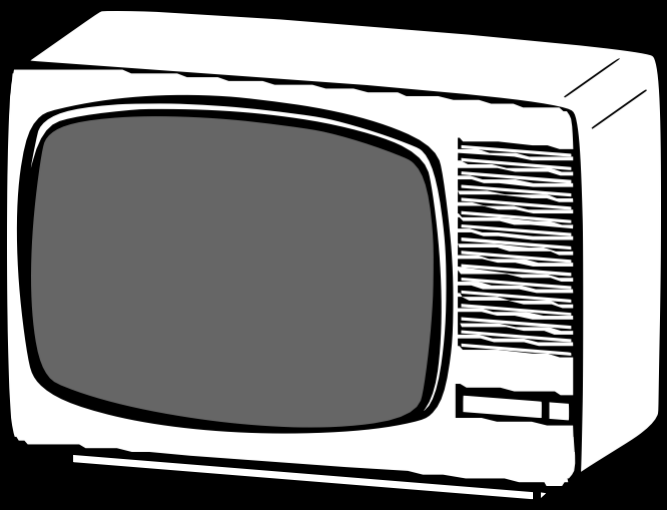


Make A

Movies App

in

Android

Make a Movies App Using TMDb API

Contents

Project Specifications

The Movie Database (TMDb) is a community-built movie and TV database. Their database contains a lot of movies, tv shows, artists and many more.

Using their API is a great way to get started with networking in Android. Let's first know the details of the app that we're going to make.

App Specifications

Architecture: None

The main third-party libraries that we're going to use are:

- Retrofit - to handle our network related features
- Gson - parsing JSON objects from TMDb API
- Glide - image loading library

Features

Feature #1: As a user, I want to see a list of movies so that I can browse through different movies.

Tasks:

1. It should be a horizontal list.
2. Each item should display an image of the movie.
3. Each item should have rounded corners.

Feature #2: As a user, I want the list of movies to be categorized by Popular, Top Rated and Upcoming so that I can easily look for movies based on the them.

Tasks:

1. Popular movies should be the first row.
2. Top Rated movies should be the second row.
3. Upcoming movies should be the last row.
4. Popular movies row should have a "Popular" label above it.
5. Top Rated movies row should have a "Top Rated" label above it.
6. Upcoming movies row should have an "Upcoming" label above it.

Feature #3: As a user, I want to be able to see the details of a movie so that I will know more about the movie.

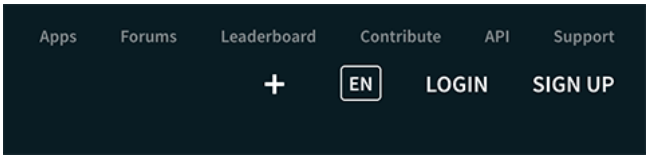
Tasks:

1. It should show the cover photo of the movie.
2. It should show the poster of the movie.
3. It should show the title of the movie.
4. It should show the summary of the movie.
5. It should show the rating of the movie.
6. It should show the release date of the movie.

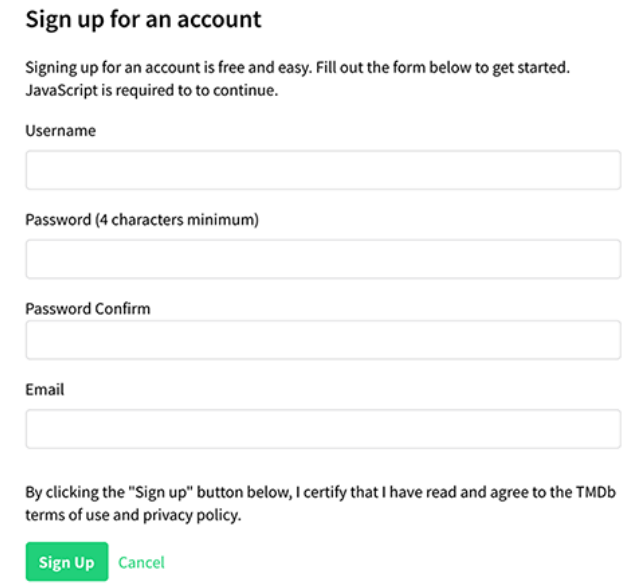
Getting a TMDb API Key

Create a TMDb Account

1. Go to The Movie Database (TMDb) Website.
2. Click "**SIGN UP**" at the top right corner.

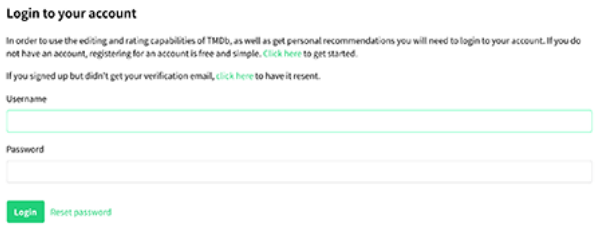


3. Enter your username, password and email.



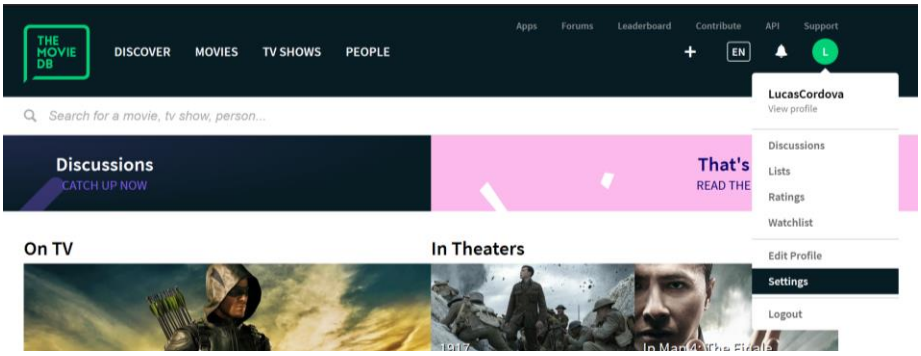
4. After you sign up, check your email and click "**ACTIVATE MY ACCOUNT**" to verify.

5. Login to your account.

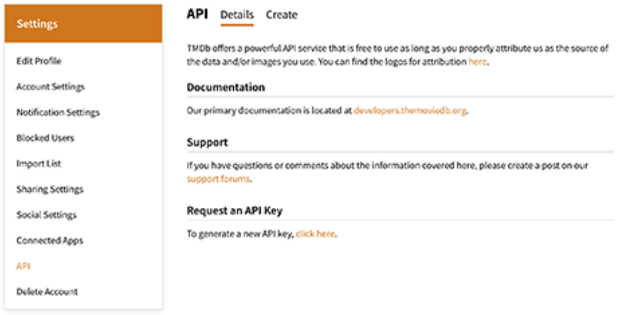


Request an API Key

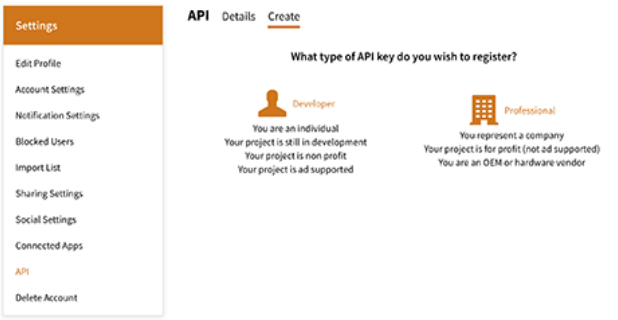
1. Go to Settings.



2. In the sidebar, select API.
3. Under Request an API key, click "**click here**".

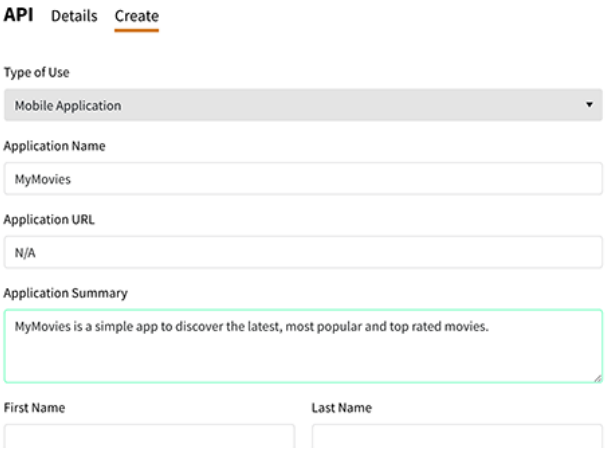


4. Select "**Developer**" and scroll down to the bottom and click Accept.

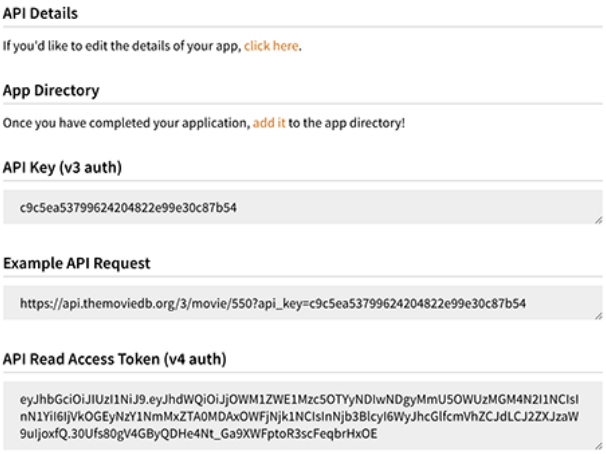


5. Fill in the details of your app and your personal details.

- Type of Use - Mobile Application
- Application Name - you can name it whatever you want
- Application URL - just put N/A because we don't have a url for this app
- Application Summary - a brief summary of what your app is all about
- Fill in the rest of details.



6. After you've completed the details, you should see your API key under **API Key (v3 Auth)**.
7. Open the url under **Example API request** and you should receive a JSON response.



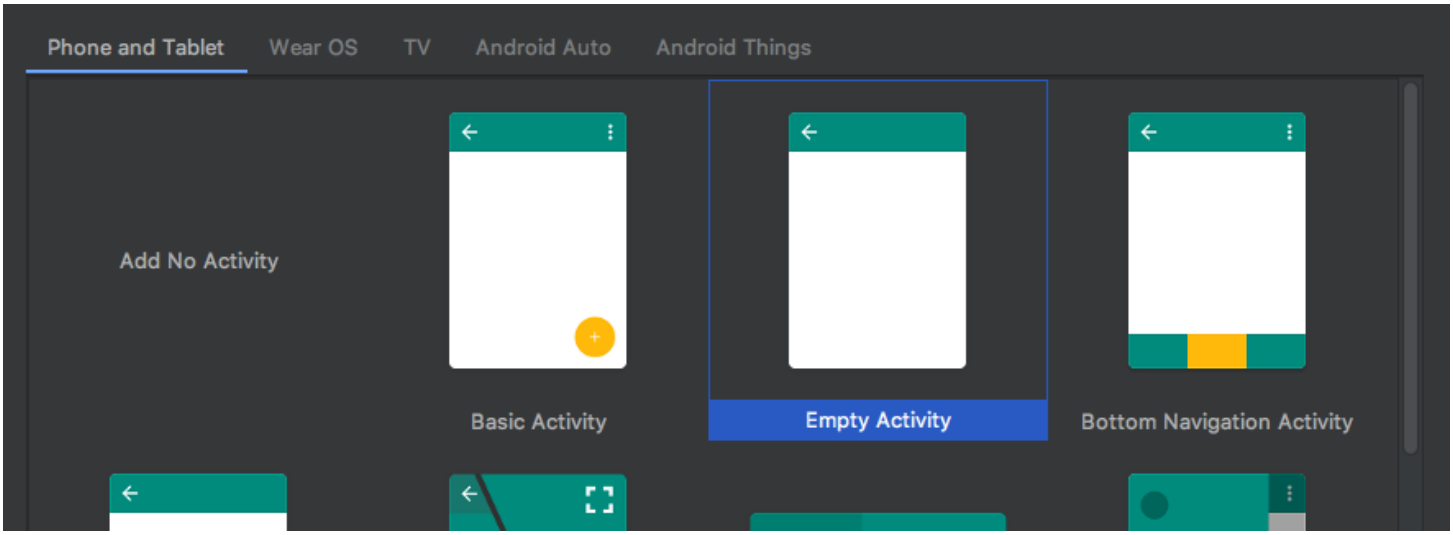
You can save your API key anywhere that you want or you can come back to it later by going to **Settings -> API**.

Now that this stuff is out of the way, let's start coding!

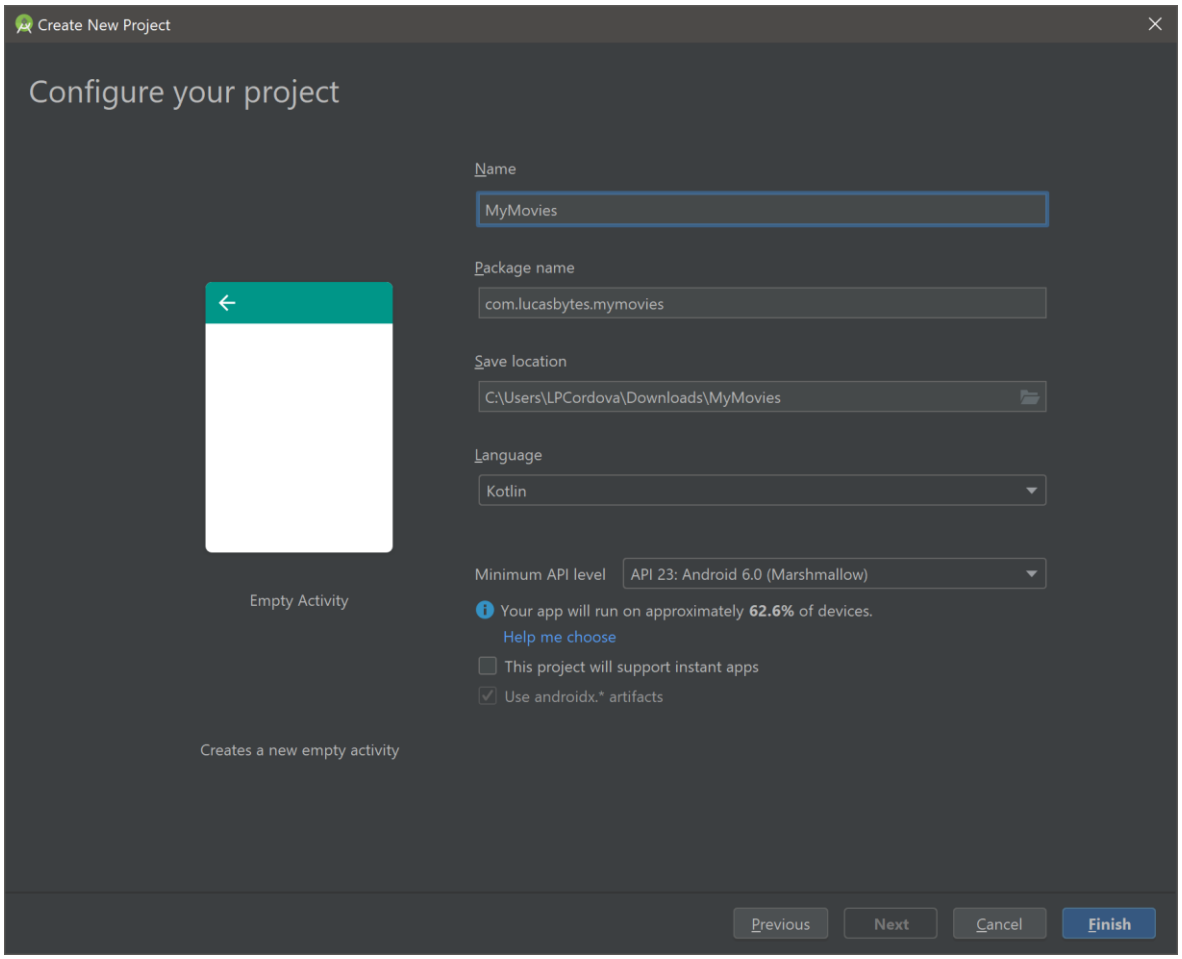
Feature #1: As a user, I want to see a list of movies so that I can browse through different movies

Create a New Project

1. Open Android Studio and start a new project.
2. Select Empty Activity.



3. You can name the app whatever you want.
4. For this project, we set our minimum API level to 21.
5. Make sure androidx.* artifacts is checked and click Finish.



Import Dependencies

After it's done building, open your app-level build.gradle and add these dependencies. Note you may already have some.

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-kapt'
```

```
android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.lucasbytes.mymovies"
        minSdkVersion 23
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    // Kotlin
    implementation"org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"

    // Support
    // https://developer.android.com/jetpack/androidx/versions
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'androidx.recyclerview:recyclerview:1.0.0'
    implementation 'androidx.cardview:cardview:1.0.0'

    // ViewModel and LiveData
    //   https://developer.android.com/jetpack/androidx/releases/lifecycle
    implementation  "androidx.lifecycle:lifecycle-extensions:2.1.0"

    // Retrofit
    // https://github.com/square/retrofit
    implementation 'com.squareup.retrofit2:retrofit:2.6.1'
    implementation 'com.squareup.retrofit2:converter-gson:2.6.1'

    // Gson
    // https://github.com/google/gson
    implementation 'com.google.code.gson:gson:2.8.5'

    // Glide
    // https://github.com/bumptech/glide
    implementation 'com.github.bumptech.glide:glide:4.10.0'
    kapt 'com.github.bumptech.glide:compiler:4.10.0'

    // Testing
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.2.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

As of right now, those are the latest versions. Make sure to check the latest version of each dependency from the links provided. Click Sync Now at the top right corner and we're done setting up our project.

Project Resources

Open your **strings.xml** and add these strings.

```
<resources>
    <string name="app_name">MyMovies</string>
    <string name="popular">Popular</string>
    <string name="most_popular_movies">Most popular movies</string>
    <string name="error_fetch_movies">Please check your internet connection</string>
    <string name="top_rated">Top Rated</string>
    <string name="highest_rated_movies">Highest rated movies of all time</string>
    <string name="upcoming">Upcoming</string>
    <string name="stay_updated">Stay updated with the latest movies</string>
</resources>
```

Open your **colors.xml** and change the colors.

```
<resources>
    <color name="colorPrimary">#212121</color>
    <color name="colorPrimaryDark">#000000</color>
    <color name="colorAccent">#FF5252</color>
</resources>
```

Open your **styles.xml** and change the AppTheme to Theme.AppCompat.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Fetch Movies from TMDb API Using Retrofit

In this section, we will use Retrofit to connect to TMDb's API.

1. Add **INTERNET** permission in AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.lucasbytes.mymovies">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        ...
    </application>

</manifest>
```

2. Create a new data class called **Movie**.

```
data class Movie(
    @SerializedName("id") val id: Long,
    @SerializedName("title") val title: String,
    @SerializedName("overview") val overview: String,
    @SerializedName("poster_path") val posterPath: String,
    @SerializedName("backdrop_path") val backdropPath: String,
    @SerializedName("vote_average") val rating: Float,
    @SerializedName("release_date") val releaseDate: String
)
```

3. Create a new data class called **GetMoviesResponse**.

```
data class GetMoviesResponse(
    @SerializedName("page") val page: Int,
    @SerializedName("results") val movies: List<Movie>,
    @SerializedName("total_pages") val pages: Int
)
```

4. Create a new interface called **Api**. For all imports, if you are offered multiple options by pressing alt-enter, be sure to choose the retrofit one.

```
interface Api {

    @GET("movie/popular")
    fun getPopularMovies(
        @Query("api_key") apiKey: String = "YOUR_API_KEY_HERE",
        @Query("page") page: Int
    ): Call<GetMoviesResponse>
}
```

Be sure to replace **YOUR_API_KEY_HERE** with your own API key that you generated from the previous chapter.

5. Create a new object called **MoviesRepository**.

```
object MoviesRepository {

    private val api: Api

    init {
        val retrofit = Retrofit.Builder()
            .baseUrl("https://api.themoviedb.org/3/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        api = retrofit.create(Api::class.java)
    }
}
```

Take note that it uses the object keyword of Kotlin which is an easy way to declare a **Singleton** in Kotlin.

Using the init block of Kotlin which is called when an instance is initialized, we instantiate a Retrofit instance using its builder. Then, instantiate an instance of Api using the Retrofit instance.

6. Add a new method in MoviesRepository called getPopularMovies().

```
object MoviesRepository {

    ...

    fun getPopularMovies(page: Int = 1) {
        api.getPopularMovies(page = page)
            .enqueue(object : Callback<GetMoviesResponse> {
                override fun onResponse(
                    call: Call<GetMoviesResponse>,
                    response: Response<GetMoviesResponse>
                ) {
                    if (response.isSuccessful) {
                        val responseBody = response.body()

                        if (responseBody != null) {
                            Log.d("Repository", "Movies: ${responseBody.movies}")
                        } else {
                            Log.d("Repository", "Failed to get response")
                        }
                    }
                }
```

```
                override fun onFailure(call: Call<GetMoviesResponse>, t: Throwable) {
                    Log.e("Repository", "onFailure", t)
                }
            })
    }
}
```

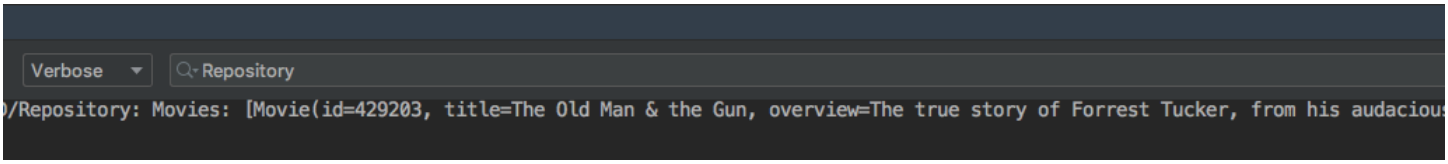
For now, we default the page to 1. We will deal with pagination later on.

First off, we execute api.getPopularMovies() asynchronously using the .enqueue() method. Then, we log the movies if the response was successful.

7. Open your **MainActivity** and call the getPopularMovies() method of MoviesRepository.

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        MoviesRepository.getPopularMovies()
    }
}
```

8. Run the app and check your Logcat.



It should log the movies list from the response. Type **Repository** to easily find the log.

Callbacks Using Kotlin's Higher-Order Functions

In Java, we're used to creating an interface to represent callbacks in our code like this:

```
public interface OnGetMoviesCallback {

    void  onSuccess(List<Movie> movies);

    void onError();
}
```

In Kotlin, we no longer need to do that because we can pass a function to another function and we call it in Kotlin - **higher-order functions**. Let's take a closer look.

1. Open your MoviesRepository and let's refactor getPopularMovies().

```
fun  getPopularMovies(
        page: Int = 1,
        onSuccess: (movies: List<Movie>) -> Unit,
        onError: () -> Unit
    ) {
        ...
    }
```

onSuccess is a parameter that is a function that doesn't return anything -> Unit but it accepts a list of movies.

onError is the same with onSuccess but it doesn't accept anything. All we need to is to just invoke this method.

How do we use it?

2. In your getPopularMovies() method, remove the logs and replace it invocations of the functions.

```
fun getPopularMovies(
        page: Int = 1,
        onSuccess: (movies: List<Movie>) -> Unit,
        onError: () -> Unit
    ) {
        api.getPopularMovies(page = page)
            .enqueue(object : Callback<GetMoviesResponse> {
                override fun onResponse(
                    call: Call<GetMoviesResponse>,
                    response: Response<GetMoviesResponse>
                ) {
                    if (response.isSuccessful) {
                        val responseBody = response.body()

                        if (responseBody != null) {
                            onSuccess.invoke(responseBody.movies)
                        } else {
                            onError.invoke()
                        }
                    } else {
                        onError.invoke()
                    }
                }

                override fun onFailure(call: Call<GetMoviesResponse>, t: Throwable) {
                    onError.invoke()
                }
            })
    }
```

invoke() is how you execute a higher-order function. Take note that it will vary depending if the higher-order function has parameter(s) or not. You can see the difference by comparing onSuccess.invoke(responseBody.movies) and onError.invoke().

onSuccess: (movies: List<Movie>) -> Unit is to onSuccess.invoke(responseBody.movies).

onError: () -> Unit is to onError.invoke().

3. Open your **MainActivity** and let's pass the functions needed by getPopularMovies().

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        MoviesRepository.getPopularMovies(
            onSuccess = ::onPopularMoviesFetched,
            onError = ::onError
        )
    }

    private fun onPopularMoviesFetched(movies: List<Movie>) {
        Log.d("MainActivity", "Movies: $movies")
    }

    private fun onError() {
        Toast.makeText(this, getString(R.string.error_fetch_movies), Toast.LENGTH_SHORT).show()
    }
}
```

The :: colon operator is used to create a class or function reference. An alternative is doing this:

```
MoviesRepository.getPopularMovies(
    onSuccess = { movies ->
        Log.d("MainActivity", "Movies: $movies")
    },
    onError = {
        Toast.makeText(this, getString(R.string.error_fetch_movies), Toast.LENGTH_SHORT).show()
    }
)
```

But using the **::** operator approach just make things much cleaner. However, I leave it to your preference on which approach you want.

4. Run the app, check your Logcat and be sure to type **MainActivity** in the search bar to filter the logs. You should see a log that is the same as the previous section.

   If you'd like to know more about Kotlin's higher-order functions. Check out the documentation.

Create a Horizonal List and Load Images Using Glide

Now that we can finally fetch movies from TMDb, it's time to show these movies to your UI.

1. Open your **activity_main.xml** and add a RecyclerView for popular movies.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:text="@string/popular"
        android:textColor="@android:color/white"
        android:textSize="18sp"
        android:textStyle="bold"  />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:text="@string/most_popular_movies" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/popular_movies"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:clipToPadding="false"
        android:paddingStart="16dp"
        android:paddingEnd="16dp" />

</LinearLayout>
```

2. Under **res->layout** folder, create a new layout resource file called **item_movie.xml**.

```
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="128dp"
```

```
    android:layout_height="172dp"
    android:layout_marginEnd="8dp"
    app:cardCornerRadius="4dp">

    <ImageView
        android:id="@+id/item_movie_poster"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</androidx.cardview.widget.CardView>
```

3. Create a new class called **MoviesAdapter**.

```
class MoviesAdapter(
    private var movies: List<Movie>
) : RecyclerView.Adapter<MoviesAdapter.MovieViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieViewHolder {
        val view = LayoutInflater
            .from(parent.context)
            .inflate(R.layout.item_movie, parent, false)
        return MovieViewHolder(view)
    }

    override fun getItemCount(): Int = movies.size

    override fun onBindViewHolder(holder: MovieViewHolder, position: Int) {
        holder.bind(movies[position])
    }

    fun updateMovies(movies: List<Movie>) {
        this.movies = movies
        notifyDataSetChanged()
    }

    inner class MovieViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

        private val poster: ImageView = itemView.findViewById(R.id.item_movie_poster) fun

        bind(movie: Movie) {
            Glide.with(itemView)
                .load("https://image.tmdb.org/t/p/w342${movie.posterPath}")
                .transform(CenterCrop())
                .into(poster)
        }
    }
}
```

.load("https://image.tmdb.org/t/p/w342/<poster_url>") is how you fetch a poster of a movie from TMDb. You can learn more about fetching images from TMDb here.

Available poster sizes are: - w92 - w154 - w185 - w342 - w500 - w780 - original

You can go for original if you want to have the highest quality image but it will take time to load. A size of w342 should be enough for most screens.

4. Open your **MainActivity** and instantiate your RecyclerView and Adapter.

```
class MainActivity : AppCompatActivity() {

    private lateinit var popularMovies: RecyclerView
    private lateinit var popularMoviesAdapter: MoviesAdapter
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        popularMovies = findViewById(R.id.popular_movies)
        popularMovies.layoutManager = LinearLayoutManager(
            this,
            LinearLayoutManager.HORIZONTAL,
            false
        )
        popularMoviesAdapter = MoviesAdapter(listOf())
        popularMovies.adapter = popularMoviesAdapter

        MoviesRepository.getPopularMovies(
            onSuccess = ::onPopularMoviesFetched,
            onError = ::onError
        )
    }

    private fun onPopularMoviesFetched(movies: List<Movie>) {
        popularMoviesAdapter.updateMovies(movies)
    }

    ...
}
```

To make a horizontal list in RecyclerView, just provide the LinearLayoutManager with an orientation and a boolean flag that reverses the list or not.

```
popularMovies.layoutManager = LinearLayoutManager(
    this,
    LinearLayoutManager.HORIZONTAL,
    false
)
```

We removed the log in onPopularMoviesFetched() and replaced it by updating the movies inside popularMoviesAdapter.

```
private fun onPopularMoviesFetched(movies: List<Movie>) {
    popularMoviesAdapter.updateMovies(movies)
}
```

5. Run the app. Scroll through the list and take a moment to enjoy.

As of right now, these are the most popular movies. We might not have the same list by the time you've finished this section.

Pagination

While you're scrolling through the list, you'll notice that you only see a limited number of movies. Specifically, you only see 20 movies. Why is that?

TMDb has thousands and thousands of movies in their database. Imagine sending all those data into one API call. It would take a lot of time to receive the response and also the size of the response would be super big which is not ideal and efficient. Especially, when most of the time the user won't scroll all of it. That's why they paginate their movies API.

Open **Api** and you'll see a page parameter.

```
interface Api {

    @GET("movie/popular")
    fun getPopularMovies(
        @Query("api_key") apiKey: String = "YOUR_API_KEY_HERE",
        @Query("page") page: Int
    ): Call<GetMoviesResponse>
}
```

For now, in our **MoviesRepository** we default the page to 1.

```
object MoviesRepository {
    ...

    fun getPopularMovies(
        page: Int = 1,
        onSuccess: (movies: List<Movie>) -> Unit,
        onError: () -> Unit
    ) {
        ...
}
```

In this section, our goal is to fetch the next page of movies when the user scrolls halfway through our list.

1. Open your **MoviesAdapter**, change the type of the movies variable to MutableList, and rename your updateMovies() method to appendMovies().

```
class MoviesAdapter(
    private var movies: MutableList<Movie>
) : RecyclerView.Adapter<MoviesAdapter.MovieViewHolder>() {
    ...

    fun appendMovies(movies: List<Movie>) {
        this.movies.addAll(movies)
        notifyItemRangeInserted(
            this.movies.size,
            movies.size - 1
        )
    }

    ...
}
```

We changed the type of the movies variable to MutableList because we now have a dynamic list of movies.

```
class MoviesAdapter(
    private var movies: MutableList<Movie>
) : ...
```

Instead of using notifyDataSetChanged(), we use notifyItemRangeInserted() because we don't want to refresh the whole list. We just want to notify that there are new items added from this start and end positions.

```
fun appendMovies(movies: List<Movie>) {
    this.movies.addAll(movies)
    notifyItemRangeInserted(
        this.movies.size,
        movies.size - 1
    )
}
```

2. Open your **MainActivity** and declare a member variable for our page and LinearLayoutManager.

```
class MainActivity : AppCompatActivity() {

    private lateinit var popularMovies: RecyclerView
    private lateinit var popularMoviesAdapter: MoviesAdapter
    private lateinit var popularMoviesLayoutMgr: LinearLayoutManager

    private var popularMoviesPage = 1

    ...
}
```

3. In onCreate(), instantiate popularMoviesLayoutMgr variable, assign it to popularMovies RecyclerView, pass popularMoviesPage to MoviesRepository.getPopularMovies(), and an empty MutableList in MoviesAdapter.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    popularMovies = findViewById(R.id.popular_movies)
    popularMoviesLayoutMgr = LinearLayoutManager(
        this,
        LinearLayoutManager.HORIZONTAL,
        false
    )
    popularMovies.layoutManager = popularMoviesLayoutMgr
    popularMoviesAdapter = MoviesAdapter(mutableListOf())
    popularMovies.adapter = popularMoviesAdapter

    MoviesRepository.getPopularMovies(
        popularMoviesPage,
        ::onPopularMoviesFetched,
        ::onError
    )
}
```

4. Create a new method called getPopularMovies().

```
private fun getPopularMovies() {
    MoviesRepository.getPopularMovies(
        popularMoviesPage,
        ::onPopularMoviesFetched,
        ::onError
    )
}
```

5. Use getPopularMovies() in onCreate().

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    popularMovies.layoutManager = popularMoviesLayoutMgr
    popularMoviesAdapter = MoviesAdapter(mutableListOf())
    popularMovies.adapter = popularMoviesAdapter

    getPopularMovies()
}
```

6. Create a new method called attachPopularMoviesOnScrollListener().

```
private fun attachPopularMoviesOnScrollListener() {
    popularMovies.addOnScrollListener(object : RecyclerView.OnScrollListener() {
        override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
            val totalItemCount = popularMoviesLayoutMgr.itemCount
```

```
            val visibleItemCount = popularMoviesLayoutMgr.childCount
            val firstVisibleItem = popularMoviesLayoutMgr.findFirstVisibleItemPosition()

            if (firstVisibleItem + visibleItemCount >= totalItemCount / 2) {
                popularMovies.removeOnScrollListener(this)
                popularMoviesPage++
                getPopularMovies()
            }
        }
    })
}
```

Let's go over the code bit by bit. The first three variables are:

- totalItemCount - the total number of movies inside our popularMoviesAdapter. This will keep increasing the more we call popularMoviesAdapter.appendMovies().
- visibleItemCount - the current number of child views attached to the RecyclerView that are currently being recycled over and over again. The value of this variable for common screen sizes will range roughly around 4-5 which are 3 visible views, +1 left view that's not seen yet and +1 right view that's not seen yet also. The value will be higher if you have a bigger screen.
- firstVisibleItem - is the position of the leftmost visible item in our list.

The condition will be true if the user has scrolled past halfway plus a buffered value of visibleItemCount.

```
if (firstVisibleItem + visibleItemCount >= totalItemCount / 2) {
    ---
}
```

After condition is met, we disable the scroll listener since we only want this code to run once. Next, we increment popularMoviesPage and then call getPopularMovies().

```
if (firstVisibleItem + visibleItemCount >= totalItemCount / 2) {
    popularMovies.removeOnScrollListener(this)
    popularMoviesPage++
    getPopularMovies()
}
```

7. In onPopularMoviesFetched() method, call the newly renamed appendMovies() method and reattach the OnScrollListener again.

```
private fun onPopularMoviesFetched(movies: List<Movie>) {
    popularMoviesAdapter.appendMovies(movies)
    attachPopularMoviesOnScrollListener()
}
```

When the user has scrolled past halfway, detach the OnScrollListener and then after the new movies have been fetched reattach it again. The complete MainActivity code should like this:

```
class MainActivity : AppCompatActivity() {

    private lateinit var popularMovies: RecyclerView
    private lateinit var popularMoviesAdapter: MoviesAdapter
    private lateinit var popularMoviesLayoutMgr: LinearLayoutManager

    private var popularMoviesPage = 1

    override fun onCreate(savedInstanceState: Bundle?) {
        ---

        popularMovies = findViewById(R.id.popular_movies)
        popularMoviesLayoutMgr = LinearLayoutManager(
            this,
            LinearLayoutManager.HORIZONTAL,
            false
        )
```

```
        popularMovies.layoutManager = popularMoviesLayoutMgr
        popularMoviesAdapter = MoviesAdapter(mutableListOf())
        popularMovies.adapter = popularMoviesAdapter

        getPopularMovies()
    }

    private fun getPopularMovies() {
        MoviesRepository.getPopularMovies(
            popularMoviesPage,
            ::onPopularMoviesFetched,
            ::onError
        )
    }

    private fun onPopularMoviesFetched(movies: List<Movie>) {
        popularMoviesAdapter.appendMovies(movies)
        attachPopularMoviesOnScrollListener()
    }

    private fun attachPopularMoviesOnScrollListener() {
        popularMovies.addOnScrollListener(object : RecyclerView.OnScrollListener() {
            override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
                val totalItemCount = popularMoviesLayoutMgr.itemCount
                val visibleItemCount = popularMoviesLayoutMgr.childCount
                val firstVisibleItem = popularMoviesLayoutMgr.findFirstVisibleItemPosition()

                if (firstVisibleItem + visibleItemCount >= totalItemCount / 2) {
                    popularMovies.removeOnScrollListener(this)
                    popularMoviesPage++
                    getPopularMovies()
                }
            }
        })
    }

    ---
}
```

8. Run the app. Keep scrolling and you'll notice that it now fetches a new batch of movies. Good job!

   If you want to know why we detach and reattach a scroll listener, comment out popularMovies.removeOnScrollListener(this) and replace it with Log.d("MainActivity", "Fetching movies") and you'll see how many times you're fetching the movies.

Feature #2: As a user, I want the list of movies to be categorized by Popular, Top Rated and Upcoming

This chapter will be quite easy. We will just replicate what we did for popular movies in the previous chapter for top rated and upcoming movies.

Show Top Rated Movies

1. Open your **Api** interface and let's add a new endpoint for fetching top rated movies.

```
interface Api {

    ---

    @GET("movie/top_rated")
```

```
    fun getTopRatedMovies(
        @Query("api_key") apiKey: String = "YOUR_API_KEY_HERE",
        @Query("page") page: Int
    ): Call<GetMoviesResponse>
}
```

As you can see, getPopularMovies() and getTopRatedMovies() are basically the same. The only difference is that getTopRatedMovies() has a different endpoint - @GET("movie/top_rated").

2. Open your **MoviesRepository** and add a new method called getTopRatedMovies().

```
object MoviesRepository {

    ...

    fun getTopRatedMovies(
        page: Int = 1,
        onSuccess: (movies: List<Movie>) -> Unit,
        onError: () -> Unit
    ) {
        api.getTopRatedMovies(page = page)
            .enqueue(object : Callback<GetMoviesResponse> {
                override fun onResponse(
                    call: Call<GetMoviesResponse>,
                    response: Response<GetMoviesResponse>
                ) {
                    if (response.isSuccessful) {
                        val responseBody = response.body()

                        if (responseBody != null) {
                            onSuccess.invoke(responseBody.movies)
                        } else {
                            onError.invoke()
                        }
                    } else {
                        onError.invoke()
                    }
                }

                override fun onFailure(call: Call<GetMoviesResponse>, t: Throwable) {
                    onError.invoke()
                }
            })
    }
}
```

3. Open your **activity_main.xml** and add a new RecyclerView for top rated movies.

```
<androidx.core.widget.NestedScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        ...

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            android:text="@string/top_rated"
            android:textColor="@android:color/white"
            android:textSize="18sp"
            android:textStyle="bold"   />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginEnd="16dp"
            android:text="@string/highest_rated_movies"   />

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/top_rated_movies"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:clipToPadding="false"
            android:paddingStart="16dp"
            android:paddingEnd="16dp" />

    </LinearLayout>

</androidx.core.widget.NestedScrollView>
```

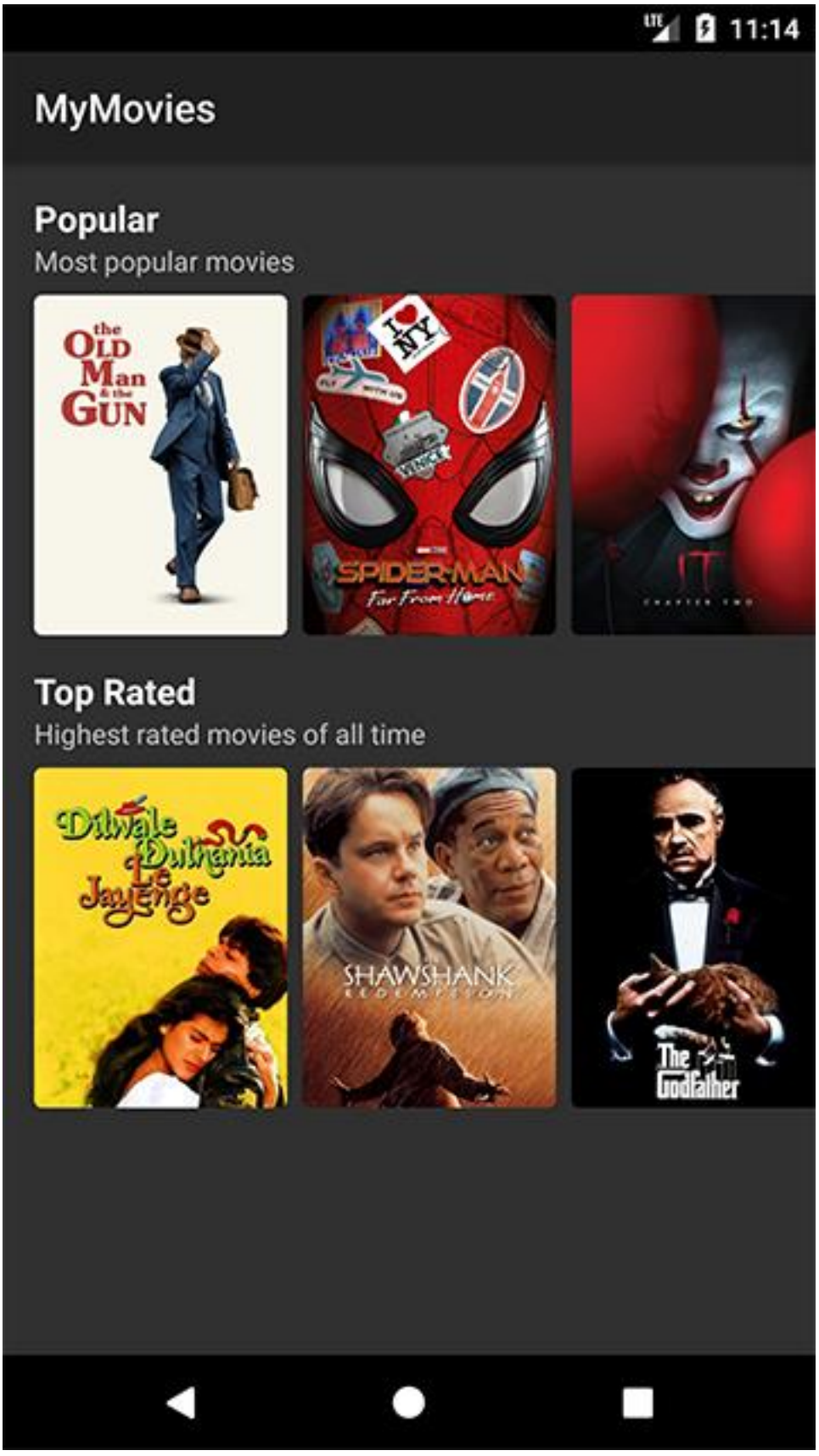
Be sure to wrap your LinearLayout with NestedScrollView. You will use NestedScrollView if it has scrollable views as its children such as a RecyclerView as it will take care of handling nested scrolling for you. If you don't have scrollable views as children, a normal ScrollView will do.

4. Open your **MainActivity** and populate your top rated movies list.

```
class MainActivity : AppCompatActivity() {

    ...

    private lateinit var topRatedMovies: RecyclerView
    private lateinit var topRatedMoviesAdapter: MoviesAdapter
    private lateinit var topRatedMoviesLayoutMgr: LinearLayoutManager

    private var topRatedMoviesPage = 1

    override fun onCreate(savedInstanceState: Bundle?) {
        ...

        topRatedMovies = findViewById(R.id.top_rated_movies)
        topRatedMoviesLayoutMgr = LinearLayoutManager(
            this,
            LinearLayoutManager.HORIZONTAL,
            false
        )
        topRatedMovies.layoutManager = topRatedMoviesLayoutMgr
        topRatedMoviesAdapter = MoviesAdapter(mutableListOf())
        topRatedMovies.adapter = topRatedMoviesAdapter

        getPopularMovies()
        getTopRatedMovies()
    }
```

```
    ...

    private fun getTopRatedMovies() {
        MoviesRepository.getTopRatedMovies(
            topRatedMoviesPage,
            ::onTopRatedMoviesFetched,
            ::onError
        )
    }

    private fun attachTopRatedMoviesOnScrollListener() {
        topRatedMovies.addOnScrollListener(object : RecyclerView.OnScrollListener() {
            override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
                val totalItemCount = topRatedMoviesLayoutMgr.itemCount
                val visibleItemCount = topRatedMoviesLayoutMgr.childCount
                val firstVisibleItem = topRatedMoviesLayoutMgr.findFirstVisibleItemPosition()

                if (firstVisibleItem + visibleItemCount >= totalItemCount / 2) {
                    topRatedMovies.removeOnScrollListener(this)
                    topRatedMoviesPage++
                    getTopRatedMovies()
                }
            }
        })
    }

    private fun onTopRatedMoviesFetched(movies: List<Movie>) {
        topRatedMoviesAdapter.appendMovies(movies)
        attachTopRatedMoviesOnScrollListener()
    }

    ...
}
```

5. Run the app and now you can see a list of top rated movies!



Show Upcoming Movies

Next up is to show a list of upcoming movies and you've probably guessed it, the process will be the same. I suggest doing it on your own first by referencing what you did in the previous section. Then, come back to this section to compare and double check your code.

1. Open your **Api** interface and add a new endpoint for upcoming movies.

```
interface Api {

    ...

    @GET("movie/upcoming")
    fun getUpcomingMovies(
        @Query("api_key") apiKey: String = "YOUR_API_KEY_HERE",
        @Query("page") page: Int
    ): Call<GetMoviesResponse>
}
```

2. Open your **MoviesRepository** and add a new method called getUpcomingMovies().

```
object MoviesRepository {

    ...

    fun getUpcomingMovies(
        page: Int = 1,
        onSuccess: (movies: List<Movie>) -> Unit,
        onError: () -> Unit
    ) {
        api.getUpcomingMovies(page = page)
            .enqueue(object : Callback<GetMoviesResponse> {
                override fun onResponse(
                    call: Call<GetMoviesResponse>,
                    response: Response<GetMoviesResponse>
                ) {
                    if (response.isSuccessful) {
                        val responseBody = response.body()

                        if (responseBody != null) {
                            onSuccess.invoke(responseBody.movies)
                        } else {
                            onError.invoke()
                        }
                    } else {
                        onError.invoke()
                    }
                }

                override fun onFailure(call: Call<GetMoviesResponse>, t: Throwable) {
                    onError.invoke()
                }
            })
    }
}
```

3. Open your **activity_main.xml** and add a new RecyclerView for upcoming movies.

```
<androidx.core.widget.NestedScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        ...
```

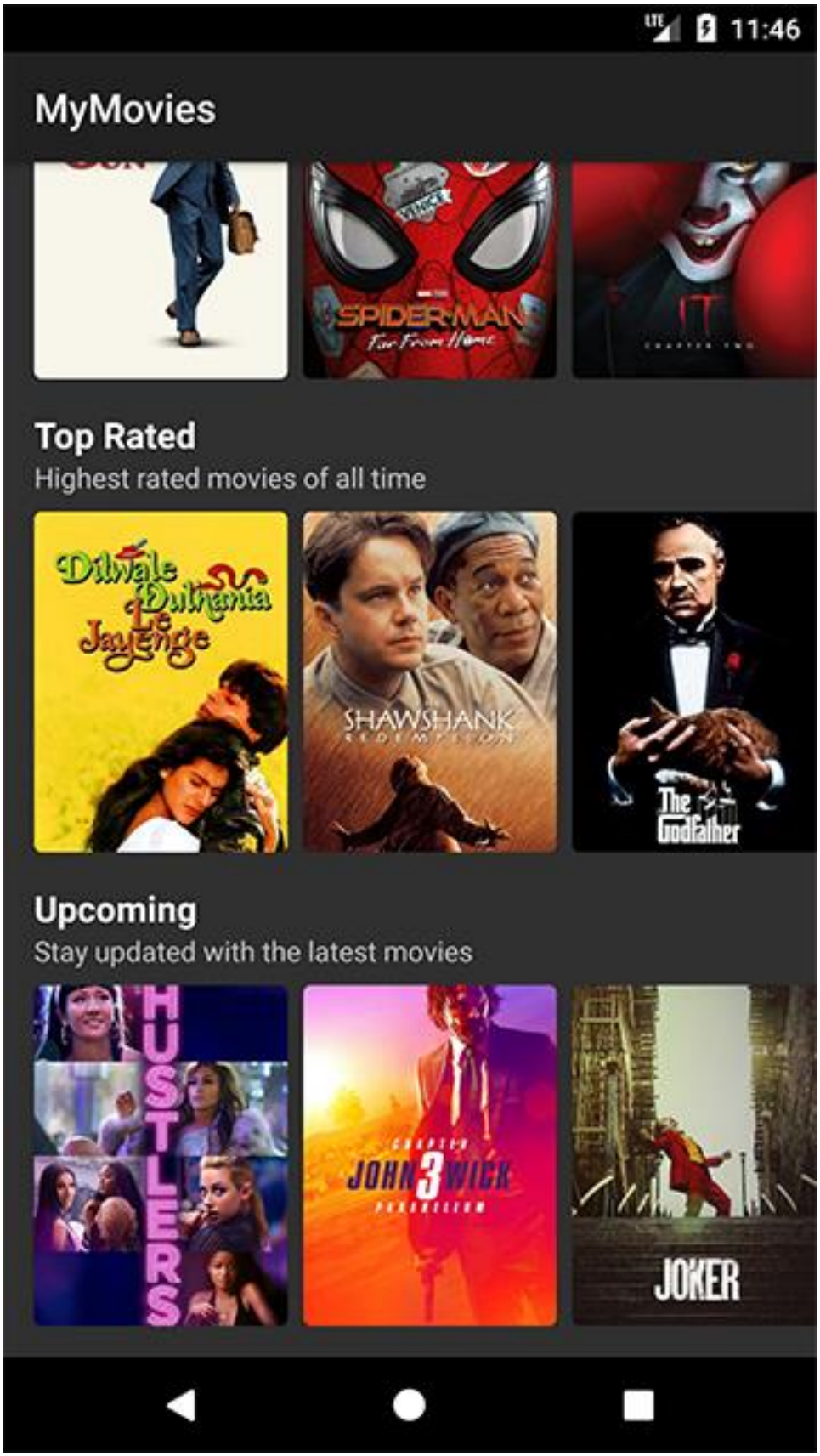
```
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            android:text="@string/upcoming"
            android:textColor="@android:color/white"
            android:textSize="18sp"
            android:textStyle="bold"  />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginEnd="16dp"
            android:text="@string/stay_updated"  />

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/upcoming_movies"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:layout_marginBottom="16dp"
            android:clipToPadding="false"
            android:paddingStart="16dp"
            android:paddingEnd="16dp" />

    </LinearLayout>

</androidx.core.widget.NestedScrollView>
```

4. Open your **MainActivity** and populate your upcoming movies list.

```
class MainActivity : AppCompatActivity() {

    ...

    private lateinit var upcomingMovies: RecyclerView
    private lateinit var upcomingMoviesAdapter: MoviesAdapter
    private lateinit var upcomingMoviesLayoutMgr: LinearLayoutManager

    private var upcomingMoviesPage = 1

    override fun onCreate(savedInstanceState: Bundle?) {
        ...

        upcomingMovies = findViewById(R.id.upcoming_movies)
        upcomingMoviesLayoutMgr = LinearLayoutManager(
            this,
            LinearLayoutManager.HORIZONTAL,
            false
        )
        upcomingMovies.layoutManager = upcomingMoviesLayoutMgr
        upcomingMoviesAdapter = MoviesAdapter(mutableListOf())
        upcomingMovies.adapter = upcomingMoviesAdapter

        getPopularMovies()
        getTopRatedMovies()
        getUpcomingMovies()
```

```
    }

    ...

    private fun getUpcomingMovies() {
        MoviesRepository.getUpcomingMovies(
            upcomingMoviesPage,
            ::onUpcomingMoviesFetched,
            ::onError
        )
    }

    private fun attachUpcomingMoviesOnScrollListener() {
        upcomingMovies.addOnScrollListener(object : RecyclerView.OnScrollListener() {
            override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
                val totalItemCount = upcomingMoviesLayoutMgr.itemCount
                val visibleItemCount = upcomingMoviesLayoutMgr.childCount
                val firstVisibleItem = upcomingMoviesLayoutMgr.findFirstVisibleItemPosition()

                if (firstVisibleItem + visibleItemCount >= totalItemCount / 2) {
                    upcomingMovies.removeOnScrollListener(this)
                    upcomingMoviesPage++
                    getUpcomingMovies()
                }
            }
        })
    }

    private fun onUpcomingMoviesFetched(movies: List<Movie>) {
        upcomingMoviesAdapter.appendMovies(movies)
        attachUpcomingMoviesOnScrollListener()
    }

    ...
}
```

5. Run the app and enjoy scrolling through the list of different movies.



Before we proceed to the next chapter, I want you to pause, take a moment, and realize you've just made a fully functional app! Gone were the days where you were just stuck in tutorials after tutorials without producing something tangible and functional that users can actually use.

Feature #3: As a user, I want to be able to see the details of a movie

Movie Details Screen

1. Right click **mymovies** package, select **New -> Activity -> Empty Activity**.
2. Enter the activity name - **MovieDetailsActivity**.
3. Open **activity_movie_details.xml** and let's build our UI.

```
<androidx.core.widget.NestedScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fillViewport="true">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <ImageView
            android:id="@+id/movie_backdrop"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintBottom_toBottomOf="@+id/backdrop_guideline"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <androidx.cardview.widget.CardView
            android:id="@+id/movie_poster_card"
            android:layout_width="128dp"
            android:layout_height="172dp"
            android:layout_marginStart="16dp"
            android:layout_marginEnd="8dp"
            app:cardCornerRadius="4dp"
            app:layout_constraintBottom_toBottomOf="@+id/backdrop_guideline"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@id/backdrop_guideline">

            <ImageView
                android:id="@+id/movie_poster"
                android:layout_width="match_parent"
                android:layout_height="match_parent"  />

        </androidx.cardview.widget.CardView>

        <androidx.constraintlayout.widget.Guideline
            android:id="@+id/backdrop_guideline"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            app:layout_constraintGuide_percent="0.4"   />

        <TextView
            android:id="@+id/movie_title"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            android:textColor="@android:color/white"
            android:textSize="18sp"
```

```
            android:textStyle="bold"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toEndOf="@+id/movie_poster_card"
            app:layout_constraintTop_toBottomOf="@+id/backdrop_guideline"   />

        <TextView
            android:id="@+id/movie_release_date"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#757575"
            android:textSize="12sp"
            app:layout_constraintStart_toStartOf="@+id/movie_title"
            app:layout_constraintTop_toBottomOf="@+id/movie_title"  />

        <androidx.constraintlayout.widget.Barrier
            android:id="@+id/movie_poster_title_barrier"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:barrierDirection="bottom"
            app:constraint_referenced_ids="movie_rating,movie_release_date"    />

        <TextView
            android:id="@+id/movie_overview"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/movie_poster_title_barrier" />

        <RatingBar
            android:id="@+id/movie_rating"
            style="@style/Widget.AppCompat.RatingBar.Small"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            app:layout_constraintEnd_toEndOf="@+id/movie_poster_card"
            app:layout_constraintStart_toStartOf="@+id/movie_poster_card"
            app:layout_constraintTop_toBottomOf="@+id/movie_poster_card"  />
    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.core.widget.NestedScrollView>
```

4. Open your **MovieDetailsActivity** and instantiate the views.

```
class MovieDetailsActivity : AppCompatActivity() {

    private lateinit var backdrop: ImageView
    private lateinit var poster: ImageView
    private lateinit var  title: TextView
    private lateinit var rating: RatingBar
    private lateinit var releaseDate: TextView
    private lateinit var overview: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_movie_details)

        backdrop  =  findViewById(R.id.movie_backdrop)
```

```
        poster = findViewById(R.id.movie_poster)
        title = findViewById(R.id.movie_title)
        rating = findViewById(R.id.movie_rating)
        releaseDate = findViewById(R.id.movie_release_date)
        overview = findViewById(R.id.movie_overview)
    }
}
```

5. Just above the class name, add your intent variables.

```
const val MOVIE_BACKDROP = "extra_movie_backdrop"
const val MOVIE_POSTER = "extra_movie_poster"
const val MOVIE_TITLE = "extra_movie_title"
const val MOVIE_RATING = "extra_movie_rating"
const val MOVIE_RELEASE_DATE = "extra_movie_release_date"
const val MOVIE_OVERVIEW = "extra_movie_overview"

class MovieDetailsActivity : AppCompatActivity() { ... }
```

These variables will be used as keys when we pass intent extras to MovieDetailsActivity.

6. Use the keys above to populate the movie's details.

```
class MovieDetailsActivity : AppCompatActivity() {
    ...

    override fun onCreate(savedInstanceState: Bundle?) {
        ...

        val extras = intent.extras

        if (extras != null) {
            populateDetails(extras)
        } else {
            finish()
        }
    }

    private fun populateDetails(extras: Bundle) {
        extras.getString(MOVIE_BACKDROP)?.let { backdropPath ->
            Glide.with(this)
                .load("https://image.tmdb.org/t/p/w1280$backdropPath")
                .transform(CenterCrop())
                .into(backdrop)
        }

        extras.getString(MOVIE_POSTER)?.let { posterPath ->
            Glide.with(this)
                .load("https://image.tmdb.org/t/p/w342$posterPath")
                .transform(CenterCrop())
                .into(poster)
        }

        title.text = extras.getString(MOVIE_TITLE, "")
        rating.rating = extras.getFloat(MOVIE_RATING, 0f) / 2
        releaseDate.text = extras.getString(MOVIE_RELEASE_DATE, "")
        overview.text = extras.getString(MOVIE_OVERVIEW, "")
    }
}
```

The available backdrop sizes are:

- w300
- w780
- w1280
- original

Open Movie Details Screen from Movie List Screen

1. Open your **MainActivity** and create a new method called showMovieDetails().

```
class MainActivity : AppCompatActivity() {
    ---
    private fun showMovieDetails(movie: Movie) {
        val intent = Intent(this, MovieDetailsActivity::class.java)
        intent.putExtra(MOVIE_BACKDROP, movie.backdropPath)
        intent.putExtra(MOVIE_POSTER, movie.posterPath)
        intent.putExtra(MOVIE_TITLE, movie.title)
        intent.putExtra(MOVIE_RATING, movie.rating)
        intent.putExtra(MOVIE_RELEASE_DATE, movie.releaseDate)
        intent.putExtra(MOVIE_OVERVIEW, movie.overview)
        startActivity(intent)
    }
    ---
}
```

2. Open your **MoviesAdapter** and add a new parameter in the constructor which is a higher-order function that will be called when a movie is clicked.

```
class MoviesAdapter(
    private var movies: MutableList<Movie>,
    private val onMovieClick: (movie: Movie) -> Unit
) : ---
```

3. Invoke onMovieClick when a movie is clicked.

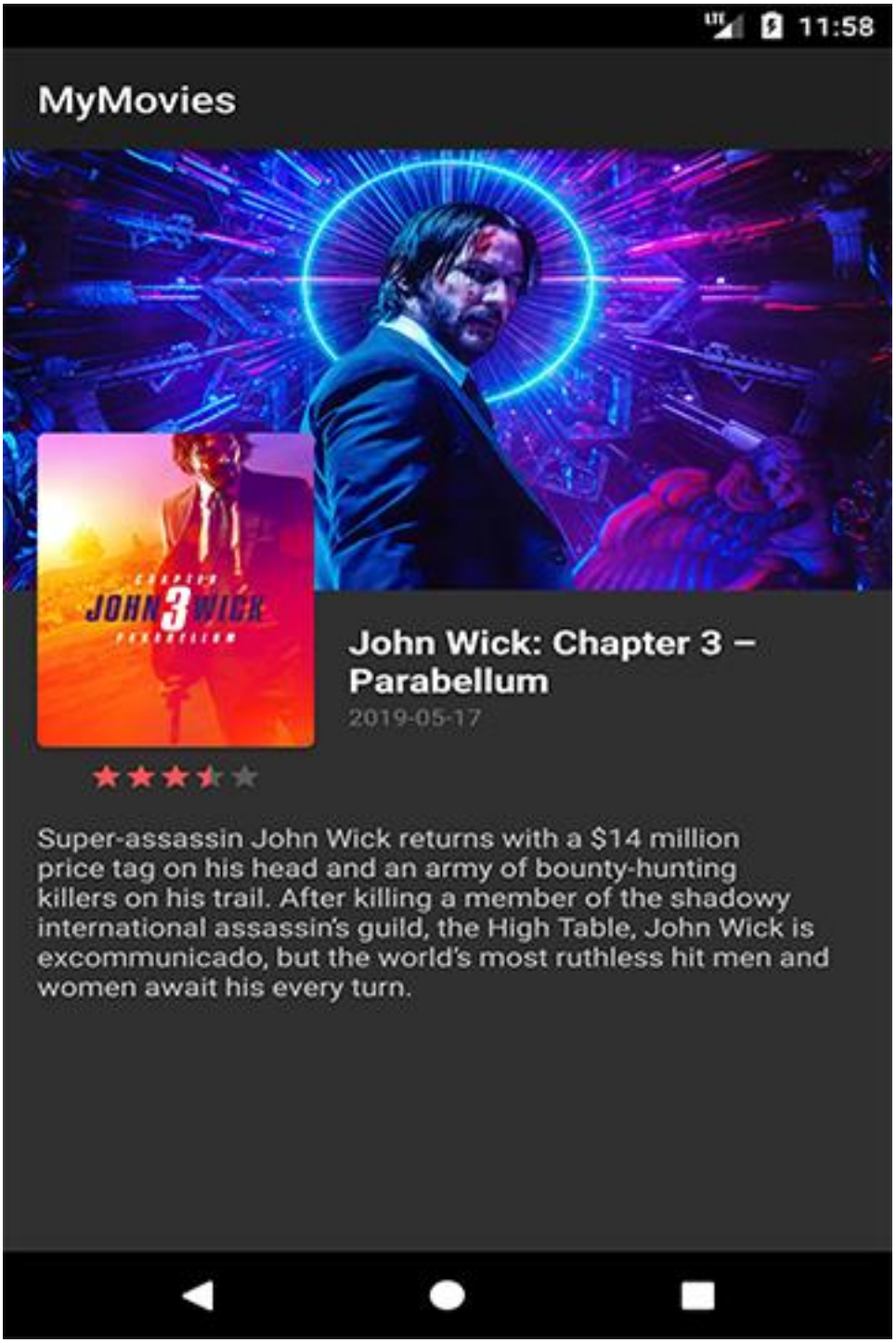
```
class MoviesAdapter(
    private var movies: MutableList<Movie>,
    private val onMovieClick: (movie: Movie) -> Unit
) : RecyclerView.Adapter<MoviesAdapter.MovieViewHolder>() {

    ---

    inner class MovieViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        ---
        fun bind(movie: Movie) {
            ---
            itemView.setOnClickListener { onMovieClick.invoke(movie) }
        }
    }
}
```

4. Open your **MainActivity** and pass a higher-order function to your adapters which calls the showMovieDetails() method that you've just created.

```
class MainActivity : AppCompatActivity() {

    ---

    override fun onCreate(savedInstanceState: Bundle?) {
        ---
        popularMoviesAdapter = MoviesAdapter(mutableListOf()) { movie -> showMovieDetails(movie) }
        ---

        ---
```

```
        topRatedMoviesAdapter = MoviesAdapter(mutableListOf()) { movie -> showMovieDetails(movie)    }
        ---

        ---
        upcomingMoviesAdapter = MoviesAdapter(mutableListOf()) { movie -> showMovieDetails(movie)   }
        ---
    }
    ---
}
```

5. Run the app, tap any movie and you should see something like this:



Congratulations! You have just made a full-blown and portfolio-worthy Android app that you can put on Google Play Store.