**MOVIE APP**

What will the app do? (Objectives)

1. Provide the user with a list of top trending movies
2. Allow the user to view detailed information of the movie
3. Allow the user to save and share the movie details

**A real-world example of where such this application will be useful**

With the advancement of technology, media companies have made tremendous progress over the years. A lot of channels, TV shows, and movies use these technologies to entertain their users. Here, we see the requirement of an application that can keep track of all these entertainment-related updates.

It becomes hard to keep track of the best and trending movies of a particular genre. This Android application will allow users to get updates on trending movies and save and share the same with their friends.

**Task**

Your task will be to develop an Android application in Kotlin using Android Studio. The app will contain a list of trending movies. Users can filter this list on the basis of the most popular, highest rated, and favourite categories. They can save and share the details of a particular movie in the SQLite database. Saving and fetching from the SQLite database should be done using the content provider.

You have to use [Themoviedb.org](https://www.themoviedb.org/en) to develop this application. Go through the documentation link for a better understanding. This API will provide the response in JSON type when the network call is made successfully.

**Link:** <https://www.themoviedb.org/documentation/api>

You are required to sign up and get your own API key from the following link.

**GET API KEY :** <https://developers.themoviedb.org/3/getting-started/introduction>

You are advised to use material design in your application for better UI/UX.

Material Design Guidelines :

Accessibility : <https://material.io/design/usability/accessibility.html#color-contrast>

Layouts : [https://material.io/design/layout/understanding-layout.html#](https://material.io/design/layout/understanding-layout.html)

Component used in the application:

1. RecyclerView
2. Fragments
3. SQLite
4. Content Provider
5. CardView

Libraries used

1. **Retrofit**: <https://square.github.io/retrofit/> (optional)

Retrofit is a type-safe HTTP client for Android and Java. With Retrofit, we can compose the HTTP connection easily through a simple expressive interface, just like an API document. Besides the elegant syntax, it is also easy to incorporate with the different library.

b) **Glide**: <https://github.com/bumptech/glide>

Glide is a fast and efficient open source media management and image loading

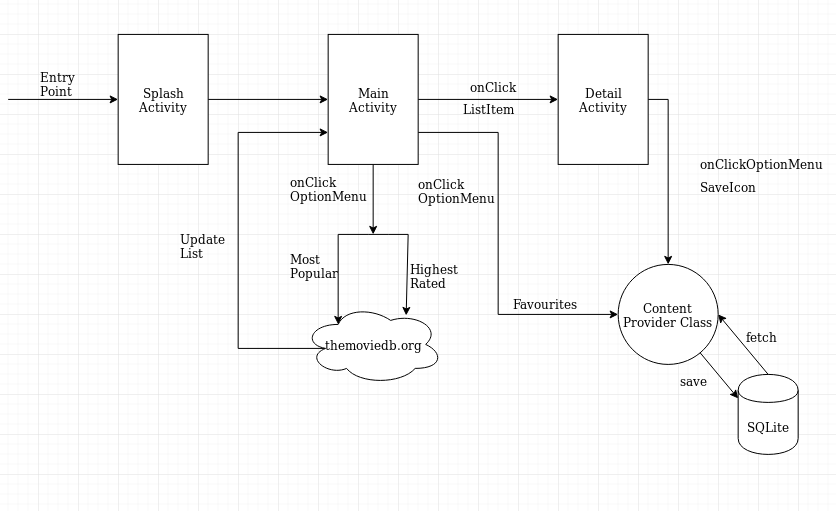
framework for Android that wraps media decoding, memory and disk caching, and

resource pooling into a simple and easy-to-use interface.

The app will contain three Activities (screens):

1. SplashActivity
2. MainActivity
3. DetailActivity

The following diagram depicts the flow of the application through activities:

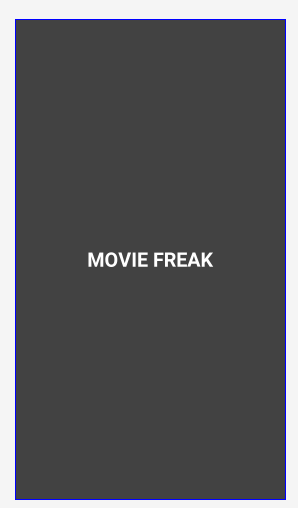


\*\***All the points provided below have to be covered for the respective activities in the Android application.**

**SplashActivity :**

1. The SplashActivity should always open first at the time of the app launch.
2. It should contain the name of the app in the custom font, centre-aligned on the screen.
3. After 2-3 seconds, the *MainActivity* should open.

**Fig.1 SplashActivity**

****

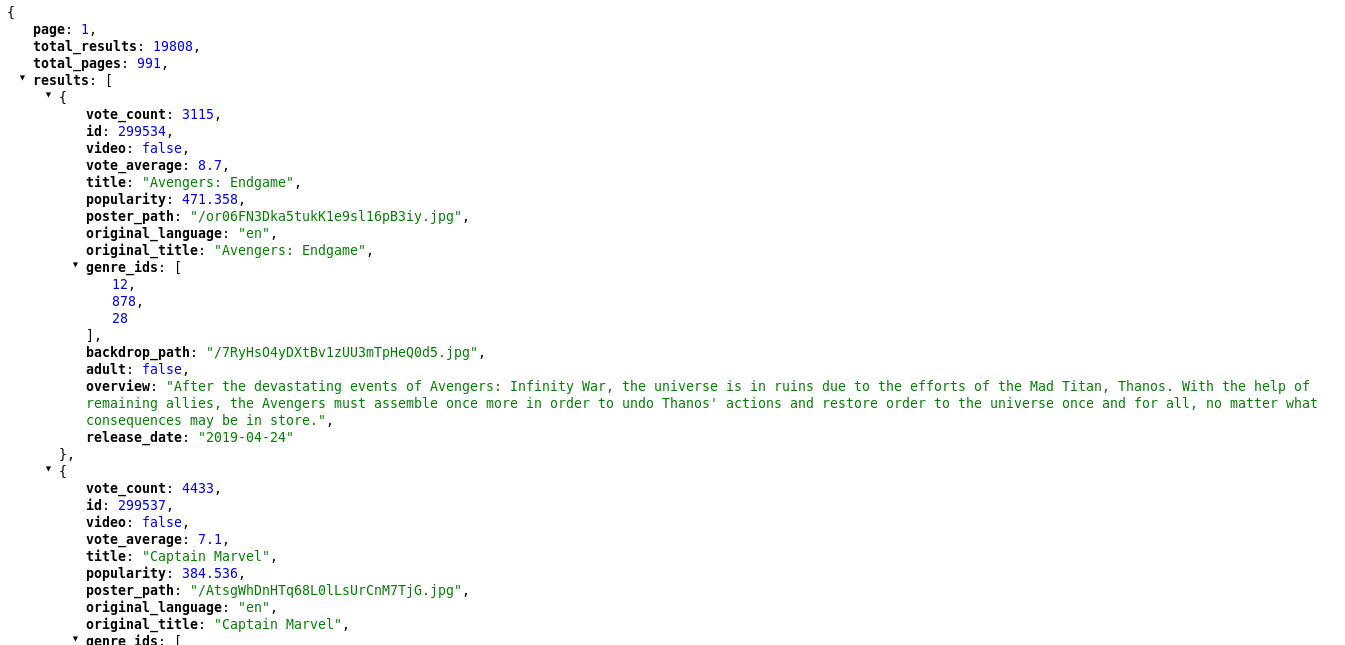
**MainActivity :**

1. When the user enters the MainActivity, the most popular movies should be fetched from the API by default.
2. As shown in the screenshot (**Fig1**), the list item has to be designed accordingly.
3. The action bar of the *MainActivity* should have “Movies App” as the title.
4. In the top-right corner, there is an OptionMenu, which shows three items as shown in **Fig2**:
   1. **Most Popular**: By default, this option is selected. When this is clicked on, the app should fetch the list of the most popular movies.

Doc link**:** <https://developers.themoviedb.org/3/movies/get-popular-movies>

API link**:** <https://api.themoviedb.org/3/movie/popular?region=in&page=1&language=en-US&api_key=82813177051f3dab7ef745864fbfb36c>

Response:



As per the JSON response, ‘movies’ response is in the “results” key of JSON as an array. Each list item contains the name, description, poster path, release date, etc. of the movie. To get the link of a poster of the movie, you need to create a URL using the following code snippet:

***"http://image.tmdb.org/t/p/w"*** *+ Integer.toString(width) + fileName*

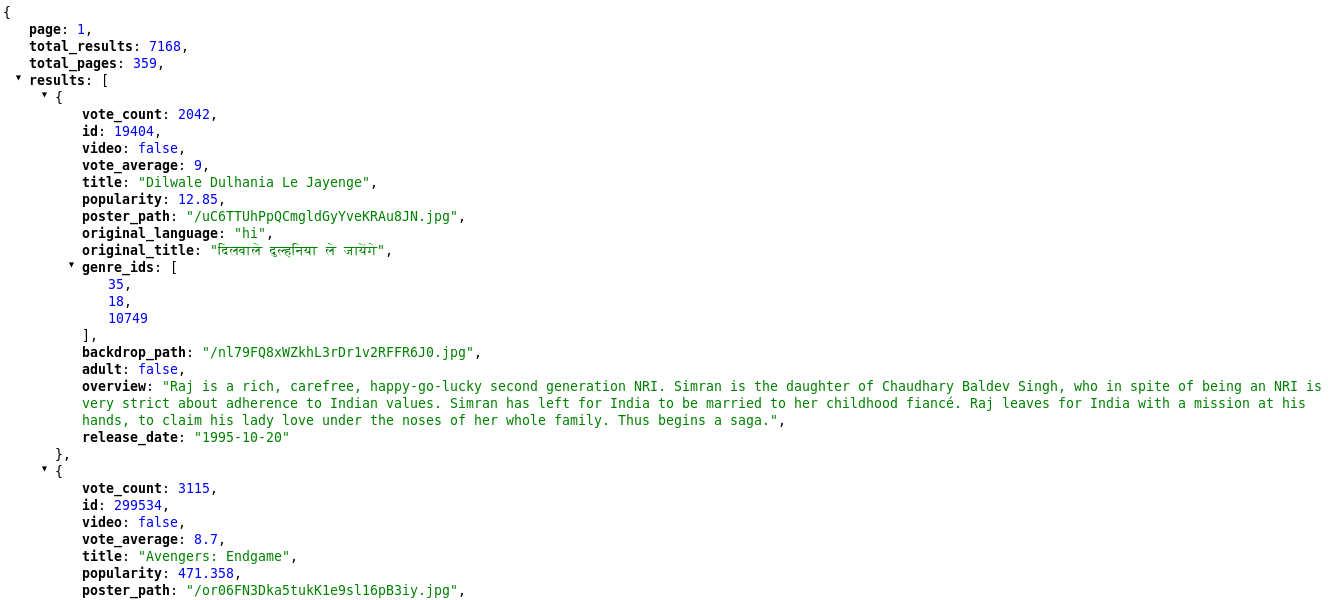
Here, the width of the image is in pixels and the *fileName* is the value of the *poster\_path or backdrop\_path* keys from the JSON response.

ii) **Highest Rated**: When the user clicks on this option menu, the highest rated movies should be shown in the app.

Doc Link: <https://developers.themoviedb.org/3/movies/get-top-rated-movies>

API link : <https://api.themoviedb.org/3/movie/top_rated?region=in&page=1&language=en-US&api_key=82813177051f3dab7ef745864fbfb36c>

Response :



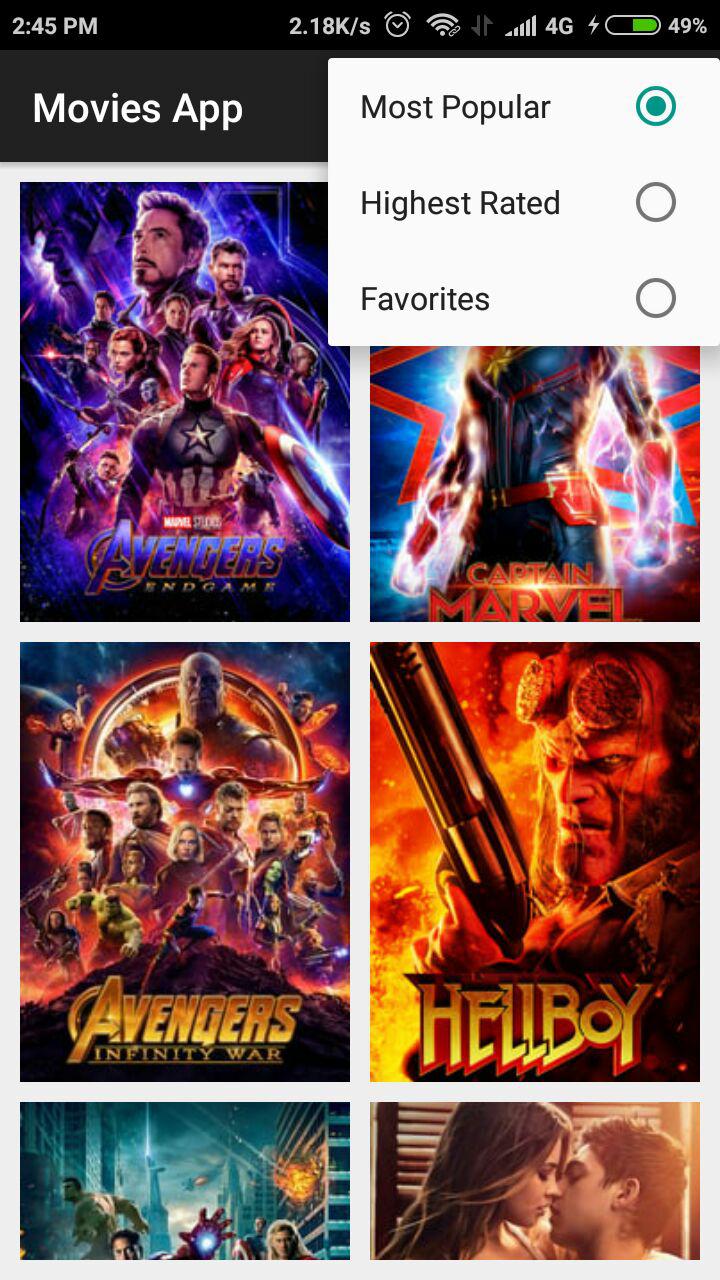
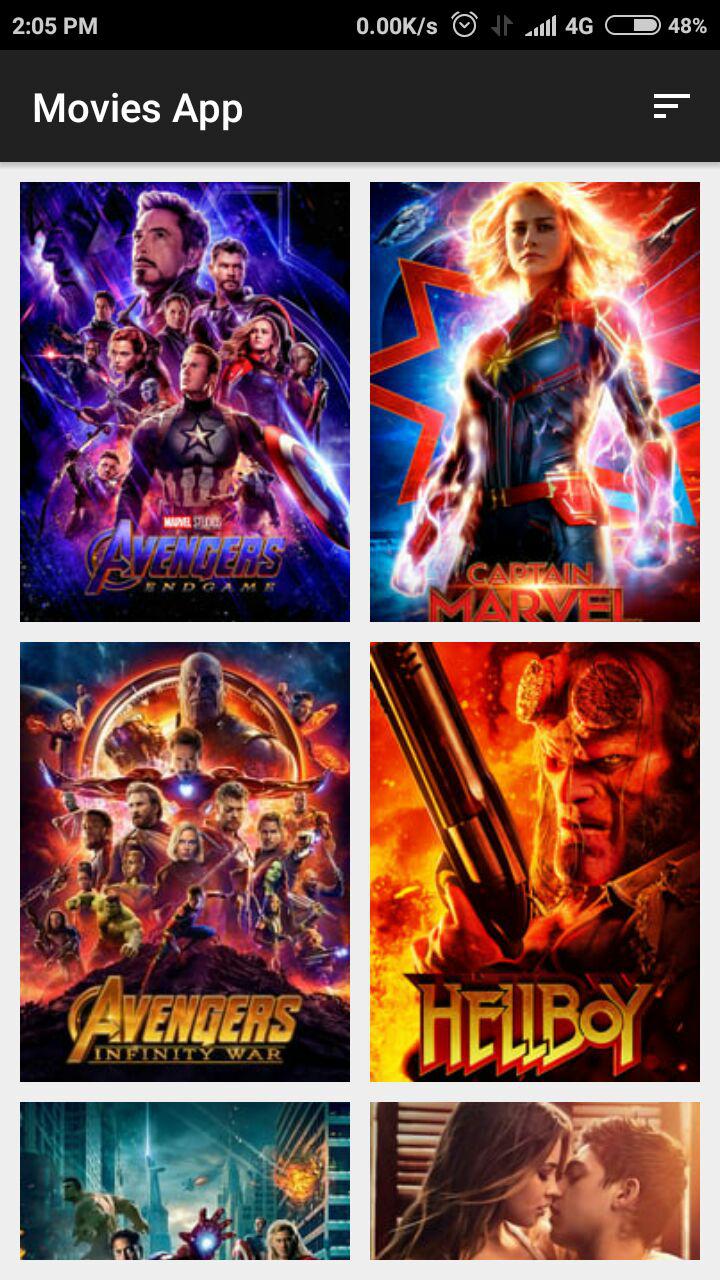
As per the JSON response, the movies response is in the “results” key of JSON as an array. Each list item contains the name, description, poster path, release date, etc. of the movie. To get a poster of the movie, you need to create a URL using the following code snippet:

***"http://image.tmdb.org/t/p/w"*** *+ Integer.toString(width) + fileName*

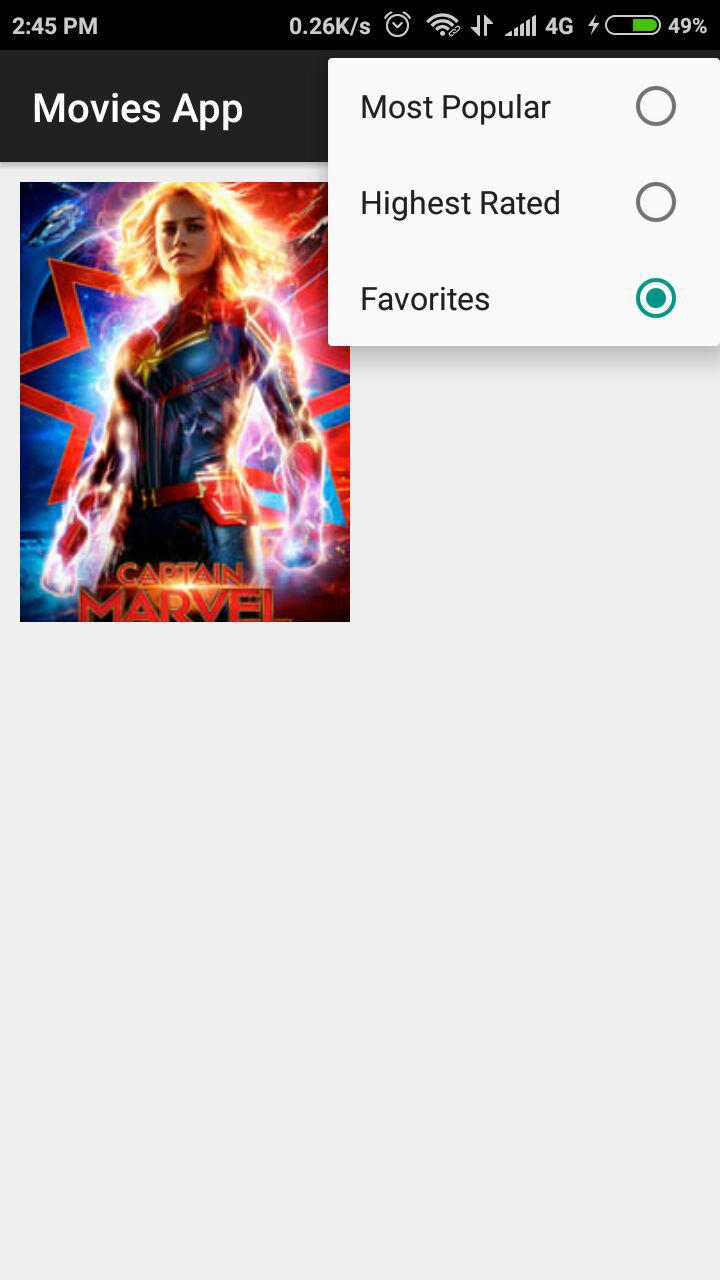
Here, the width of the image is in pixels, and the *fileName* is the value of the *poster\_path or backdrop\_path*  keys from the JSON response above.

**iii) Favourites**: When the user clicks on this option, the saved movies in the local database, i.e., SQLite, should be fetched using the content provider and displayed in the list. If there are no saved movies, show a relevant toast message stating “No item in the database”.

\*\***NOTE: You are required to use the RecyclerView to show the data in a grid view. In the OptionMenu, the user should be able to select only one option at a time.**

****

**Fig 1. Fig.2**

****

**Fig 3.**

When the user clicks on the list item of the MainAcitivty, the information of the movie should open in the DetailActivity, passed via *Intent.* The Intent should also contain the information about the movie list item that has been clicked on and then passed to the DetailActivity.

**DetailAcitivty :**

1. The DetailActivity shows the details of the movie item that has been clicked on in the MainActivity.
2. Since the data is passed to the DetailActivity using Intent, get the data from the intent and display it to the user as shown in **Fig.3.**
3. You also have to fetch the list of YouTube videos related to the movie IDs and show them in a horizontal scrolling list as shown in the screenshot **Fig 5** below. When the user clicks on the video, he/she should be redirected to the YouTube app, where the video starts playing.

API Doc: <https://developers.themoviedb.org/3/movies/get-movie-videos>

API link: <https://api.themoviedb.org/3/movie/299534/videos?api_key=82813177051f3dab7ef745864fbfb36c&language=en-US>

Response:



The list of trailer videos is in the “results” key of the JSON response. Each list item contains the ID, key, name, website, genre IDs, etc. The value of the “key” key in JSON response can be used to get the thumbnail and URL of the YouTube videos.

Example:

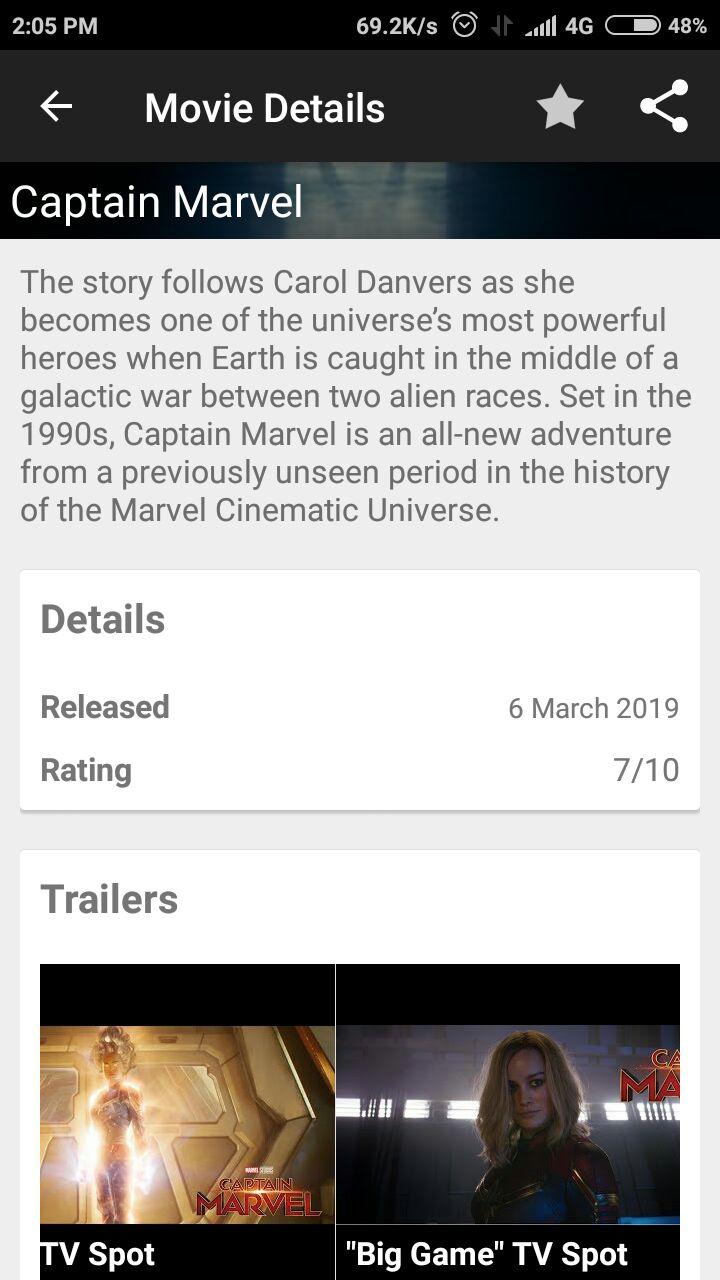
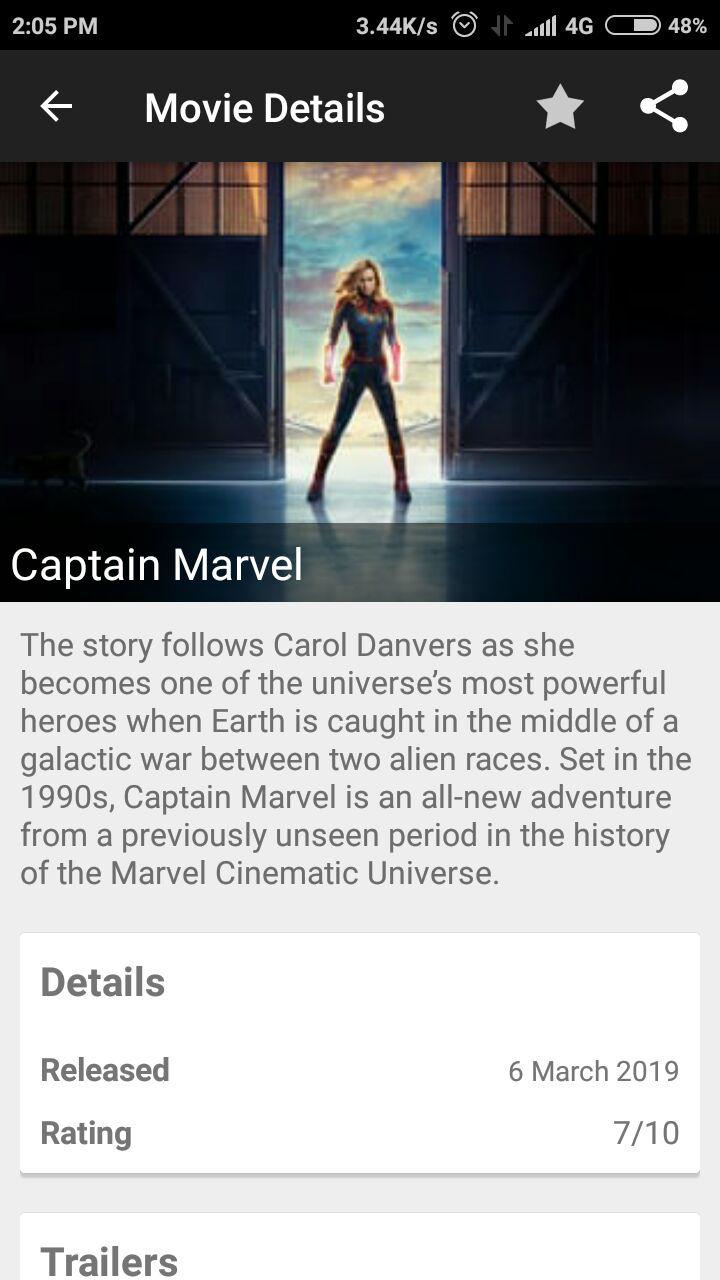
For the thumbnail: “http://img.youtube.com/vi/" + key + "/0.jpg**"**

For the video URL: “[http://www.youtube.com/watch?v="+key](http://www.youtube.com/watch?v=%22+key)

The action bar of the *DetailActivity* should contain two *OptionMenus* as icons (refer to the screenshots **Fig 4** and **Fig 5**):

1. Save (start shape icon): When the user clicks on this, the information of the current movie should be saved in the SQLite database using the content provider, and the colour of the icon should change from white to yellow.
2. Share (share shape icon): When the user clicks on the share icon, the YouTube link of the movies should be shared to other apps along with the movie name.

**Fig 4. Fig 5.**



**Evaluation Rubric**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Criteria** | | |  | | --- | | **Meets Specifications** | | |  | | --- | | **Does Not Meet Specifications** | |
| **Code Functionality (80%)** |  |  |
| Does the code work? | The Java code produces no error. | The code produces compilation errors or run-time errors when executed. |
| Are the activities styled as per the requirement? | The styling of the activities matches that of the screenshots given in the problem statement. It may not be exactly the same, but it should resemble the given structure. | The styling of the activity does not match that of the screenshots given in the problem statement. |
| Is the project made using Android Studio and is Java the programming language used? | Java is used as a primary programming language to create this project. The project loads in Android Studio without any build error or warning. The final project is able to generate APK. | Java is not used to create this project in Android Studio. The project throws a build error or warning and is not able to generate APK. |
| Does the movies information displayed in each activity follow the requirements mentioned in the tasks? | For each activity, the action bar title follows the requirements mentioned in the problem statement.  All the list item in the app display only the required data as per the screenshot.  Spacing, colour, and font are as per the material design guidelines. | The Action bar of each activity, by default, displays the name of the app on each activity.  The list items do not display appropriate data as required.  The design format does not follow the material design guidelines. |
| Does the Android application call the correct API? | The application calls the correct API when the user clicks on the list item, or it gets the related YouTube videos for a particular movie. | The application calls the wrong API wherever required. It does not follow the correct order of networking call depending on the activities. It makes unnecessary API calls from the activities. |
| Error handling done properly while making API calls ? | If the API call returns an error as per the API doc, the error message stating “No data has been received” should be shown in the toast. | If the API call returns an error, no error message is shown in the toast. |
| Handling the data after successful API call ? | If the API call is successful, the data returned by the API is correctly displayed in the respective lists of activities.  Update the list with fresh data using the appropriate method. | If the API call is successful, there is no updation of lists.  The list items are not updated efficiently or with fresh data. |
| Is the data in the list items well formatted? | The required data is shown to the user in each list item as per the screenshots of the activities provided.  Further, the list items consist of an effective UI and well-organised data. | The list items in the corresponding activities do not display sufficient list items or data to the user.  Ambiguous or unwanted data is shown in the list item of the respective lists. |
| Does the student use the SQLite database to store data? | The student uses the SQLite database to store information about movies offline. | The student does not use the SQLite database. Instead, he/she uses other means to store information about movies. |
| Is the content provider used to do all the CRUD operations on the SQLite database? | The student creates a content provider for the SQLite database to perform the CRUD operations on it. | The student directly executes raw queries on the database instance instead of using the content provider. |
| Does the student use the RecyclerView? | For each list in this app, the student uses the RecyclerView along with its custom adapter for the optimised list. | The student does not use the recyclerview. Instead, he/she uses static data to show the movies list and relevant information to the user. |
| Is the code formatted correctly and easy to read? | The code is formatted correctly; it uses the right spacing and indentation and follows the formatting guidelines laid out in the Google Material Design Style Guide.  The code contains useful comments that explain how the complicated portions of the code work.  Java objects, classes, packages, layouts, activities, or variables have proper and logical names. | The code is not formatted correctly. Extra spaces, line breaks, incorrect indentations, and bad formatting are used throughout the code.  The code does not contain any comment, or it contains poor comments that do not explain properly how the complicated portions of the code work.  Java objects, classes, packages, layouts, activities, or variables do not have proper or logical names. |
| Are all the libraries mentioned at the start used in the project? | The students use **Glide** image loading libraries to load the image in ImageView in the project.  To convert Java objects to JSON and vice versa, it is mandatory to use the **GSON** library, as it will be a plus point for the students in future for parsing large JSON data using only Java POJOs. | The students directly set the drawable to ImageView from XML or use trivial *setImageDrawable()* in Java.  The students parse the JSON data directly instead of using the library provided. |
| Does the student use Git and GitHub to conduct version control on his/her code? | The student uses Git and GitHub to conduct version control on the assignment code.  The student makes small, incremental commits.  The student writes clear and concise commit messages. | The student does not use Git or GitHub to conduct version control on the assignment code.  The student makes big commits that contain multiple features or bug fixes.  The student writes short or unclear commit messages. |