**Section 30 – MVVM + Retrofit – The Movie App**

**Goal:** Build a Movie listing app that fetches data from an API (The Movie DB) using **MVVM architecture** + **Retrofit**.

**Step 0 – MVVM + Retrofit Flow Overview**

UI (Activity/Fragment)

↑ observes LiveData

ViewModel

↑ calls

Repository

↑ makes API calls via

Retrofit Service Interface

↑ fetches JSON from

The Movie DB API

↓ returns mapped objects to Repository

ViewModel updates LiveData → UI updates automatically

**Step 1 – Add Dependencies**

**Java version (Gradle)**

// Retrofit

implementation 'com.squareup.retrofit2:retrofit:2.x.x'

implementation 'com.squareup.retrofit2:converter-gson:2.x.x'

// LiveData + ViewModel

implementation "androidx.lifecycle:lifecycle-livedata:2.x.x"

implementation "androidx.lifecycle:lifecycle-viewmodel:2.x.x"

// Glide for images

implementation 'com.github.bumptech.glide:glide:4.x.x'

annotationProcessor 'com.github.bumptech.glide:compiler:4.x.x'

**Kotlin best-practice additions**

// Coroutines support for Retrofit

implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.x.x'

implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.x.x'

*(This will allow you to replace callbacks with cleaner suspend functions.)*

**Step 2 – Define Model Classes (Match API JSON)**

Example JSON snippet from TMDB API:

{

"results": [

{

"title": "Movie Title",

"poster\_path": "/abc123.jpg",

"release\_date": "2025-08-01"

}

]

}

**Java**

public class Movie {

@SerializedName("title") // Maps JSON key to Java field

private String title;

@SerializedName("poster\_path")

private String posterPath;

@SerializedName("release\_date")

private String releaseDate;

public String getTitle() { return title; }

public String getPosterPath() { return posterPath; }

public String getReleaseDate() { return releaseDate; }

}

public class MovieResponse {

@SerializedName("results")

private List<Movie> results;

public List<Movie> getResults() { return results; }

}

**Kotlin best-practice**

data class Movie(

val title: String,

@SerializedName("poster\_path") val posterPath: String,

@SerializedName("release\_date") val releaseDate: String

)

data class MovieResponse(

val results: List<Movie>

)

*(Data classes are immutable and better for modern Android.)*

**Step 3 – Retrofit API Interface**

**Java**

public interface MovieApi {

@GET("movie/popular")

Call<MovieResponse> getPopularMovies(

@Query("api\_key") String apiKey

);

}

**Kotlin best-practice**

interface MovieApi {

@GET("movie/popular")

suspend fun getPopularMovies(

@Query("api\_key") apiKey: String

): Response<MovieResponse>

}

*(Using suspend allows coroutine-based async calls without callbacks.)*

**Step 4 – Retrofit Client Singleton**

**Java**

public class RetrofitInstance {

private static final String BASE\_URL = "https://api.themoviedb.org/3/";

private static Retrofit retrofit;

public static MovieApi getMovieApi() {

if (retrofit == null) {

retrofit = new Retrofit.Builder()

.baseUrl(BASE\_URL)

.addConverterFactory(GsonConverterFactory.create())

.build();

}

return retrofit.create(MovieApi.class);

}

}

**Kotlin best-practice**

object RetrofitInstance {

private const val BASE\_URL = "https://api.themoviedb.org/3/"

val api: MovieApi by lazy {

Retrofit.Builder()

.baseUrl(BASE\_URL)

.addConverterFactory(GsonConverterFactory.create())

.build()

.create(MovieApi::class.java)

}

}

**Step 5 – Repository**

**Java**

public class MovieRepository {

private final MovieApi movieApi;

public MovieRepository() {

movieApi = RetrofitInstance.getMovieApi();

}

public void getMovies(MutableLiveData<List<Movie>> liveData) {

movieApi.getPopularMovies("YOUR\_API\_KEY").enqueue(new Callback<MovieResponse>() {

@Override

public void onResponse(Call<MovieResponse> call, Response<MovieResponse> response) {

if (response.isSuccessful() && response.body() != null) {

liveData.setValue(response.body().getResults());

}

}

@Override

public void onFailure(Call<MovieResponse> call, Throwable t) {

liveData.setValue(new ArrayList<>()); // empty list on error

}

});

}

}

**Kotlin best-practice**

class MovieRepository(private val api: MovieApi) {

suspend fun getMovies(apiKey: String): List<Movie> {

val response = api.getPopularMovies(apiKey)

return if (response.isSuccessful) response.body()?.results ?: emptyList()

else emptyList()

}

}

**Step 6 – ViewModel**

**Java**

public class MovieViewModel extends ViewModel {

private final MovieRepository repository;

private final MutableLiveData<List<Movie>> movies = new MutableLiveData<>();

public MovieViewModel() {

repository = new MovieRepository();

}

public LiveData<List<Movie>> getMovies() {

return movies;

}

public void fetchMovies() {

repository.getMovies(movies);

}

}

**Kotlin best-practice**

class MovieViewModel(private val repository: MovieRepository) : ViewModel() {

private val \_movies = MutableLiveData<List<Movie>>()

val movies: LiveData<List<Movie>> = \_movies

fun fetchMovies(apiKey: String) {

viewModelScope.launch {

\_movies.value = repository.getMovies(apiKey)

}

}

}

*(ViewModelScope ensures coroutines are tied to ViewModel lifecycle.)*

**Step 7 – RecyclerView Adapter**

**Java**

public class MovieAdapter extends RecyclerView.Adapter<MovieAdapter.MovieViewHolder> {

private List<Movie> movies = new ArrayList<>();

static class MovieViewHolder extends RecyclerView.ViewHolder {

TextView title, releaseDate;

ImageView poster;

MovieViewHolder(View itemView) {

super(itemView);

title = itemView.findViewById(R.id.text\_title);

releaseDate = itemView.findViewById(R.id.text\_release\_date);

poster = itemView.findViewById(R.id.image\_poster);

}

}

@Override

public void onBindViewHolder(MovieViewHolder holder, int position) {

Movie current = movies.get(position);

holder.title.setText(current.getTitle());

holder.releaseDate.setText(current.getReleaseDate());

Glide.with(holder.poster.getContext())

.load("https://image.tmdb.org/t/p/w500" + current.getPosterPath())

.into(holder.poster);

}

public void setMovies(List<Movie> movies) {

this.movies = movies;

notifyDataSetChanged();

} }

**Kotlin best-practice**

class MovieAdapter : RecyclerView.Adapter<MovieAdapter.MovieViewHolder>() {

private val movies = mutableListOf<Movie>()

inner class MovieViewHolder(val binding: MovieItemBinding) : RecyclerView.ViewHolder(binding.root)

override fun onBindViewHolder(holder: MovieViewHolder, position: Int) {

val movie = movies[position]

binding.textTitle.text = movie.title

binding.textReleaseDate.text = movie.releaseDate

Glide.with(binding.imagePoster.context)

.load("https://image.tmdb.org/t/p/w500${movie.posterPath}")

.into(binding.imagePoster)

}

fun setMovies(list: List<Movie>) {

movies.clear()

movies.addAll(list)

notifyDataSetChanged()

} }

**Step 8 – Activity**

**Java**

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

RecyclerView recyclerView = findViewById(R.id.recycler\_view);

recyclerView.setLayoutManager(new LinearLayoutManager(this));

MovieAdapter adapter = new MovieAdapter();

recyclerView.setAdapter(adapter);

MovieViewModel viewModel = new ViewModelProvider(this).get(MovieViewModel.class);

viewModel.getMovies().observe(this, adapter::setMovies);

viewModel.fetchMovies();

}

}

**Kotlin best-practice**

class MainActivity : AppCompatActivity() {

private val viewModel: MovieViewModel by viewModels()

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

val binding = ActivityMainBinding.inflate(layoutInflater)

setContentView(binding.root)

val adapter = MovieAdapter()

binding.recyclerView.layoutManager = LinearLayoutManager(this)

binding.recyclerView.adapter = adapter

viewModel.movies.observe(this) { adapter.setMovies(it) }

viewModel.fetchMovies("YOUR\_API\_KEY")

}

}

**Part B – Extra & Future-Ready**

1. **Show Loading & Error States**
   * Use a sealed class UiState with Loading, Success, and Error.
2. **Secure API Keys**
   * Store in local.properties or use NDK.
3. **Pagination**
   * Use **Paging 3** for endless scrolling.
4. **Offline Caching**
   * Combine Retrofit with Room DB.
5. **DiffUtil**
   * Replace notifyDataSetChanged() for performance.
6. **Dependency Injection**
   * Use Hilt to inject Repository, API, ViewModel.
7. **Testing**
   * Use MockWebServer to test API responses.