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

Description of Your app *3*
Functionality of your app *4*
List of functions (adapt to your own app) 4
Solution *5*
Task #1. Create Database 5
Task #2. Connect with google account 5
Task #3. Connect with imDb source 6
Task #4. In home screen be able to choose different categories of movies 13
Task #6. Set up database 17
Task #7. Add ability to search for movies 17
Task #8. When clicking a movie, show more information about it 19
Task #9. User can add movie to watchlist 21
Task #10. Be able to mark movie from watchlist as watched 23
Task #11. Check your current watchlist 23
Task #12. Sort watchlsit by alphabet, date, rating, etc 24
Task #13. Make a QR reader 27
Task #14. Make a QR reader 28
Task #15. Search for movie includes ability to add text by speaking 29
Task #16. In description of movies, user can try to watch trailer of the movie 31
Task #17. App pushes notification with a suggestion to watch a movie. Notifications time is customizable 32
Task #18. Use light sensor to check if it is dark or bright around user. If it is dark, then show suggestion for horror movie, if it is bright, then show suggestion for comedy movie 36
Reference list *36*

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

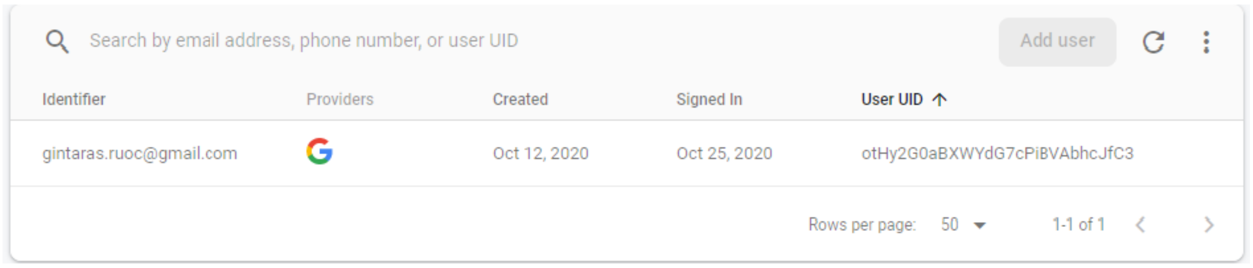
List of functions (adapt to your own app)

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). (defense task)
6. Set up database
7. Add ability to search for a movie
8. When clicking a movie, show more information about it
9. User can add a movie to a watchlist
10. Be able to mark movie as watched or remove it
11. Check your current watchlist
12. Sort watchlist by alphabet, date, rating, etc.
13. Add progress bar to watchlist, that shows how many movies from watchlist has been watched (defense task)
14. Make a QR reader.
15. Search for movie includes ability to add text by speaking.
16. In description of movies, user can try to watch trailer of the movie.
17. App pushes notification with a suggestion to watch a movie. Notifications time is customizable.
18. Use light sensor to check if it is dark or bright around user. If it is dark, then show suggestion for horror movie, if it is bright, then show suggestion for comedy movie. (defense task)

Solution

Task #1. Create Database

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



Search by email address, phone number, or user UID

Identifier	Providers	Created	Signed In	User UID ↑
gintaras.ruoc@gmail.com	G	Oct 12, 2020	Oct 25, 2020	otHy2G0aBXWYdG7cPiBVAbhcJfC3

Rows per page: 50 1-1 of 1

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

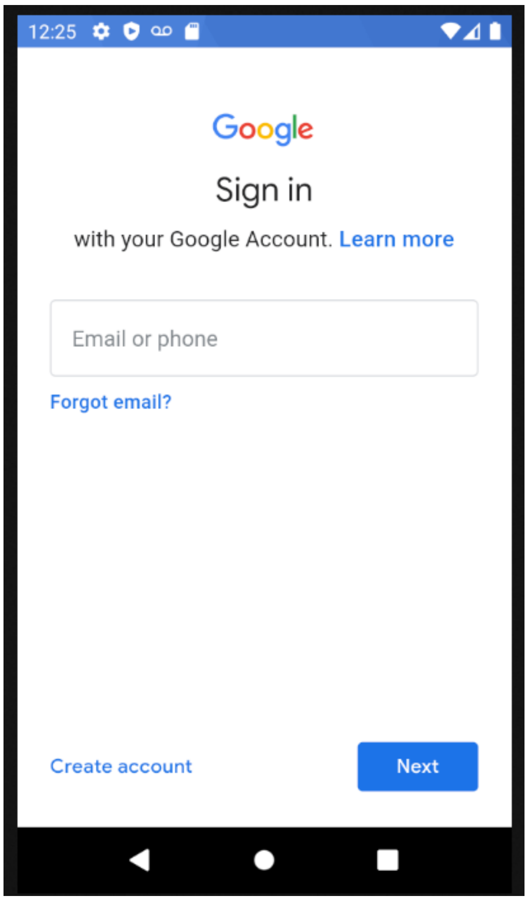


Figure 3. Goolge sign in form

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*

```java
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**

Than 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.

```java
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;
    }
}
```

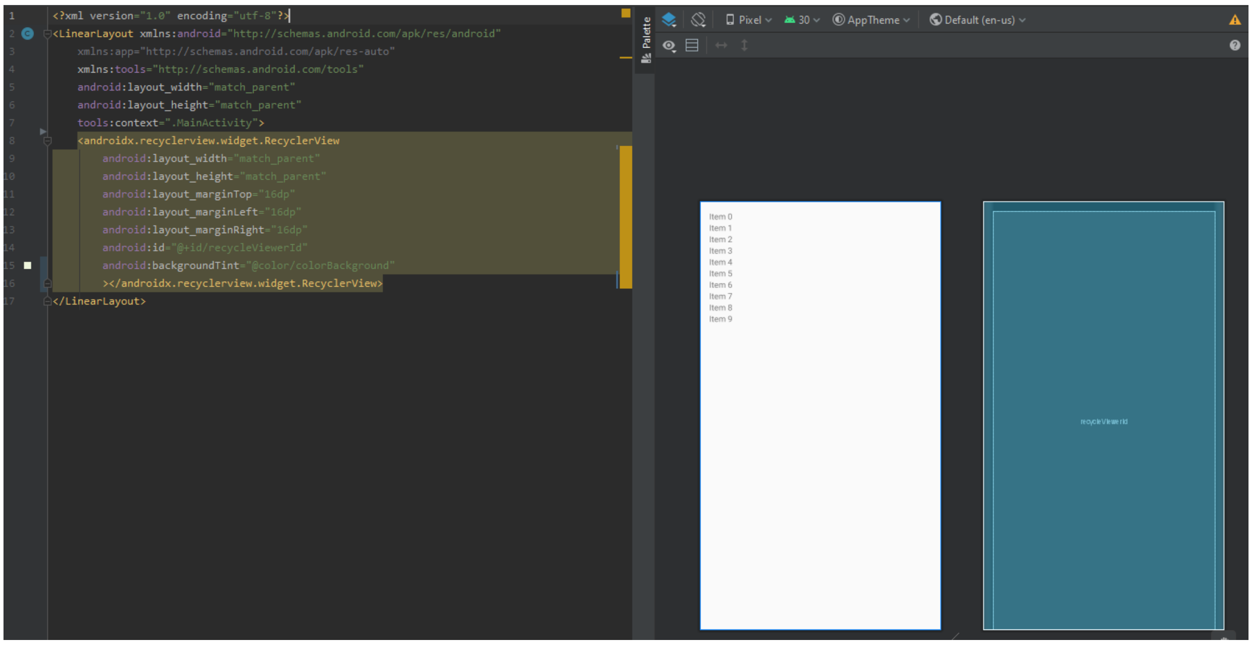
Figure 6. Used to in reading *JSON* file and storing movie info in List

```
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

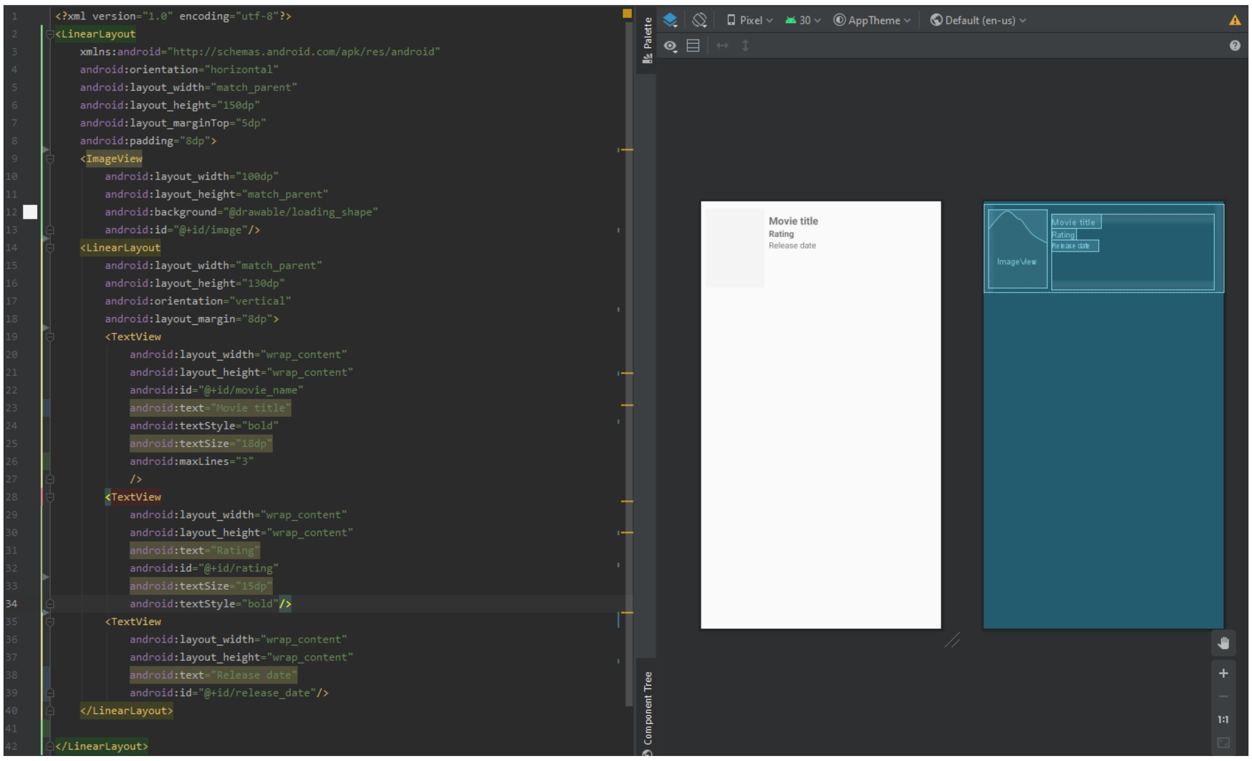
To display information from *JSON* there are three *XML* files *activity_main*, *movie_row_item* and *load_more_items*. *Activity_main* holds *recyclerView* component, which stores other two types of *XML* files (Figure 8). With *movie_row_item* it displays one movie information (Figure 9).

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



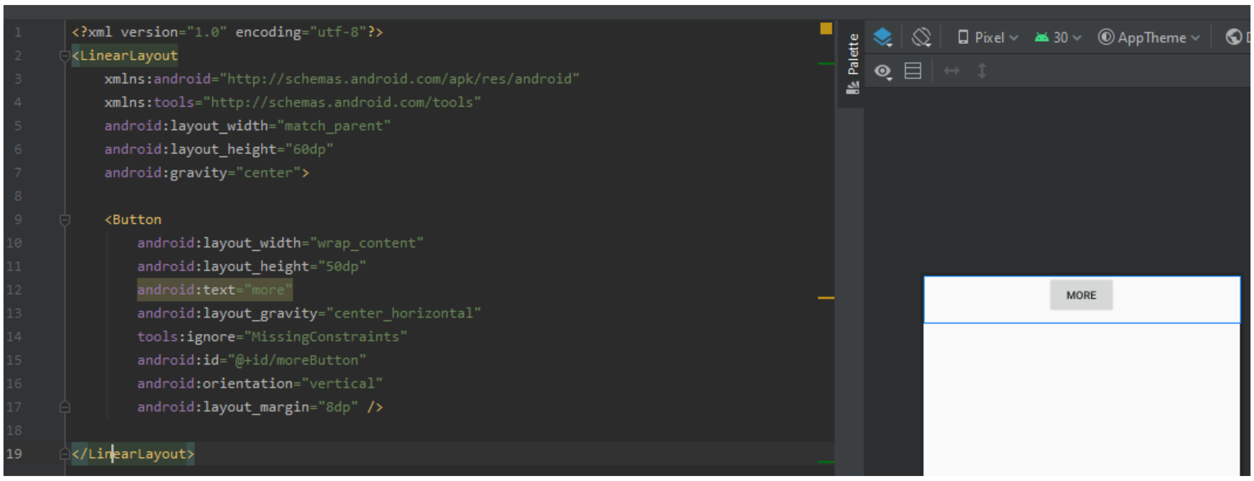
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <androidx.recyclerview.widget.RecyclerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:id="@+id/recycleViewerId"
        android:backgroundTint="@color/colorBackground"
        ></androidx.recyclerview.widget.RecyclerView>
</LinearLayout>
```

Figure 8. ***Activity_main.XML*** **file**



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="150dp"
    android:layout_marginTop="5dp"
    android:padding="8dp">
    <ImageView
        android:layout_width="100dp"
        android:layout_height="match_parent"
        android:background="@drawable/loading_shape"
        android:id="@+id/image"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="130dp"
        android:orientation="vertical"
        android:layout_margin="8dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/movie_name"
            android:text="Movie title"
            android:textStyle="bold"
            android:textSize="18dp"
            android:maxLines="3"
            />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Rating"
            android:id="@+id/rating"
            android:textSize="15dp"
            android:textStyle="bold"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Release date"
            android:id="@+id/release_date"/>
    </LinearLayout>

</LinearLayout>
```

Figure 9. ***Movie_row_items.XML*** **file**



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="60dp"
    android:gravity="center">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="50dp"
        android:text="more"
        android:layout_gravity="center_horizontal"
        tools:ignore="MissingConstraints"
        android:id="@+id/moreButton"
        android:orientation="vertical"
        android:layout_margin="8dp" />

</LinearLayout>
```

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*.

```java
@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(poster).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.

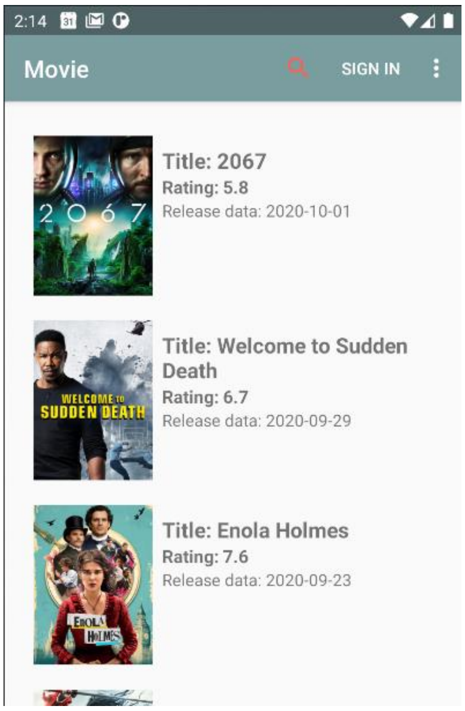
```
private void recyclerViewOrientation()
{
    recyclerView = (RecyclerView) findViewById(R.id.recycleViewerId);
    recyclerView.removeAllViewsInLayout();

    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]. Than 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


2:14

Movie

SIGN IN

Title: 2067
Rating: 5.8
Release data: 2020-10-01

Title: Welcome to Sudden Death
Rating: 6.7
Release data: 2020-09-29

Title: Enola Holmes
Rating: 7.6
Release data: 2020-09-23

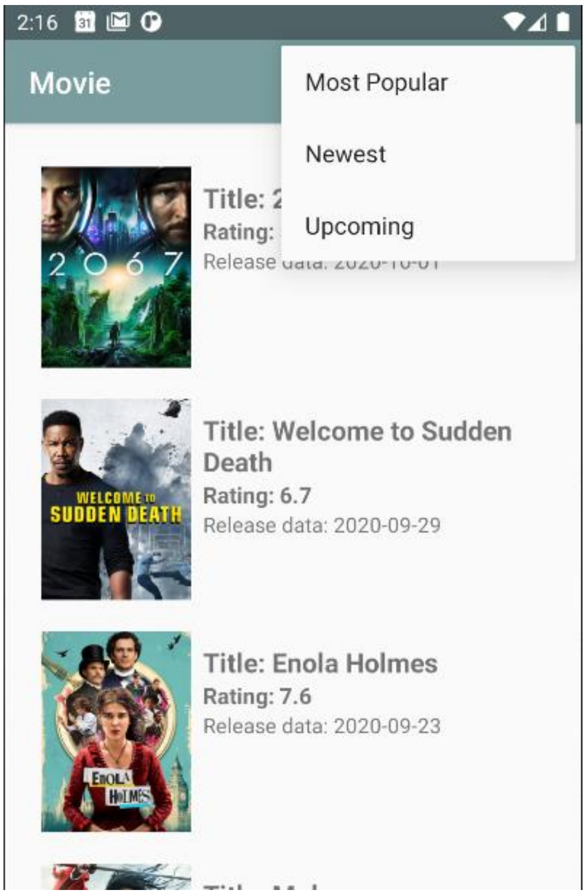
Figure 14. Representing gotten information from source

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 than sends search type to *changeRecycleView* (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



Title: 2

Rating:

Release data: 2020-10-01

Title: Welcome to Sudden Death

Rating: 6.7

Release data: 2020-09-29

Title: Enola Holmes

Rating: 7.6

Release data: 2020-09-23

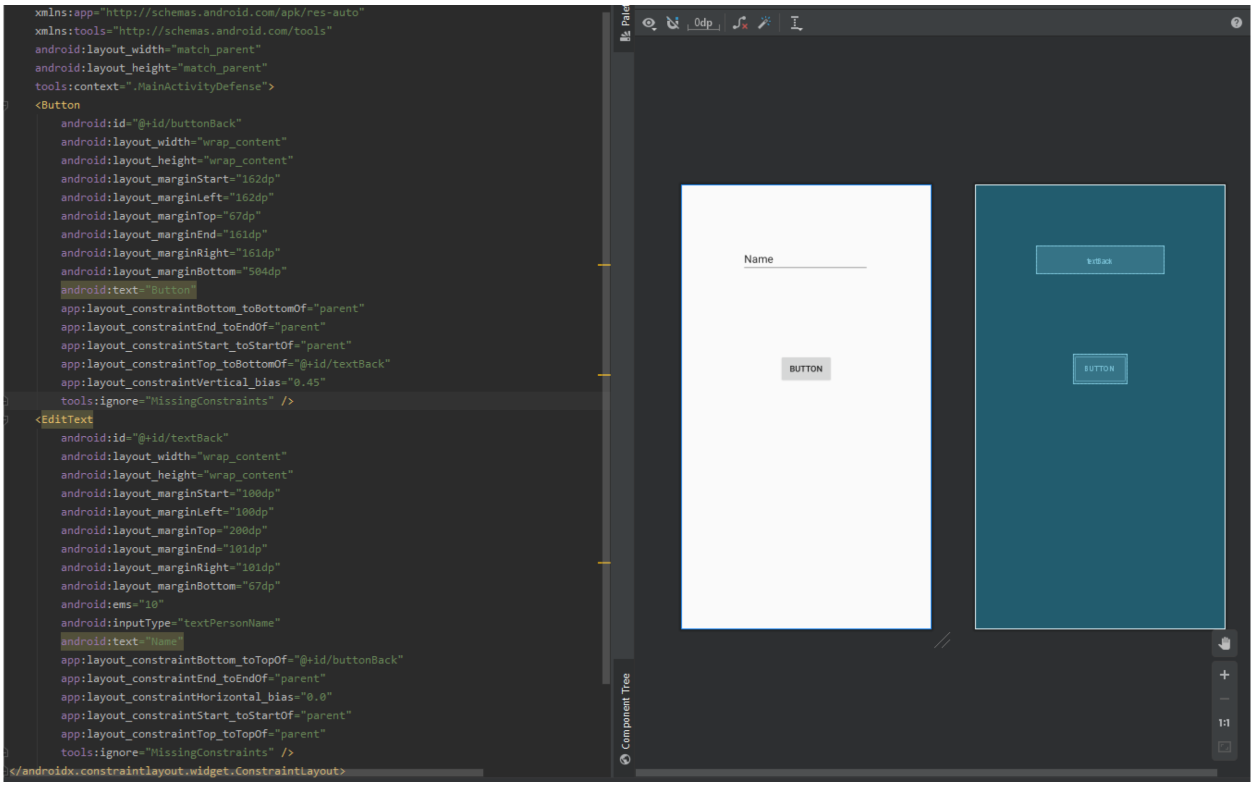
Figure 16. Top right corner let's choose search type.

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



```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivityDefense">
    <Button
        android:id="@+id/buttonBack"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="162dp"
        android:layout_marginLeft="162dp"
        android:layout_marginTop="67dp"
        android:layout_marginEnd="161dp"
        android:layout_marginRight="161dp"
        android:layout_marginBottom="504dp"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textBack"
        app:layout_constraintVertical_bias="0.45"
        tools:ignore="MissingConstraints" />
    <EditText
        android:id="@+id/textBack"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="100dp"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="200dp"
        android:layout_marginEnd="101dp"
        android:layout_marginRight="101dp"
        android:layout_marginBottom="67dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="Name"
        app:layout_constraintBottom_toTopOf="@+id/buttonBack"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="MissingConstraints" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

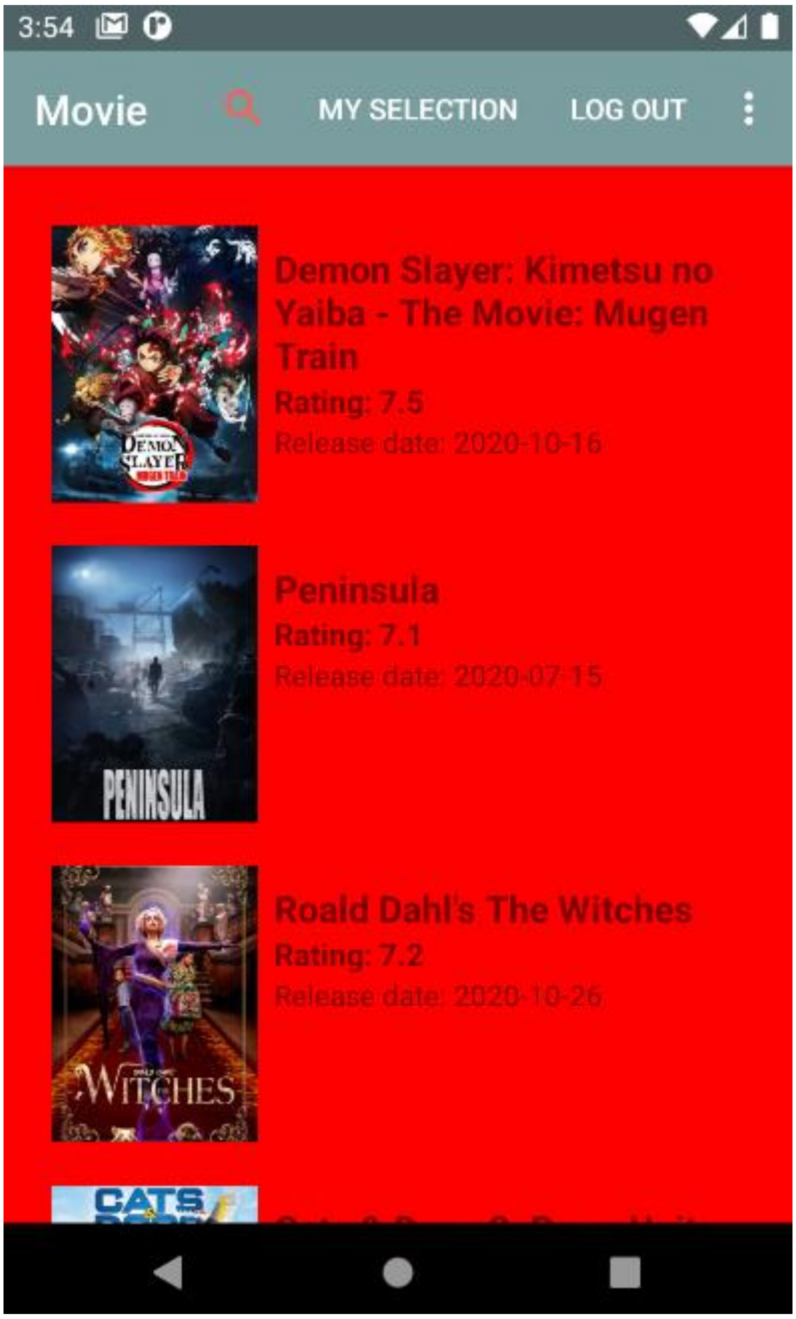
Figure 18. Second activity layout

```
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"



Demon Slayer: Kimetsu no Yaiba - The Movie: Mugen Train
Rating: 7.5
Release date: 2020-10-16

Peninsula
Rating: 7.1
Release date: 2020-07-15

Roald Dahl's The Witches
Rating: 7.2
Release date: 2020-10-26

Figure 21. Main activity background color red

Task #6. Set up database

As a database application uses *firebase* service. Firebase is *noSQL* type of database, so there is no need to add *SQL* type of relations to use it. To be able to add information to database developer needs to install *google* repository by going to **Tools** > **SDK Manager** > **SDK Tools**. When *google* repository is installed, developer can access *firebase* assistant by going **Tools** > **Firebase**. To store user watchlist, application stores user id, under user id there is movies list. In movies list it saves new item movie id, under which database saves mark if movie is watched (Figure 22).

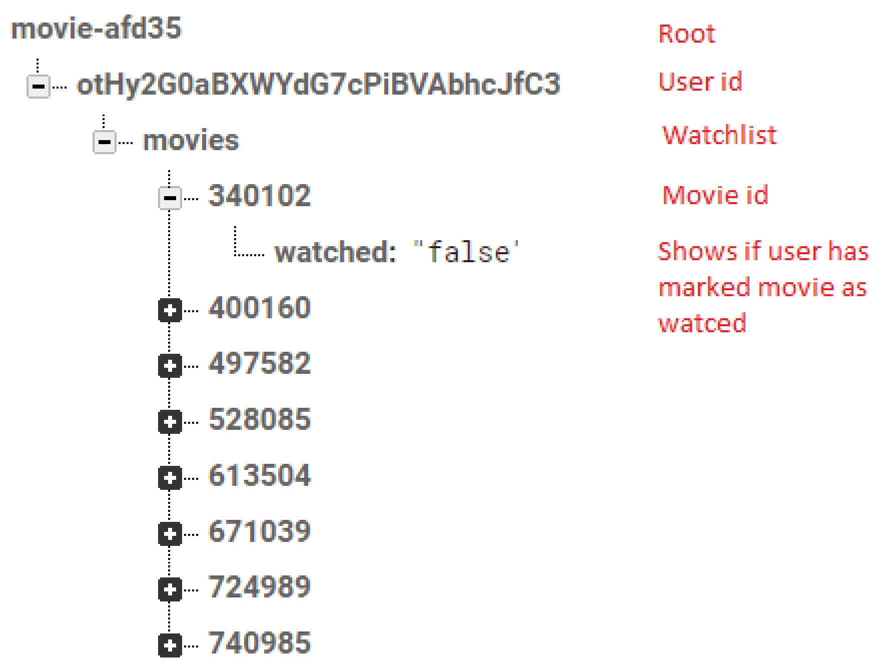


Figure 22. Direbase database, a way of storing users watchlist

Task #7. Add ability to search for movies

When user clicks search button, app prompts dialog in which it requires to write something to find movies (Figure 23). There was no need to add filtration for unwanted symbols, such as “/?.”, movie database handles it themselves. If user doesn’t write anything into dialog input, app makes a toast notifying about it. Getting information of search is handled the same way as other gotten information like most popular, etc.

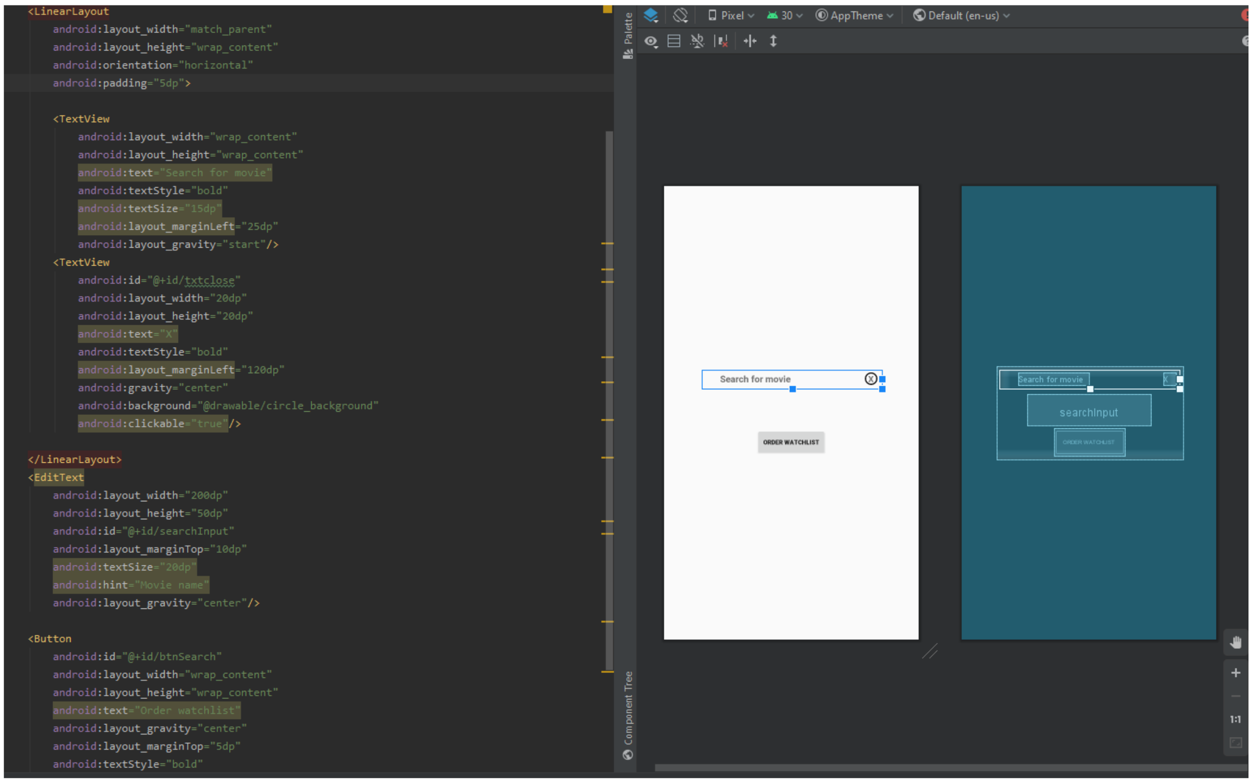

```xml
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="5dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Search for movie"
        android:textStyle="bold"
        android:textSize="15dp"
        android:layout_marginLeft="25dp"
        android:layout_gravity="start"/>
    <TextView
        android:id="@+id/txtclose"
        android:layout_width="20dp"
        android:layout_height="20dp"
        android:text="X"
        android:textStyle="bold"
        android:layout_marginLeft="120dp"
        android:gravity="center"
        android:background="@drawable/circle_background"
        android:clickable="true"/>

</LinearLayout>
<EditText
    android:layout_width="200dp"
    android:layout_height="50dp"
    android:id="@+id/searchInput"
    android:layout_marginTop="10dp"
    android:textSize="20dp"
    android:hint="Movie name"
    android:layout_gravity="center"/>

<Button
    android:id="@+id/btnSearch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Order watchlist"
    android:layout_gravity="center"
    android:layout_marginTop="5dp"
    android:textStyle="bold"
```

Figure 23. Dialog promt to search for movies

```java
public boolean searchPopUp(View v){
    final Dialog searchDialog = new Dialog(this);
    TextView txtclose;
    final EditText inputText;
    final Button searchBtn;
    final ImageButton imageBtn;
    searchDialog.setContentView(R.layout.search);
    txtclose = (TextView) searchDialog.findViewById(R.id.txtclose);
    inputText = searchDialog.findViewById(R.id.searchInput);
    searchBtn = searchDialog.findViewById(R.id.btnSearch);

    txtclose.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            searchDialog.dismiss();
        }
    });
    searchBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            searchQuery = inputText.getText().toString();
            if(searchQuery.isEmpty())
                Toast.makeText(getActivity(), "Movie input is empty",
Toast.LENGTH_SHORT).show();
            else {
                changeRecyclerView("search");
                searchDialog.dismiss();
            }
        }
    });

    searchDialog.getWindow().setBackgroundDrawable(new
ColorDrawable(Color.TRANSPARENT));
```

```
    searchDialog.show();
    return true;
}
```

Figure 24. Code for activating search

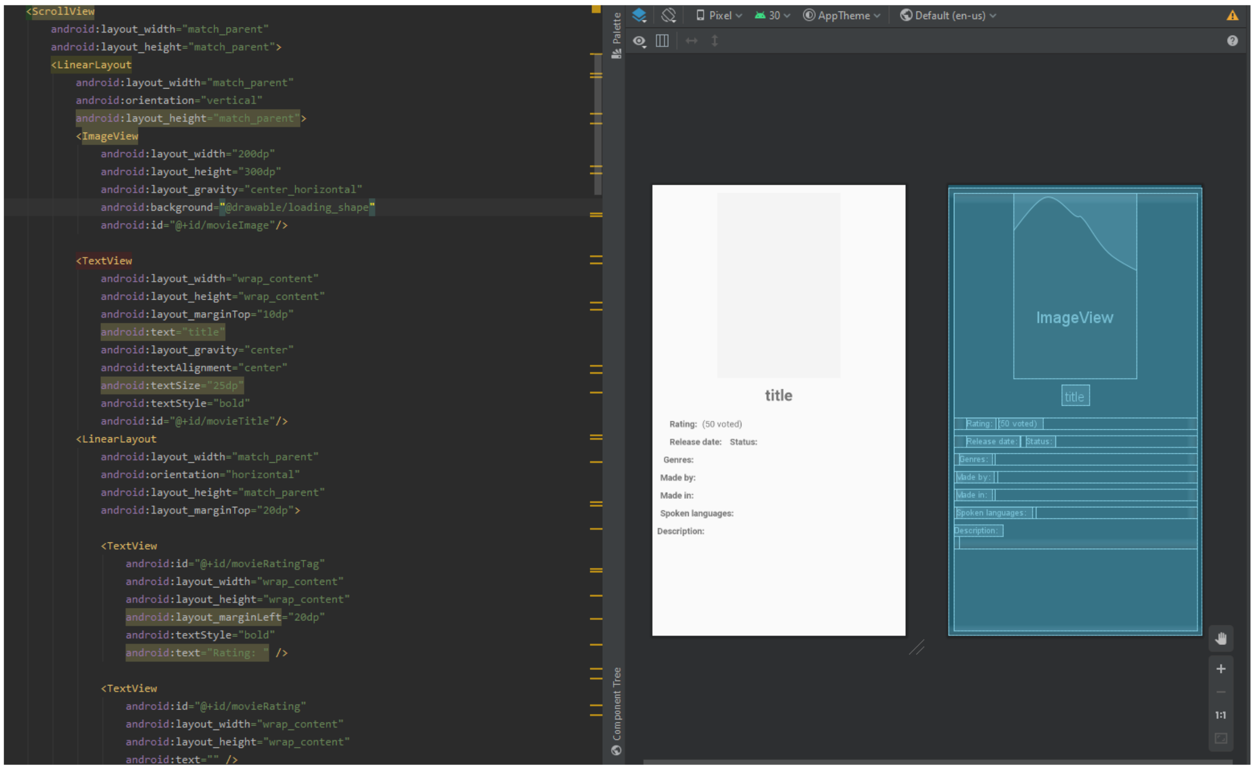
Task #8. When clicking a movie, show more information about it

In movie adapter each movie item gets a new on click listener (Figure 25), which creates new intent to movie details. For this to work intent gets a movie id, which will be used to get more details about movie.

```
itemView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int pos = getAdapterPosition();
        if(pos != RecyclerView.NO_POSITION)
        {
            Movie clickedDataItem = movieList.get(pos);
            Intent intent = new Intent(mContext, MovieDetails.class);
            intent.putExtra("ID", clickedDataItem.getId());
            if(user != null)
                intent.putExtra("USER", user);
            else intent.putExtra("USER", "");
            mContext.startActivity(intent);
        }
    }
});
```

Figure 25. Code for goint to new activity to display movie details

To dipslay movie details app needs a new layout to show image, title, rating, release date, genres, who made it, where was it made, languages and descrpition. (Figure 26)



```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:orientation="vertical"
        android:layout_height="match_parent">
        <ImageView
            android:layout_width="200dp"
            android:layout_height="300dp"
            android:layout_gravity="center_horizontal"
            android:background="@drawable/loading_shape"
            android:id="@+id/movieImage"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="title"
            android:layout_gravity="center"
            android:textAlignment="center"
            android:textSize="25dp"
            android:textStyle="bold"
            android:id="@+id/movieTitle"/>
        <LinearLayout
            android:layout_width="match_parent"
            android:orientation="horizontal"
            android:layout_height="match_parent"
            android:layout_marginTop="20dp">

            <TextView
                android:id="@+id/movieRatingTag"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dp"
                android:textStyle="bold"
                android:text="Rating: " />

            <TextView
                android:id="@+id/movieRating"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="" />
```

Figure 26. xml file for movie details layout

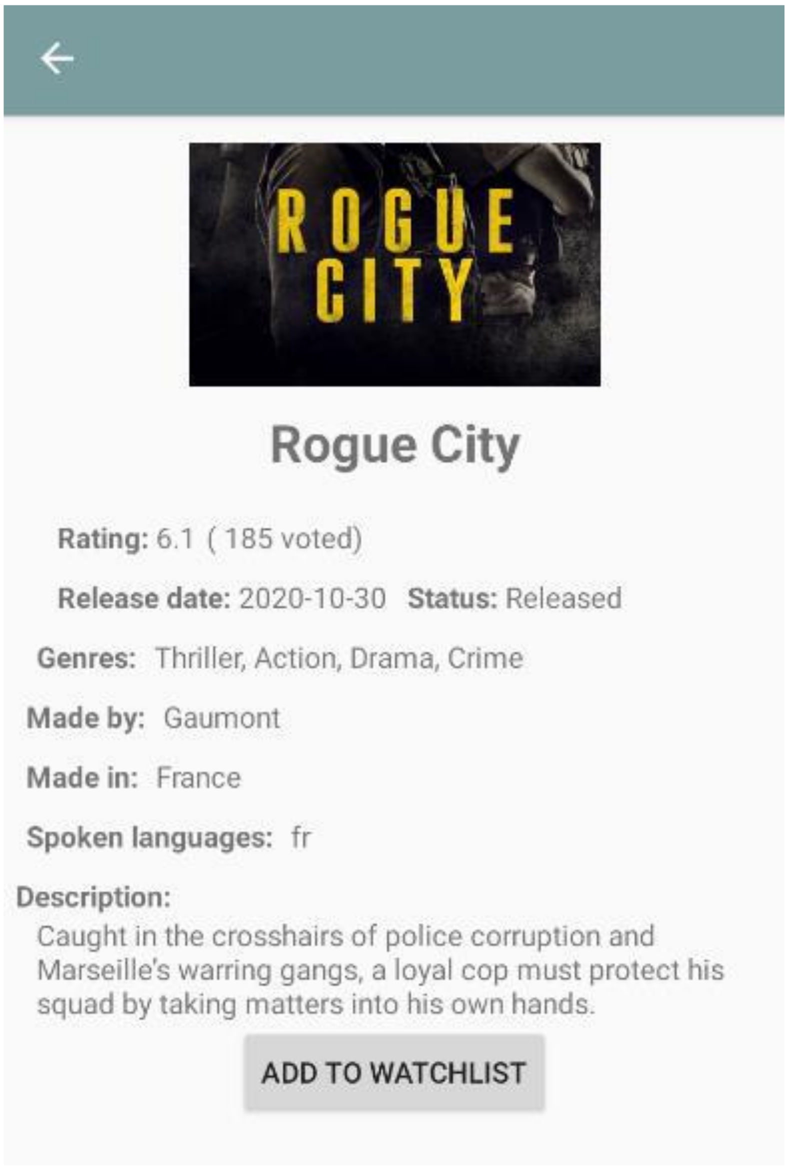
For getting information from movie database I used the same principle of getting information for most popular movies, top rated, etc (Figure 27). Just needed to create new service type for calling information. After getting needed information it is displayed on the phone (Figure 28).

```
private void getMovieDetails()
{
    String id = intent.getStringExtra("ID");
    System.out.println(id + " - io");
    Client Client = new Client();
    Service apiService =
            Client.getClient().create(Service.class);
    Call<Movie> call;
    call = apiService.getDetails(id, BuildConfig.THE_MOVIE_DB_API_TOKEN);

    call.enqueue(new Callback<Movie>() {
        @Override
        public void onResponse(Call<Movie> call, Response<Movie> response) {
            System.out.println(response.raw().request().url());
            details = response.body();
            fillInfo();
        }

        @Override
        public void onFailure(Call<Movie> call, Throwable t) {
            Toast.makeText(MovieDetails.this, "Error Fetching Data!",
Toast.LENGTH_SHORT).show();
            pd.dismiss();
        }
    } );
}
```

Figure 27. Getting information from movie database

```
@GET ("movie/{id}")
Call<Movie> getDetails(@Path("id") String id,
                       @Query("api_key") String apiKey);
```

Figure 28. Service call to get information about a movie


Rogue City

Rating: 6.1 (185 voted)

Release date: 2020-10-30 Status: Released

Genres: Thriller, Action, Drama, Crime

Made by: Gaumont

Made in: France

Spoken languages: fr

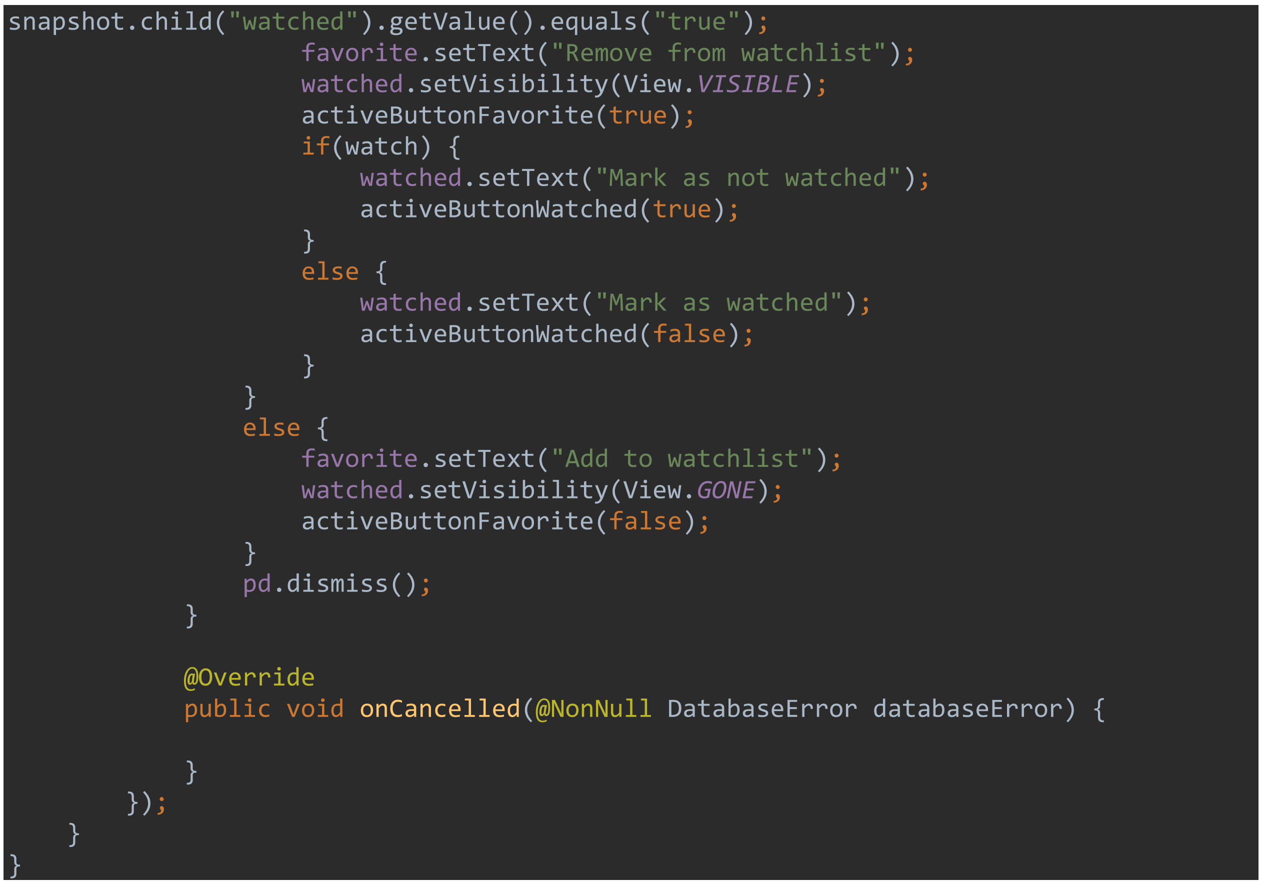
Description:
Caught in the crosshairs of police corruption and Marseille's warring gangs, a loyal cop must protect his squad by taking matters into his own hands.

Figure 29. Result of movie details display

Task #9. User can add movie to watchlist

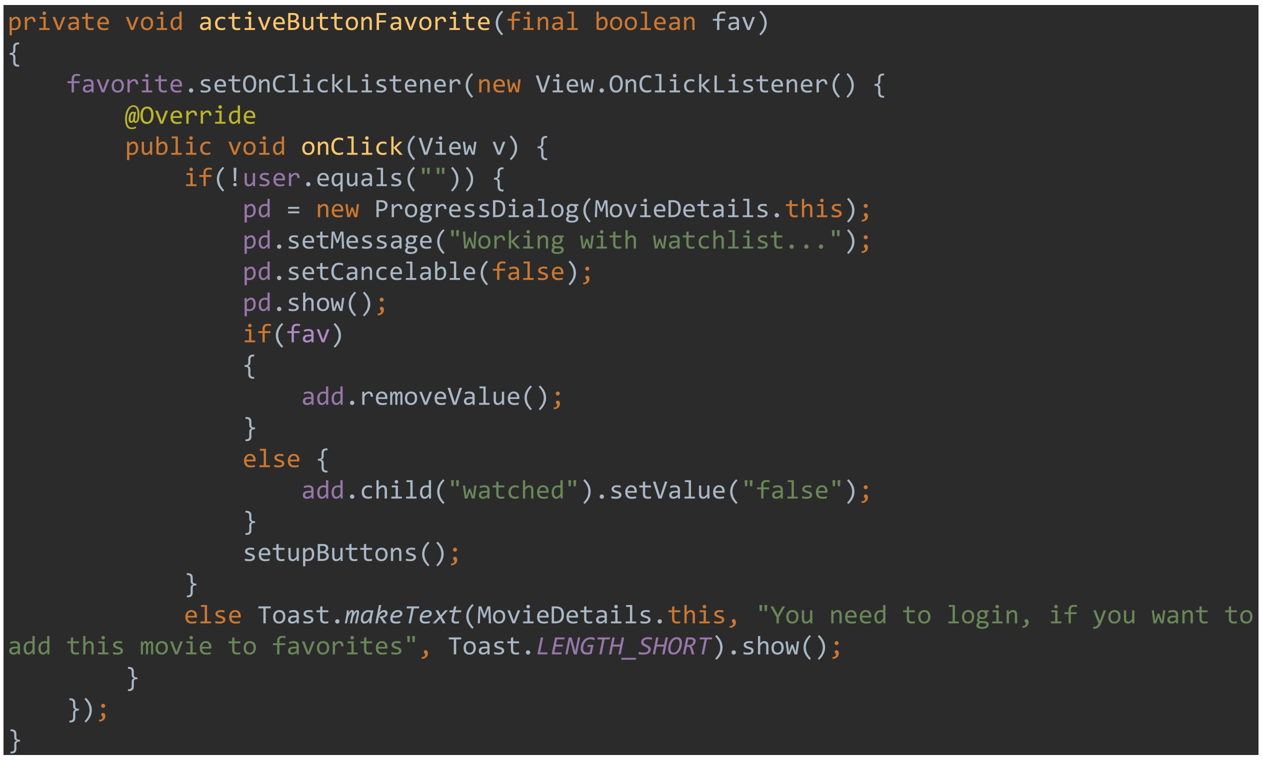
If user wants to add certain movie to the watchlist, user has to be logged in and in movie details tab. For this to work, when creating an intent to movie details user id has to be sent as reference to his database (Figure 22). When trying to add movie to watchlist if user id is not empty button for adding movie to watchlist is displayed (Figure 29). To add movie in watchlist app needs database exact point where it needs to write in database (Figure 30). In my case it saves under user id > movies > movie id > watched > false, this line adds movie info to watchlist, saving its id and if its watched, default being false (Figure 40). For removing it needs to exist in database, if it exists then instead of button showing “Add to watchlist” it will say “Remove from watchlist”.

```
private void setupButtons()
{
    add = database.getReference().child(user).child("movies").child(details.getId());
    if(!user.equals("")) {
        favorite.setVisibility(View.VISIBLE);
        add.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot snapshot) {
                boolean fav = snapshot.exists();
                if (fav) {
                    boolean watch =
```



```java
snapshot.child("watched").getValue().equals("true");
                    favorite.setText("Remove from watchlist");
                    watched.setVisibility(View.VISIBLE);
                    activeButtonFavorite(true);
                    if(watch) {
                        watched.setText("Mark as not watched");
                        activeButtonWatched(true);
                    }
                    else {
                        watched.setText("Mark as watched");
                        activeButtonWatched(false);
                    }
                }
                else {
                    favorite.setText("Add to watchlist");
                    watched.setVisibility(View.GONE);
                    activeButtonFavorite(false);
                }
                pd.dismiss();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
    }
}
```

Figure 30. Code for setting up buttons



```java
private void activeButtonFavorite(final boolean fav)
{
    favorite.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(!user.equals("")) {
                pd = new ProgressDialog(MovieDetails.this);
                pd.setMessage("Working with watchlist...");
                pd.setCancelable(false);
                pd.show();
                if(fav)
                {
                    add.removeValue();
                }
                else {
                    add.child("watched").setValue("false");
                }
                setupButtons();
            }
            else Toast.makeText(MovieDetails.this, "You need to login, if you want to
add this movie to favorites", Toast.LENGTH_SHORT).show();
        }
    });
}
```

Figure 31. Code for adding to or removing from movie watchlist

Task #10. Be able to mark movie from watchlist as watched

To be able to check movie as watched, first it needs to be in watchlist. Then app has to get information from watchlist if it is marked as watched or not. If it is marked as watched, app will display button for marking as not watched. If it is marked as not watched, app will display button for marking it as watched (Figure 30).

Task #11. Check your current watchlist

For displaying users watchlists firstly app has to read all movie ids under user id and if it is watched. Than it has to send all the movie ids to movie database, to get information about movies. After that app sorts watchlist by alphabet as default and then watchlist is displayed.

```
private void getMovies() {

    database = FirebaseDatabase.getInstance();
    reference = database.getReference().child(userId).child("movies");
    reference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            movieList.clear();
            baseMovieList.clear();
            i = (int) dataSnapshot.getChildrenCount();
            for (final DataSnapshot child : dataSnapshot.getChildren())
                {
                    String id = child.getKey();
                    Client Client = new Client();
                    Service apiService =
                            Client.getClient().create(Service.class);
                    Call<Movie> call;
                    call = apiService.getDetails(id,
BuildConfig.THE_MOVIE_DB_API_TOKEN);

                    call.enqueue(new Callback<Movie>() {
                        @Override
                        public void onResponse(Call<Movie> call, Response<Movie>
response) {
                            Movie info;
                            info = response.body();
                            if(child.child("watched").getValue().equals("true"))
                                info.setWatched(true);

                            baseMovieList.add(info);

                            i = i -1;
                            if(i <= 0) {
                                pd.dismiss();
                                progressBar();
                                sort();
                            }
                        }

                        @Override
                        public void onFailure(Call<Movie> call, Throwable t) {
                            Toast.makeText(MyWatchlist.this, "Error Fetching Data!",
Toast.LENGTH_SHORT).show();
                        }
                    } );

                }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });

}
```

Figure 32. Code for getting information about user watchlist

Task #12. Sort watchlsit by alphabet, date, rating, etc

When app gets all information about watchlist it is save in base list, which never changes and is not sorted, because than app doesn't need to read watchlist again, when sorting it just copies base list to another list. If user wants to sort watchlist app prompts with dialog (Figure 33), which contains three radio groups:

- show radio group - let's user to see all movies, only watched and only not watched;
- order by radio group – let's user to choose if watchlist should be sorted by alphabet, date or rating;
- order type radio group – let's user to choose if list should be sorted ascending or descending.

When user presses order watchlist, list is sorted by these three parameters.

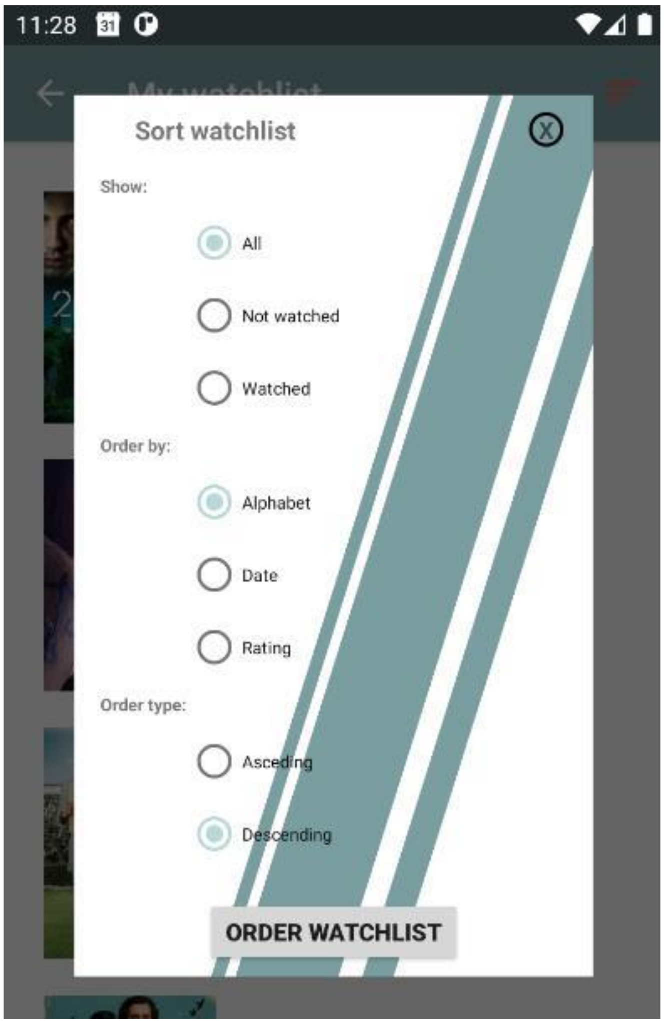


Figure 33. Dialog for sorting watchlist

```
btnSort.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(all.isChecked())
            show = "all";
        else if(notWatched.isChecked())
            show = "notWatched";
        else show = "watched";

        if(alphabet.isChecked())
            orderBy = "alphabet";
        else if(date.isChecked())
            orderBy = "date";
        else orderBy = "rating";

        if(ascending.isChecked())
            orderType = true;
        else orderType = false;

        sortDialog.dismiss();
        sort();
    }
});
```

Figure 34. Checking what was choosen in dialog

For watchlist base function is called, which checks how list needs to be sorted and calls functions which are needed for certain sorting.

```
private void sort()
{
    pd = new ProgressDialog(this);
    pd.setMessage("Sorting movies...");
    pd.setCancelable(false);
    pd.show();

    movieList.clear();
    if(show.equals("notWatched"))
        watched(false);
    else if(show.equals("watched"))
        watched(true);
    else movieList.addAll(baseMovieList);

    if(orderBy.equals("alphabet"))
        sortAlphabet();
    else if(orderBy.equals("date"))
        sortDate();
    else sortRating();

    if(orderType)
        Collections.reverse(movieList);

    recyclerView.setAdapter(movieAdapter);
    pd.dismiss();
}
```

Figure 35. Base sorting function, which determines functions that need to be called

User can choose to show all, watched or not watched movies. For this function only needs to check if movie was marked as watched (Figure 36).

```java
private void watched(boolean w){
    for (Movie temp:
         baseMovieList) {
        if(temp.isWatched() == w)
            movieList.add(temp);
    }
}
```

Figure 36. Check if movie is watched

To sort alphabetically it compares title strings (Figure 37). When sorting by release date it sorts in string format too, because release date is stored in strings, but if date matches, than it sorts by alphabet (Figure 38). Sorting by rating is the same as sorting by release date (Figure 39).

```java
private void sortAlphabet()
{
    for(int i = 0; i < movieList.size(); i++){
        for(int j = 1; j < movieList.size() - i; j++)
        {
            Movie a = movieList.get(j-1);
            Movie b = movieList.get(j);
            int result = a.getTitle().compareTo(b.getTitle());
            if(result > 0)
            {
                movieList.set(j-1, b);
                movieList.set(j, a);
            }
        }
    }
}
```

Figure 37. Sorts movies alphabetically

```java
private void sortDate()
{
    for(int i = 0; i < movieList.size(); i++){
        for(int j = 1; j < movieList.size() - i; j++)
        {
            Movie a = movieList.get(j-1);
            Movie b = movieList.get(j);
            int result = a.getReleaseDate().compareTo(b.getReleaseDate());
            if( result < 0)
            {
                movieList.set(j-1, b);
                movieList.set(j, a);
            }
            else if(result == 0)
            {
                result = a.getTitle().compareTo(b.getTitle());
                if(result > 0)
                {
                    movieList.set(j-1, b);
                    movieList.set(j, a);
                }
            }
        }
    }
}
```

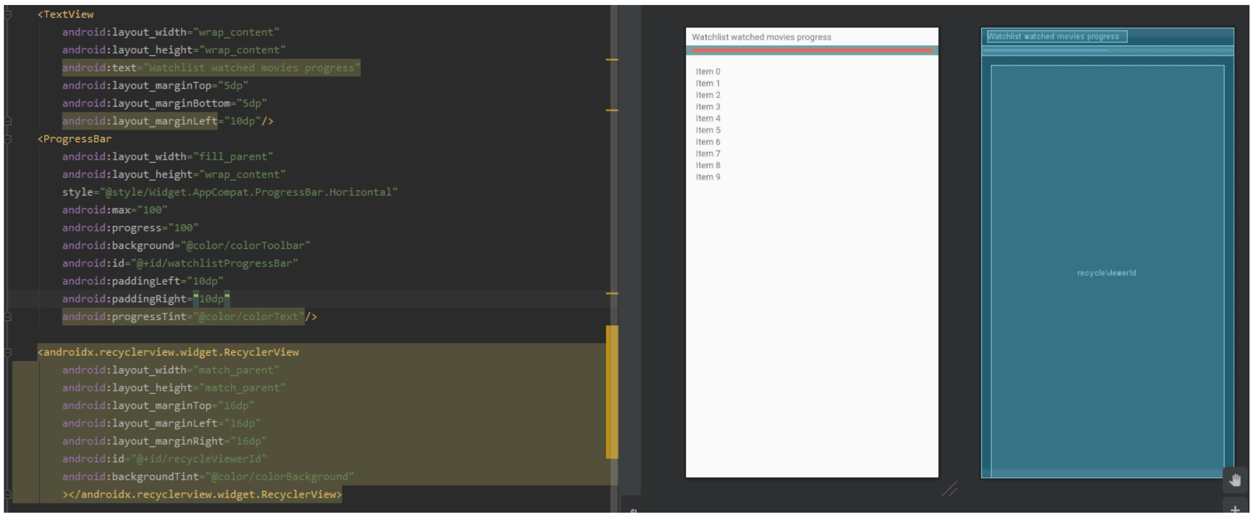
Figure 38. Sorts movies by release date

```java
private void sortRating()
{
    for(int i = 0; i < movieList.size(); i++){
        for(int j = 1; j < movieList.size() - i; j++)
        {
            Movie a = movieList.get(j-1);
            Movie b = movieList.get(j);
            int result = a.getRating().compareTo(b.getRating());
            if(result < 0)
            {
                movieList.set(j-1, b);
                movieList.set(j, a);
            }
            else if(result == 0)
            {
                result = a.getTitle().compareTo(b.getTitle());
                if(result > 0)
                {
                    movieList.set(j-1, b);
                    movieList.set(j, a);
                }
            }
        }
    }
}
```

Figure 39. Sorts movies by user rating

Task #13. Make a QR reader

Added progress bar to watchlist layout (Figure 19). In code, when movie watchlist is read from database, it than check how many movies are marked as watched. Than progress maximum is divided by watchlist size and multiplied by amount of watched movies (Figure 20).



```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Watchlist watched movies progress"
    android:layout_marginTop="5dp"
    android:layout_marginBottom="5dp"
    android:layout_marginLeft="10dp"/>
<ProgressBar
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:max="100"
    android:progress="100"
    android:background="@color/colorToolbar"
    android:id="@+id/watchlistProgressBar"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:progressTint="@color/colorText"/>

<androidx.recyclerview.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:id="@+id/recycleViewerId"
    android:backgroundTint="@color/colorBackground"
    ></androidx.recyclerview.widget.RecyclerView>
```

Figure 40. Watchlist layout with progress bar

```
private void progressBar()
{
    int amount = 0;
    for(Movie a:
        baseMovieList)
        if(a.isWatched())
            amount++;
    int fill = 100 / baseMovieList.size() * amount;
    progressBar.setProgress(fill);
}
```

Figure 41. Progress bar value counting

Task #14. Make a QR reader

To read QR code app uses barcode scanner. To start read app creates new activity, which only extends capture activity (Figure 42). When user point camera at QR code app automatically returns to main activity and checks if QR code contains *imdb* or *tmdb* url (Figure 44). If url contains id that matches movie id from *tmdb*, than it creates activity to show movie details (Figure 43).

```
private void scanCode()
{
    IntentIntegrator integrator = new IntentIntegrator(this);
    integrator.setCaptureActivity(CaptureAct.class);
    integrator.setDesiredBarcodeFormats(IntentIntegrator.QR_CODE_TYPES);
    integrator.setPrompt("Scanning Code");
    integrator.initiateScan();
}
```

Figure 42. Start of QR reader

```
private void showScannedMovie(final String id)
{
    Client Client = new Client();
    Service apiService =
            Client.getClient().create(Service.class);
    Call<Movie> call;
    call = apiService.getDetails(id, BuildConfig.THE_MOVIE_DB_API_TOKEN);

    call.enqueue(new Callback<Movie>() {
        @Override
        public void onResponse(Call<Movie> call, Response<Movie> response) {
            if(response.body() != null) {
                Intent intent = new Intent(MainActivity.this, MovieDetails.class);
                intent.putExtra("ID", id);
                if (user != null)
                    intent.putExtra("USER", user.getUid());
                else intent.putExtra("USER", "");
                startActivity(intent);
            }
            else Toast.makeText(MainActivity.this, "Can only show movies or tmdb 
doesn't have this film in database!", Toast.LENGTH_LONG).show();
        }

        @Override
        public void onFailure(Call<Movie> call, Throwable t) {
            Toast.makeText(MainActivity.this, "Error Fetching Data!", 
Toast.LENGTH_SHORT).show();
        }
    } );
}
```

Figure 43. Calls intent to display movie

```
if(requestCode == 49374) {
    IntentResult result = IntentIntegrator.parseActivityResult(requestCode, 
resultCode, data);
    if(result.getContents() != null)
    {
        String link = result.getContents();
        System.out.println(link + " url");
        // If QR code contains link to imdb website
        if(link.contains("imdb")){
            link = link.substring(0, link.lastIndexOf('/'));
            final String id = link.substring(link.lastIndexOf('/') + 1);
            showScannedMovie(id);
        }
        // If QR code contains link to tmdb website
        else if(link.contains("themoviedb"))
        {
            if(!link.contains("/tv/")) {
                link = link.substring(link.lastIndexOf("/") + 1);
                final String id = link.substring(0, link.indexOf("-"));
                showScannedMovie(id);
            }
            else Toast.makeText(MainActivity.this, "Can only show movies or tmdb
doesn't have this film in database!", Toast.LENGTH_LONG).show();
        }
        else Toast.makeText(this, "QR code doesn't contain link from themovedb or
imdb!", Toast.LENGTH_LONG).show();
    }

}
```

Figure 44. Checks gotten QR code

Task #15. Search for movie includes ability to add text by speaking

For speech to text program uses SpeechRecognizer, which activates by pressing microphone icon in search dialog. onResult function inside recognition listener app sets text gotten from speech into input text (Figure 45).

```
final SpeechRecognizer mSpeechRecognizer =
SpeechRecognizer.createSpeechRecognizer(this);

final Intent mSpeechRecognizerIntent = new
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
mSpeechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
        RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
mSpeechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
        Locale.getDefault());

mSpeechRecognizer.setRecognitionListener(new RecognitionListener() {
    @Override
    public void onReadyForSpeech(Bundle params) {

    }

    @Override
    public void onBeginningOfSpeech() {

    }

    @Override
    public void onRmsChanged(float v) {

    }

    @Override
    public void onBufferReceived(byte[] bytes) {
```

```java
    }

    @Override
    public void onEndOfSpeech() {

    }

    @Override
    public void onError(int errorCode) {
        String errorMessage;
        switch (errorCode) {
            case SpeechRecognizer.ERROR_AUDIO:
                errorMessage = "Audio recording error";
                break;
            case SpeechRecognizer.ERROR_CLIENT:
                errorMessage = "Client side error";
                break;
            case
                    SpeechRecognizer.ERROR_INSUFFICIENT_PERMISSIONS:
                errorMessage = "Insufficient permissions";
                break;
            case SpeechRecognizer.ERROR_NETWORK:
                errorMessage = "Network error";
                break;
            case SpeechRecognizer.ERROR_NETWORK_TIMEOUT:
                errorMessage = "Network timeout";
                break;
            case SpeechRecognizer.ERROR_NO_MATCH:
                errorMessage = "No match";
                break;
            case SpeechRecognizer.ERROR_RECOGNIZER_BUSY:
                errorMessage = "RecognitionService busy";
                break;
            case SpeechRecognizer.ERROR_SERVER:
                errorMessage = "error from server";
                break;
            case SpeechRecognizer.ERROR_SPEECH_TIMEOUT:
                errorMessage = "No speech input";
                break;
            default:
                errorMessage = "Didn't understand, please try again.";
                break;
        }
        inputText.setText(errorMessage);
    }

    @Override
    public void onResults(Bundle bundle) {
        //getting all the matches
        ArrayList<String> matches = bundle
                .getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);

        if (matches != null)
            inputText.setText(matches.get(0));
    }

    @Override
    public void onPartialResults(Bundle bundle) {

    }
    @Override
    public void onEvent(int i, Bundle bundle) {

    }
});
```

Figure 45. Speech recognition listener

```
imageBtn.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View view, MotionEvent motionEvent) {
        switch (motionEvent.getAction()) {
            case MotionEvent.ACTION_UP:
                mSpeechRecognizer.stopListening();
                Toast.makeText(getActivity(), "Writed text",
Toast.LENGTH_SHORT).show();
                break;

            case MotionEvent.ACTION_DOWN:
                mSpeechRecognizer.startListening(mSpeechRecognizerIntent);
                inputText.setText("");
                inputText.setHint("Listening...");
                break;
        }
        return false;
    }
});
```

Figure 46. Speech recognition activation

Task #16. In description of movies, user can try to watch trailer of the movie

Trailer that are being displayed are searched in *youtube* by writing movie title and trailer word. Then app needs to check if gotten videos from *youtube* contains trailer word. For this to work I used *youtube* api and *youtube* library to display video. Using created *youtube* api token application puts movie title and trailer word to get list of videos from database. Then program from given list, checks if video title contains trailer word and if it does, than video is played, if none of the videos contains trailer word it displays error.

```
private void watchTrailer(){
    watchTrailerBtn.setVisibility(View.GONE);
    Client Client = new Client();
    Service apiService =
            Client.getClientYoutube().create(Service.class);
    Call<YoutubeInfo> call;

    call = apiService.searchTailer("snippet", details.getTitle() + " trailer",
BuildConfig.YOUTUBE_API_TOKEN);
    call.enqueue(new Callback<YoutubeInfo>() {
        @Override
        public void onResponse(Call<YoutubeInfo> call, Response<YoutubeInfo>
response) {
            YoutubeInfo info = response.body();
            List<Video> videos;
            if(info.getError() != null)
                System.out.println("Code error: " + info.getError().getCodeError() +
" Error message" + info.getError().getMessageError());
            else {
                videos = info.getVideoList();
                for (final Video temp :
                        videos) {
                    if
(temp.getSnippet().getTitle().toLowerCase().contains("trailer")) {
                        youTubePlayerView.setVisibility(View.VISIBLE);
                        onInitializedListener = new
YouTubePlayer.OnInitializedListener() {
                            @Override
                            public void
onInitializationSuccess(YouTubePlayer.Provider provider, YouTubePlayer youTubePlayer,
boolean b) {
                                youTubePlayer.loadVideo(temp.getId().getId());

                            }

                            @Override
                            public void
onInitializationFailure(YouTubePlayer.Provider provider, YouTubeInitializationResult
youTubeInitializationResult) {

                            }
                        };
                        youTubePlayerView.initialize(BuildConfig.YOUTUBE_API_TOKEN,
onInitializedListener);
                        break;
                    }
                }

                if (youTubePlayerView.getVisibility() == View.GONE)
                    failedTrailer.setVisibility(View.VISIBLE);
            }
        }

        @Override
        public void onFailure(Call<YoutubeInfo> call, Throwable t) {
            failedTrailer.setVisibility(View.VISIBLE);
            Log.d("Error", t.getMessage());
            Toast.makeText(MovieDetails.this, "Error Fetching Data!",
Toast.LENGTH_SHORT).show();
        }
    });

    pd.dismiss();
}
```

Figure 47. Code to display trailer from *youtube*

Task #17. App pushes notification with a suggestion to watch a movie. Notifications time is customizable

One of the parts for notification is service, which manages notification activity. Service activates, when user logs in or detects that user is logged in already. If user logs out or when application starts, service disables and stops notifications (Figure 52 and 53). User is able to change notification timing and disable notifications. Application stores notification information in *firebase* (Figure 49). Each time service start, application gets notification information from firebase to set up notifications (Figure 48). If user doesn't have any notification information in *firebase*, application uses default information and puts it into database. To send notifications repeatedly to user at set time applications uses alarm manager (Figures 50 and 51). Notification has broadcast receiver class, which gets users unwatched movies watch list from *firebase* and randomly select movie to put into intent (Figure 13). Randomly selected movie is shown only, when notification is pressed.

```
private void getNotificationInfo(){
    databaseReference.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if(dataSnapshot.exists())
            {
                if(dataSnapshot.child(disabledString).getValue().equals("true")) {
                    enabled = true;

                    if(dataSnapshot.child(weekString).exists())
                        for (DataSnapshot temp:
                             dataSnapshot.child(weekString).getChildren()) {
                            week.add(Integer.parseInt(temp.getKey()));
                        }

                    hour =
Integer.parseInt(dataSnapshot.child(hourString).getValue().toString());
                    setUpNotification();
                }
                else enabled = false;
            }
            else setNotificationInfo();
        }

        @Override
```

Figure 48. Get notification information from *firebase*

```
private void setNotificationInfo(){
    databaseReference.removeValue();
    databaseReference.child(disabledString).setValue(enabled);
    if(enabled) {
        if (week != null)

            for (int temp :
                    week) {

databaseReference.child(weekString).child(String.valueOf(temp)).setValue("true");
            }

        databaseReference.child(hourString).setValue(hour);
    }
    setUpNotification();
}
```

Figure 49. Put notification information into firebase

```
private void setUpNotification(){
    cancelNotification();
    if(!enabled)
        return;
    Calendar calendar = Calendar.getInstance();
    calendar.set(Calendar.HOUR_OF_DAY, hour);
    if(week == null) {
        notificationFinnish(calendar, true);
    }
    else {
        for (int temp:
             week) {
            calendar.set(Calendar.DAY_OF_WEEK, temp);
            notificationFinnish(calendar, false);
        }
    }
    Toast.makeText(mContext, "Notification set up", Toast.LENGTH_SHORT).show();
}
```

Figure 50. Sets multiple notifications if needed

```
private void notificationFinnish(Calendar calendar, boolean daily)
{
    Intent intent = new Intent(mContext, NotificationReceiver.class);

    PendingIntent pendingIntent = PendingIntent.getBroadcast(mContext, 100, intent,
PendingIntent.FLAG_UPDATE_CURRENT);

    AlarmManager alarmManager = (AlarmManager)
mContext.getSystemService(Context.ALARM_SERVICE);
    if(daily)
        alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(), AlarmManager.INTERVAL_DAY, pendingIntent);
    else alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(), weekMillis, pendingIntent);

}
```

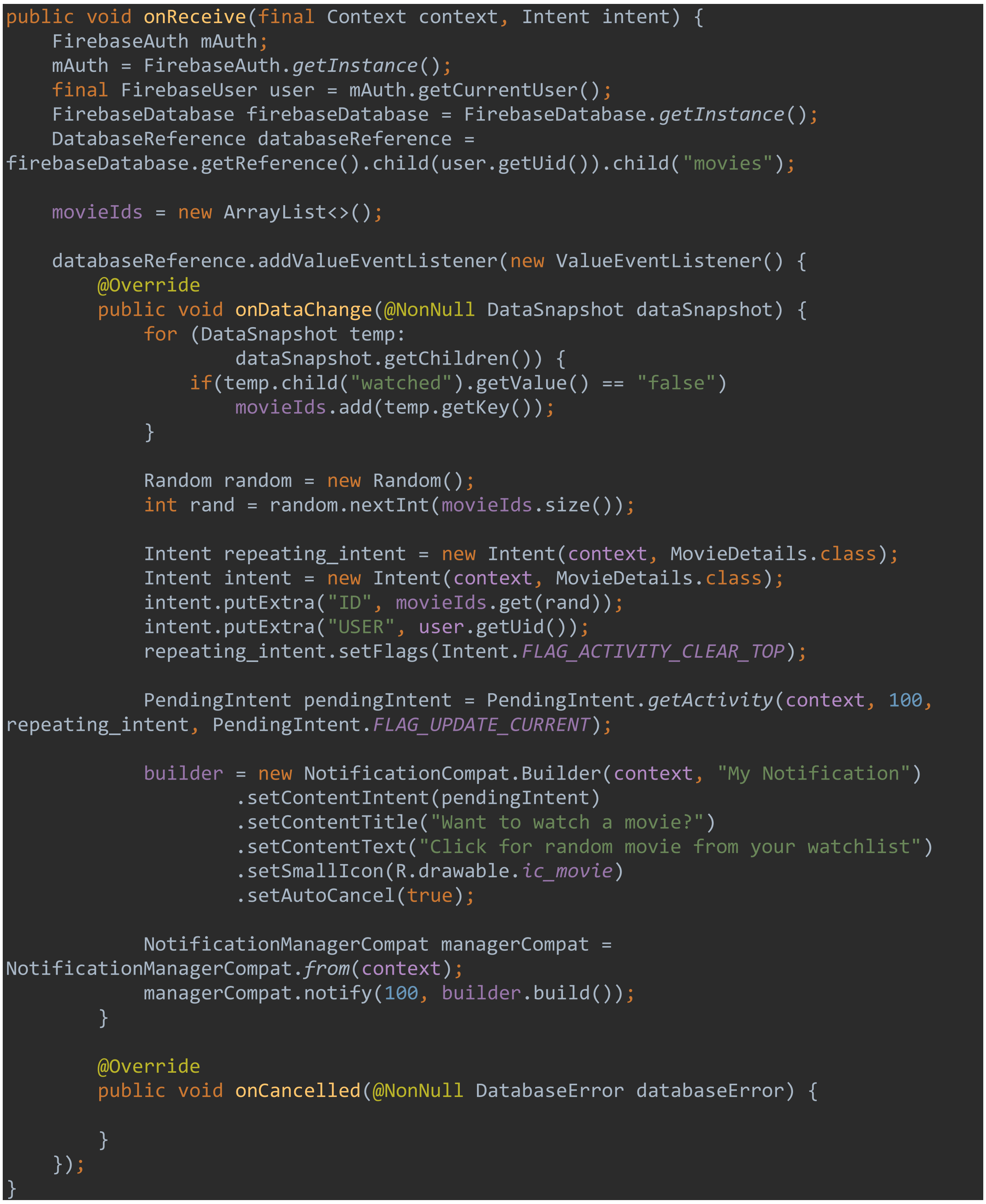
Figure 51. Creates notification alarms

```
private void cancelNotification() {
    if(!enabled)
        return;
    if(week == null) {
        cancelNotificationFinnish();
    }
    else {
        for (int temp:
                week) {
            cancelNotificationFinnish();
        }
    }
    Toast.makeText(mContext, "Notification disabled", Toast.LENGTH_SHORT).show();
}
```

Figure 52. Cancels multiple notifications if needed

```
private void cancelNotificationFinnish(){
    Intent intent = new Intent(mContext, NotificationReceiver.class);

    PendingIntent pendingIntent = PendingIntent.getBroadcast(mContext, 100, intent,
PendingIntent.FLAG_UPDATE_CURRENT);

    AlarmManager alarmManager = (AlarmManager)
mContext.getSystemService(Context.ALARM_SERVICE);
    alarmManager.cancel(pendingIntent);
}
```

Figure 53. Cancels specific notification



```java
public void onReceive(final Context context, Intent intent) {
    FirebaseAuth mAuth;
    mAuth = FirebaseAuth.getInstance();
    final FirebaseUser user = mAuth.getCurrentUser();
    FirebaseDatabase firebaseDatabase = FirebaseDatabase.getInstance();
    DatabaseReference databaseReference =
firebaseDatabase.getReference().child(user.getUid()).child("movies");

    movieIds = new ArrayList<>();

    databaseReference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for (DataSnapshot temp:
                    dataSnapshot.getChildren()) {
                if(temp.child("watched").getValue() == "false")
                    movieIds.add(temp.getKey());
            }

            Random random = new Random();
            int rand = random.nextInt(movieIds.size());

            Intent repeating_intent = new Intent(context, MovieDetails.class);
            Intent intent = new Intent(context, MovieDetails.class);
            intent.putExtra("ID", movieIds.get(rand));
            intent.putExtra("USER", user.getUid());
            repeating_intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

            PendingIntent pendingIntent = PendingIntent.getActivity(context, 100,
repeating_intent, PendingIntent.FLAG_UPDATE_CURRENT);

            builder = new NotificationCompat.Builder(context, "My Notification")
                    .setContentIntent(pendingIntent)
                    .setContentTitle("Want to watch a movie?")
                    .setContentText("Click for random movie from your watchlist")
                    .setSmallIcon(R.drawable.ic_movie)
                    .setAutoCancel(true);

            NotificationManagerCompat managerCompat =
NotificationManagerCompat.from(context);
            managerCompat.notify(100, builder.build());
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Figure 54. Creates notification display

Task #18. Use light sensor to check if it is dark or bright around user. If it is dark, then show suggestion for horror movie, if it is bright, then show suggestion for comedy movie.

To use light sensor activity has to implement SensorEventListener, which implements onDataChanged and onAccuracyChanged functions. onDataChanged function gets value of light sensor. When user wants to get suggestion, application uses value of light. If light value is above 20000, then user gets comedy movies, if value is below 2000, user gets horror movies.

```
@Override
public void onSensorChanged(SensorEvent event) {
    if(event.sensor.getType() == Sensor.TYPE_LIGHT){
        lightValue = event.values[0];
        //textView.setText("" + lightValue);
        }
}
```

Figure 55. Gets light value from sensor

```
private void getSensorSuggestion(){
    if(lightValue < 20000)
    {
        searchId = "27"; // horror
        changeRecyclerView("discover");
    }
    else {
        searchId = "35"; //comedy
        changeRecyclerView("discover");
    }
}
```

Figure 56. Decides what category of movies to show

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
5. https://www.youtube.com/watch?v=VQttXb6qE6k
6. https://www.simplifiedcoding.net/android-speech-to-text-tutorial/
7. https://www.youtube.com/watch?v=dfTeS41BbbI