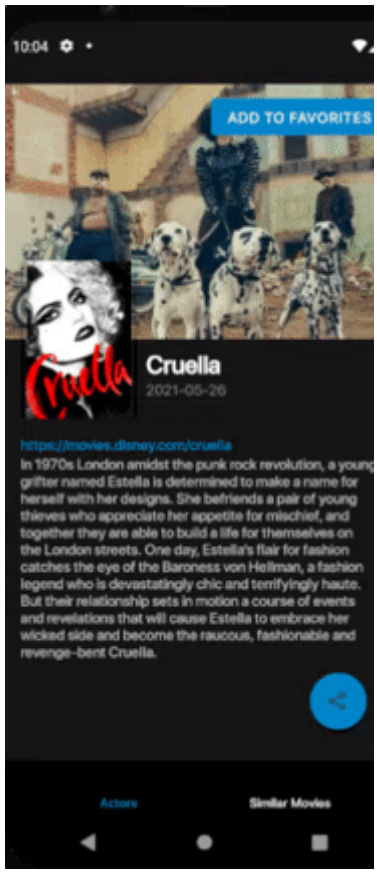
U sklopu `MovieDetailActivity` ćemo dodati button za dodavanje u favorite i dodat ćemo metodu za upis i rezultantnu poruku.

```
fun writeDB(){
    movieDetailViewModel.writeDB(applicationContext,this.movie,onSuccess =
::onSuccess1,
        onError = ::onError)
}
fun onSuccess1(message:String){
    val toast = Toast.makeText(applicationContext, "Spaseno", Toast.LENGTH_SHORT)
```

```
    toast.show()
    addFavorite.visibility= View.GONE
}
fun onError() {
    val toast = Toast.makeText(applicationContext, "Error", Toast.LENGTH_SHORT)
    toast.show()
}
```

Kroz ovaj kôd omogućili smo dodavanje filma u omiljene i prikaz omiljenih filmova korištenjem SQLite baze podataka.

Rezultantna aplikacija je data na sljedećoj slici:



Zadatak za samostalni rad

1. Izvršite spašavanje sličnih glumaca i sličnih filmova u odgovarajuće tabele.
2. Omogućite korisniku da klikom na dugme ukloni film iz liste omiljenih filmova.