

KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

T120B169 App Development for Smart Mobile Systems

Personal Movie database

IFZm-7, Gintaras Ruočkus:

Date: 2020.09.13

Kaunas, 2020

Tables of Contents

<i>Description of Your app</i>	<i>3</i>
<i>Functionality of your app.....</i>	<i>4</i>
List of functions (adapt to your own app)	4
<i>Solution</i>	<i>5</i>
Task #1. Remove a UI component from the activity when a button is clicked.....	5
<i>Reference list.....</i>	<i>16</i>

Description of Your app

1. What type is your application/game? Application that is used for managing your movies list. (“Personal Movie database” application from list of topics)
2. Description. This application will allow people to check information about movies and tv shows. Source of information about movies is IMDb. When a person is looking for movies they will be able to choose different categories of movies, like current most popular, newest, upcoming. When a person finds an interest in a movie, they will be able to add it to my movie list. They will be able to check their movie list, where they can remove unwanted movies, mark them as watched, favorite them. Also they will be able to share a movie on their social media.

Functionality of your app

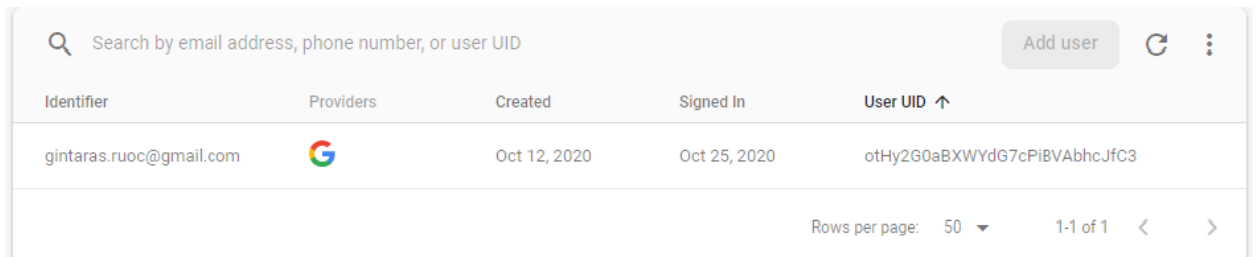
List of functions (adapt to your own app)

1. Create database.
2. Connect with google account.
3. Connect with IMDb source.
4. In home screen be able to choose different categories of movies:
 - Most popular
 - Newest
 - Upcoming
5. There should be two activities in the app (it can be the existing ones). The second activity (which is created from the first activity) has to have an input/editbox and a button. When the value “Hello” is written in the input/editbox and the button is pressed, the first activity has to change color (your choice of color but has to be different than the current one).

Solution

Task #1. Create Database

Chose to use *Firebase* as database, currently only saves users that has connected with their *google* account.



The screenshot shows the Firebase database console with a search bar at the top. Below the search bar is a table with the following columns: Identifier, Providers, Created, Signed In, and User UID. There is one row of data representing a user. At the bottom right, it shows 'Rows per page: 50' and '1-1 of 1'.


Identifier	Providers	Created	Signed In	User UID ↑
gintaras.ruoc@gmail.com		Oct 12, 2020	Oct 25, 2020	otHy2G0aBXWYdG7cPiBVAhbcJfC3

Figure 1. Firebase database, where it save google accounts

Task #2. Connect with google account

Database that is being used for this app is *Firebase*. This database has authentication function, which is able to provide ability to connect with *google* account. To be able to use you just need to follow instructions how to implement it into app [1]. After user is connected it will change menu items, which hides sign in button and shows log out and my selection buttons (Figure 2). In the Figure 3 there is a screenshot of how sign in looks like.

```
private void setVisibility(boolean type)
{
    if(type){
        signIn.setVisible(false);
        mySelection.setVisible(true);
        logOut.setVisible(true);
    }
    else {
        logOut.setVisible(false);
        mySelection.setVisible(false);
        signIn.setVisible(true);
    }
}
```

Figure 2. Changes visibility of buttons after log in or log out

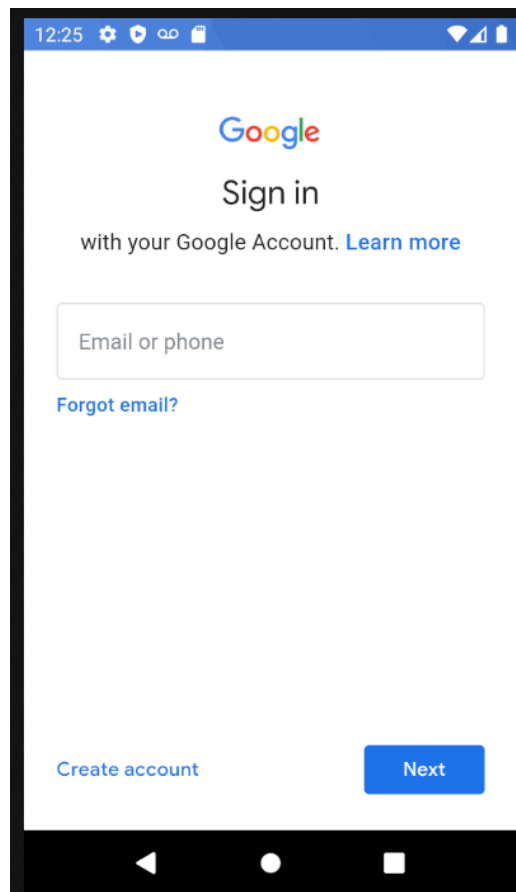


Figure 3. Goolge sign in form

Task #3. Connect with imDb source

Instead of *imDb* database I chose to use *tmDb* (the movie database) as source, because it has a bit more options of what *JSON* files I want to get. For accessing information it needs base *URL* and which service it's trying to access. Classes *Client* (Figure 4) and *Service* (Figure 5) provides *URLs* for reading *JSON*. Than there needs to be two classes, where information is stored. *Movies info* class is used for reading *JSON* file and storing information to *movie* class list, which has title, image *URL*, etc.

```
public class Client {

    public static final String BASE_URL = "https://api.themoviedb.org/3/";
    public static Retrofit retrofit = null;

    public static Retrofit getClient()
    {
        if(retrofit == null)
        {
            retrofit = new
Retrofit.Builder().baseUrl(BASE_URL).addConverterFactory(GsonConverterFactory.create(
)).build();
        }
        return retrofit;
    }
}
```

Figure 4. Information about base url

```

public interface Service {

    @GET ("movie/popular")
    Call<MoviesInfo> getPopularMovies(@Query("api_key") String apiKey, @Query("page")
String page);

    @GET ("movie/top_rated")
    Call<MoviesInfo> getTopRatedMovies(@Query("api_key") String apiKey,
@Query("page") String page);

    @GET ("movie/upcoming")
    Call<MoviesInfo> getUpcomingMovies(@Query("api_key") String apiKey,
@Query("page") String page);
}

```

Figure 5. Service class used for searching different types of sorting

Then there needs to be two classes, where information is stored. *Movies info* class (Figure 6) is used for reading *JSON* file and storing information to *movie* class (Figure 7) list, which has title, image *URL*, etc.

```

public class MoviesInfo {
    private int page;

    private List<Movie> results;

    private int totalResults;

    private int totalPages;

    public int getPage()
    {
        return page;
    }

    public void setPage(int _page)
    {
        this.page = _page;
    }

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

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

    public void setResults (List<Movie> _results)
    {
        this.results = results;
    }

    public void setMovies(List<Movie> _results)
    {
        this.results = _results;
    }

    public int getTotalResults()

```

```

    {
        return totalResults;
    }

    public void setTotalResults(int _totalResults)
    {
        this.totalResults = _totalResults;
    }

    public int getTotalPages()
    {
        return totalPages;
    }

    public void setTotalPages(int _totalPages)
    {
        this.totalPages = _totalPages;
    }
}

```

Figure 6. Used to in reading *JSON* file and storing movie info in List

```

public class Movie {
    @SerializedName("id")
    private String id;
    @SerializedName("title")
    private String title;
    @SerializedName("poster_path")
    private String imageUrl;
    @SerializedName("vote_average")
    private String rating;
    @SerializedName("vote_count")
    private String voted;
    @SerializedName("release_date")
    private String releaseDate;
    private List<String> genres;
    @SerializedName("overview")
    private String overview;

    public Movie()
    {
    }

    public Movie(String id, String title, String imageUrl, String rating, String
voted, String releaseDate, List<String> genres, String overview)
    {
        this.id = id;
        this.title = title;
        this.imageUrl = imageUrl;
        this.rating = rating;
        this.voted = voted;
        this.releaseDate = releaseDate;
        this.genres = genres;
        this.overview = overview;
    }
}

```

Figure 7. *Movie* class used to store information about movie

To display information from *JSON* there are three *XML* files *activity_main*, *movie_row_item* and *load_more_items*. *Activity_main* holds *recyclerView* component, which stores other two types of *XML* files (Figure 8). With *movie_row_item* it displays one movie information (Figure 9).

Load_more_items is XML file for displaying button, which adds more movies to *recyclerView* (Figure 10).

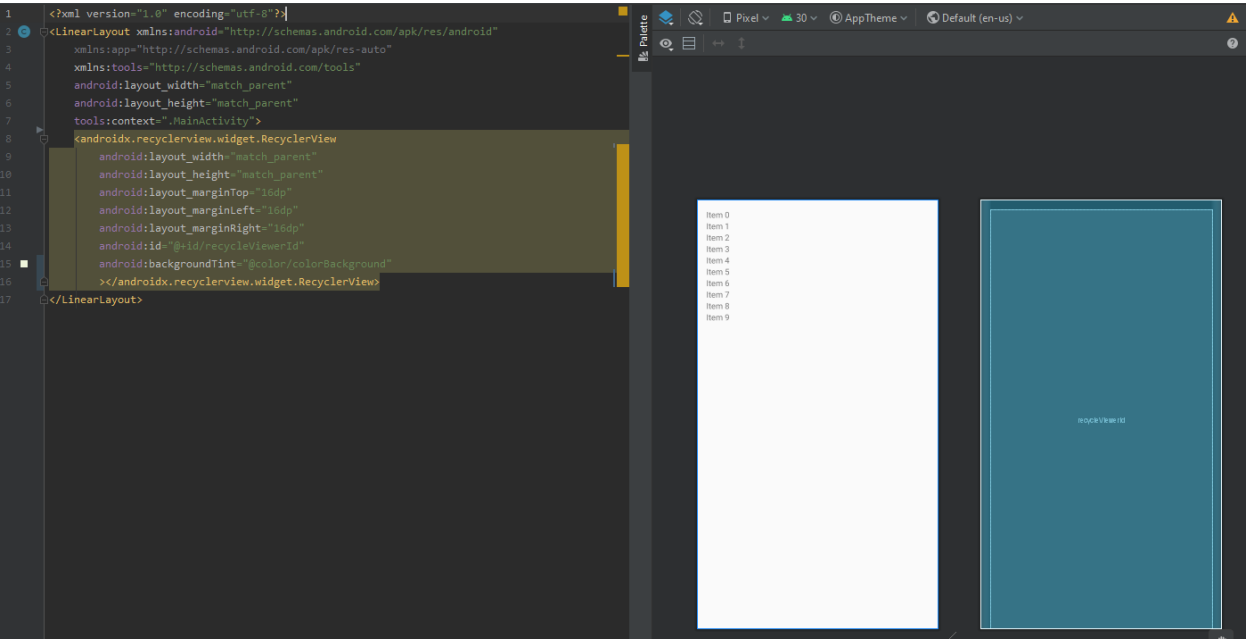


Figure 8. *Activity_main.XML* file

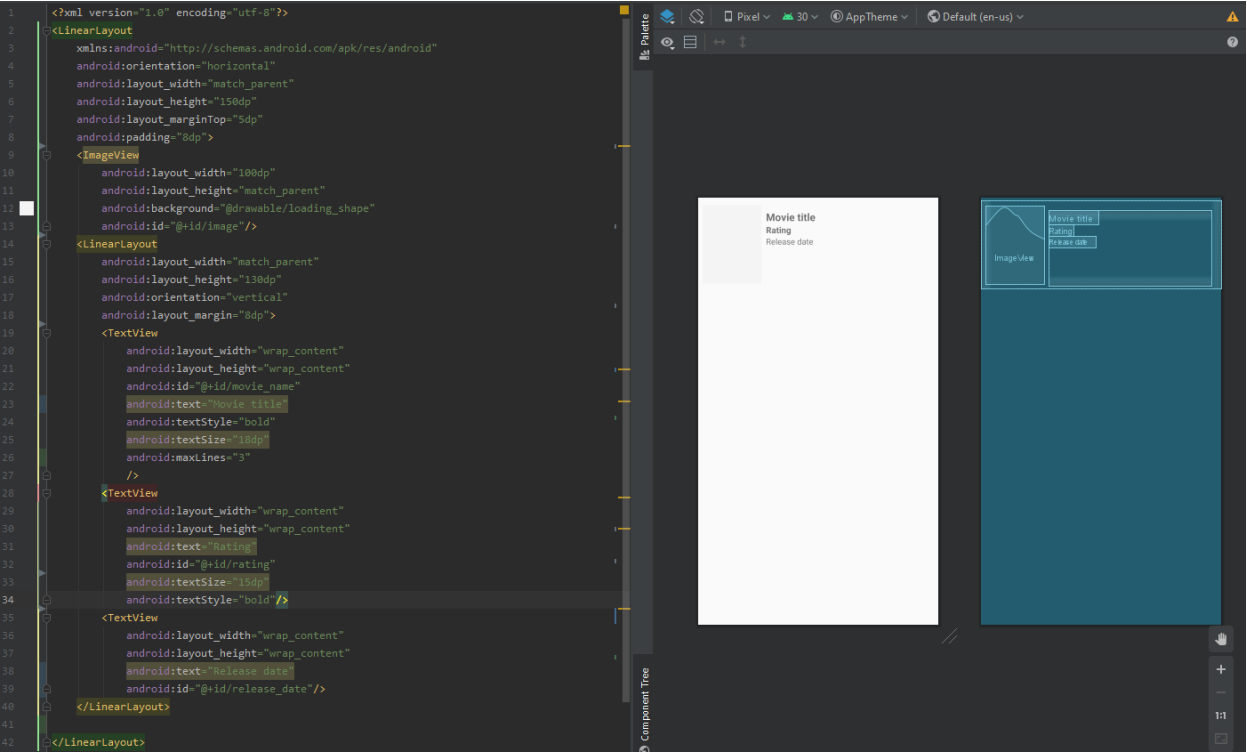


Figure 9. *Movie_row_items.XML* file

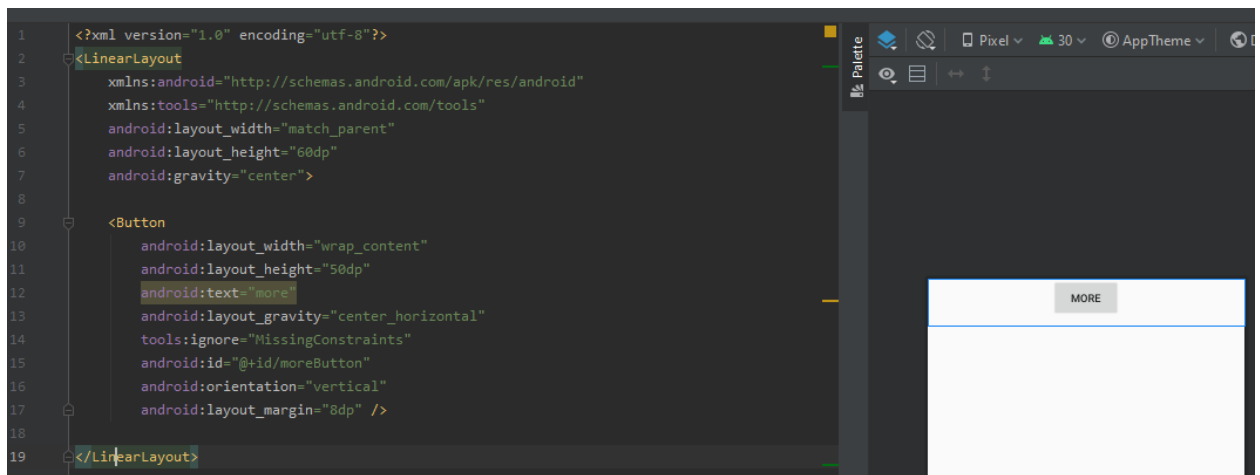


Figure 10. *load_more_items.XML* file

To display all information into *recyclerView* it needs custom adapter, which decides if it needs to show button or movie information (Figure 11). When creating button it adds *setOnClickListener*, which goes to function to add more movies information. If it is trying to add movie information it sets texts to *textView* boxes and adds photo from *URL* to *imageView* with *Glide*.

```
@Override
public int getItemViewType(int position) {
    return (position == movieList.size()) ? VIEW_TYPE_MORE : VIEW_TYPE_ITEM ;
}

@Override
public MovieAdapter.MyViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup,
int i)
{
    View view;
    if(i == VIEW_TYPE_ITEM) {
        view = LayoutInflater.from(viewGroup.getContext())
            .inflate(R.layout.movie_row_item, viewGroup, false);
    } else {
        view =
        LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.load_more_items,
        viewGroup, false);
    }
    return new MyViewHolder(view);
}

@Override
public void onBindViewHolder(final MovieAdapter.MyViewHolder viewHolder, int i){
    if(i == movieList.size()) {
        viewHolder.button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(mContext, "More", Toast.LENGTH_SHORT).show();
                mainActivity.preLoadJSON(mContext);
            }
        });
    } else {
        viewHolder.title.setText(movieList.get(i).getTitle());
        viewHolder.rating.setText("Rating: " + movieList.get(i).getRating());
        viewHolder.release_date.setText("Release date: " +
        movieList.get(i).getReleaseDate());
    }
}
```

```

        String poster = movieList.get(i).getImageurl();

        Glide.with(mContext).load(posters).apply(option).into(viewHolder.imageUrl);
    }
}

```

Figure 11. Main functions of choosing and setting information in adapter

To start adding information program needs to set up *recyclerView* and movie list (Figure 12). To set up *recyclerView* program needs to find *recyclerView* id and create new adapter. When *recyclerView* and adapter created, it checks orientation of phone, if it is hold vertically it displays one movie information in a row, if it is hold horizontally it displays two movies information in a row.

```

private void recyclerViewOrientation()
{
    recyclerView = (RecyclerView) findViewById(R.id.recycleViewerId);
    recyclerView.removeAllViewsInLayout();

    if(exists == false) {
        movieInfoList = new ArrayList<>();
        exists = true;
    }
    movieAdapter = new MovieAdapter(this, movieInfoList, this);

    if(getActivity().getResources().getConfiguration().orientation ==
    Configuration.ORIENTATION_PORTRAIT)
    {
        recyclerView.setLayoutManager(new GridLayoutManager(this, 1));
    } else
        recyclerView.setLayoutManager(new GridLayoutManager(this, 2));

    recyclerView.setItemAnimator(new DefaultItemAnimator());
    recyclerView.setAdapter(movieAdapter);
    movieAdapter.notifyDataSetChanged();
}

```

Figure 12. Function for setting new *recyclerView* and choosing its layout

After *recyclerView* and adapter is setup it needs to read *JSON* file and display (Figure 13). *JSON* files from *tmDb* shows twenty movies in one page and each page needs to be requested separately. Before loading *JSON* it checks, which search type is currently set and which page is currently needed. After checking which type of movies is needed, it creates *URL* and loads *JSON* file [3]. Then it sent to adapter to display information (Figure 15) [4].

```

private void loadJSON(Context mContext){

    try{
        if (BuildConfig.THE_MOVIE_DB_API_TOKEN.isEmpty()){
            Toast.makeText(getApplicationContext(), "Please obtain API Key firstly
            from themoviedb.org", Toast.LENGTH_SHORT).show();
            pd.dismiss();
            return;
        }
        final String pageString = String.valueOf(currentPage++);
        System.out.println("Current page " + currentPage);
        Client client = new Client();
        Service apiService =
            Client.getClient().create(Service.class);
    }
}

```

```

        Call<MoviesInfo> call;
        switch (type) {
            case "upcoming":
                call =
apiService.getUpcomingMovies(BuildConfig.THE_MOVIE_DB_API_TOKEN, pageString);
                break;
            case "topRated":
                call =
apiService.getTopRatedMovies(BuildConfig.THE_MOVIE_DB_API_TOKEN, pageString);
                break;
            default: call =
apiService.getPopularMovies(BuildConfig.THE_MOVIE_DB_API_TOKEN, pageString);
        }
        call.enqueue(new Callback<MoviesInfo>() {
            @Override
            public void onResponse(Call<MoviesInfo> call, Response<MoviesInfo>
response) {
                List<Movie> movies;
                movies = response.body().getResults();
                int scrollPosition = movieInfoList.size();
                movieInfoList.addAll(movies);
                recyclerView.setAdapter(movieAdapter);
                recyclerView.scrollToPosition(scrollPosition);
                pd.dismiss();
            }

            @Override
            public void onFailure(Call<MoviesInfo> call, Throwable t) {
                Log.d("Error", t.getMessage());
                Toast.makeText(MainActivity.this, "Error Fetching Data!",
Toast.LENGTH_SHORT).show();
            }
        } );
    } catch (Exception e){
        Log.d("Error", e.getMessage());
        Toast.makeText(this, e.toString(), Toast.LENGTH_SHORT).show();
    }
}

```

Figure 13. Function to read information from JSON file

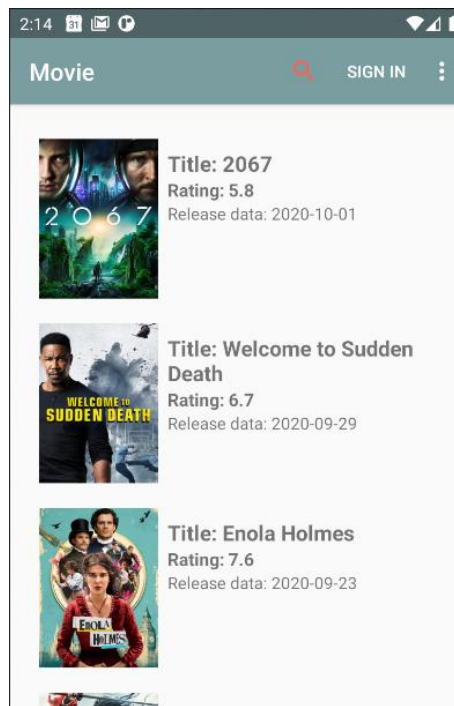


Figure 14. Representing gotten information from source

Task #4. In home screen be able to choose different categories of movies

For searching different types of categories I added items to action bar and made them to be hidden as sub item (Figure 17). When one of the categories is chosen it calls *onOptionsItemSelected* function, which then sends search type to *changeRecyclerView* (Figure 16) and renews *recyclerView* by deleting current stored information about movies in list, setting current page to 1 and changing type of search.

```
private void changeRecyclerView(String searchType)
{
    if(type != searchType) {
        type = searchType;
        exists = false;
        recyclerViewOrientation();
        currentPage = 1;
        preloadJSON(this);
    } else Toast.makeText(this, "This search type is already selected",
        Toast.LENGTH_SHORT).show();
}
```

Figure 15. Code for showing, which category of movies to show

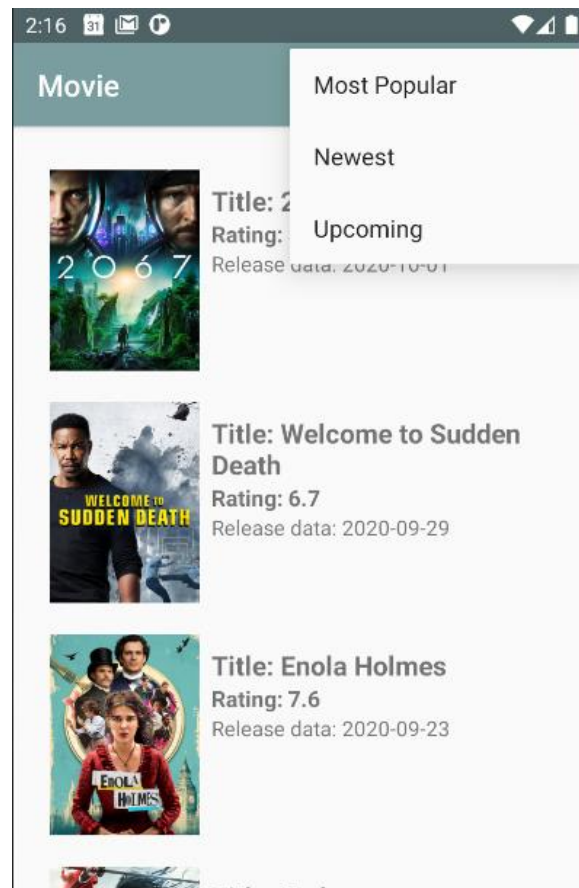


Figure 16. Top right corner let's choose search type.

Defense Task #5. There should be two activities in the app (it can be the existing ones). The second activity (which is created from the first activity) has to have an input/editbox and a button. When the value "Hello" is written in the input/editbox and the button is pressed, the first activity has to change color (your choice of color but has to be different than the current one).

For this task edited menu search icon, which wasn't used yet. When the search icon is pressed it start activity wanting to get information. In second activity user has edit box and button. It sends text written in edit box, if it says hello it changes layout background color to red.

```
Intent intent = new Intent(this, MainActivityDefense.class);
startActivityForResult(intent, 1);
```

Figure 17. Initiates second activity expecting result

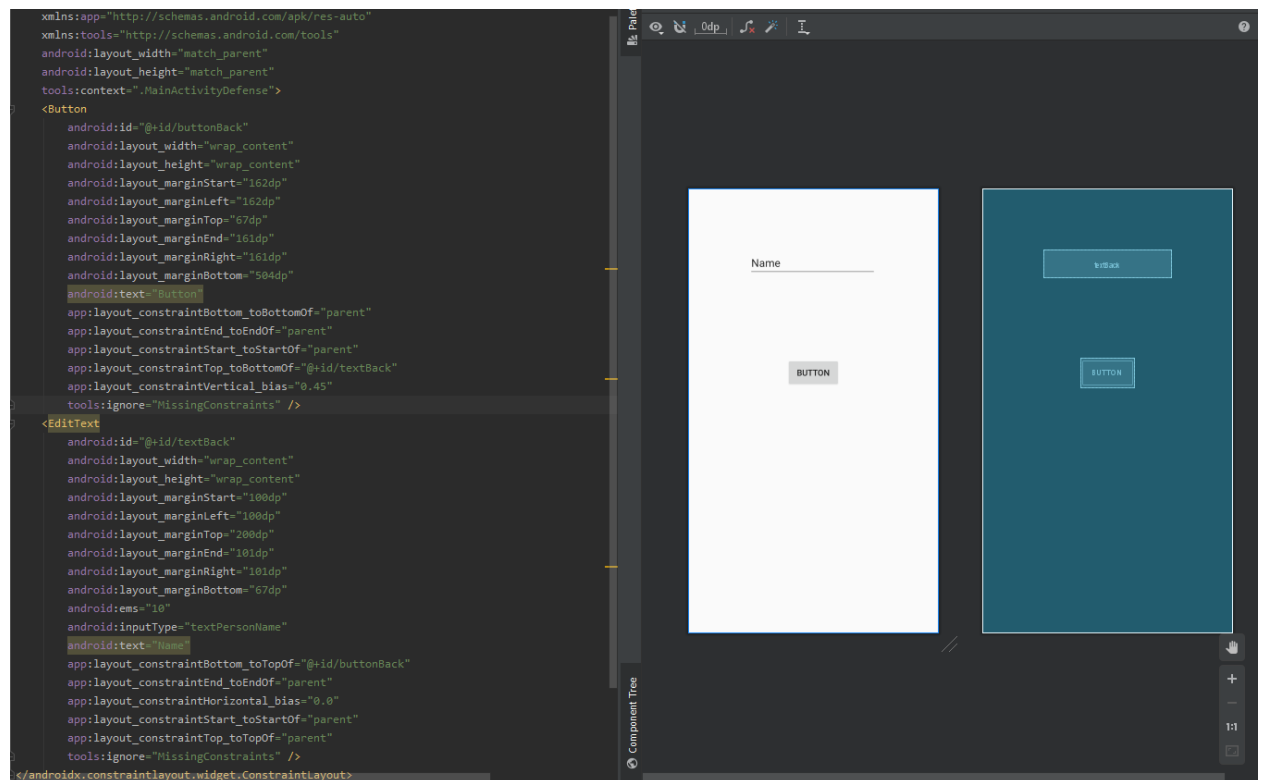


Figure 18. Second activity layout

```
Button button = (Button) findViewById(R.id.buttonBack);
final EditText editText = findViewById(R.id.textBack);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String text = editText.getText().toString();
        Intent intent = new Intent(MainActivityDefense.this, MainActivity.class);
        intent.putExtra("TEXT", text);

        setResult(RESULT_OK, intent);
        finish();
    }
});
```

```

    }
});

```

Figure 19. Code to set up second activity and go back to main activity

```

if (requestCode == 1) {
    String temp = data.getStringExtra("TEXT");
    System.out.println(temp);
    if(temp.equals("Hello")) {
        View view = this.getWindow().getDecorView();
        view.setBackgroundColor(Color.RED);
    }
}

```

Figure 20. Checks if text from second activity is "Hello"

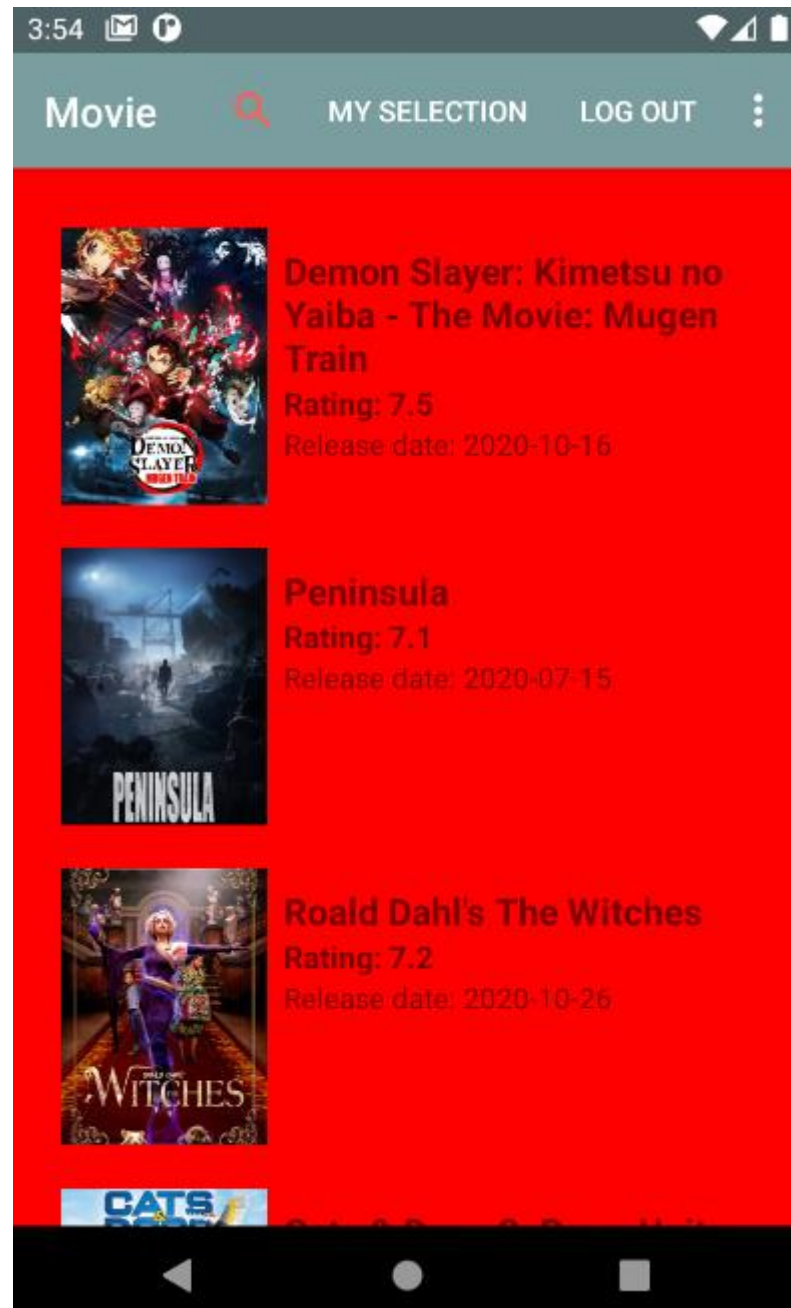


Figure 21. Main activity background color red

Reference list

1. <https://firebase.google.com/docs/auth/android/google-signin#java>

2. https://www.youtube.com/watch?v=bBJF1M5h_UU
3. <https://www.youtube.com/watch?v=OOLFhtyCspA&t>
4. <https://awsrh.blogspot.com/2018/03/volley-glide-tutorial-parse-json.html>