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

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. Set up database.
2. Add ability to search for a movie
3. When clicking a movie, show more information about it
4. User can add a movie to watchlist
5. Be able to mark movie from watchlist as watched or remove it.
6. Check your current watchlist
7. Sort watchlist by alphabet, date, rating, etc.
8. Add progress bar to watchlist, that shows how many movies from watchlist has been watched

Solution

Task #1. Set up database

As a database app uses *firebase* services. *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, app 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 movies is watched (Figure 1).

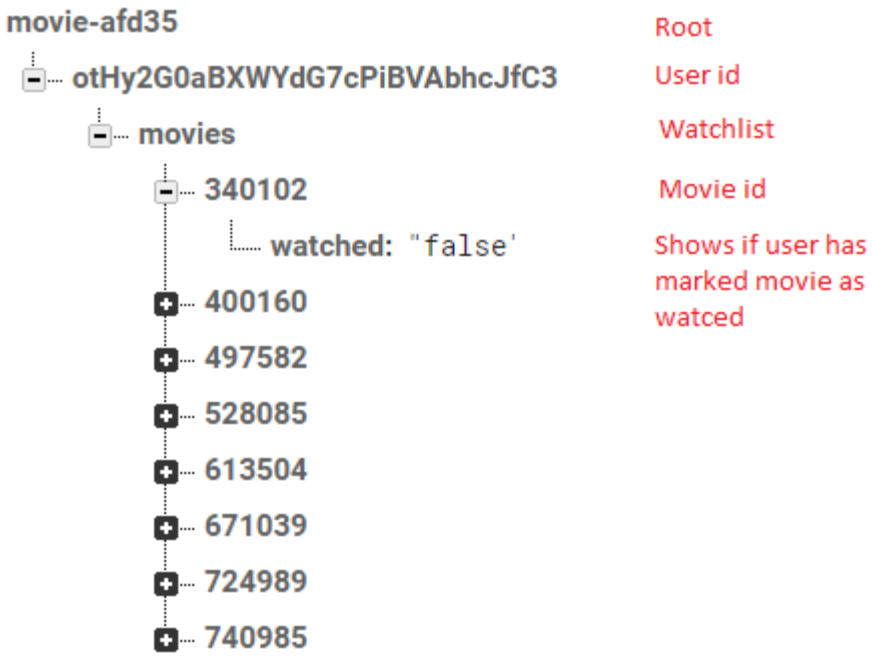

Figure 1. Firebase database, a way of storing users watchlist

Task #2. 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 2). 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.

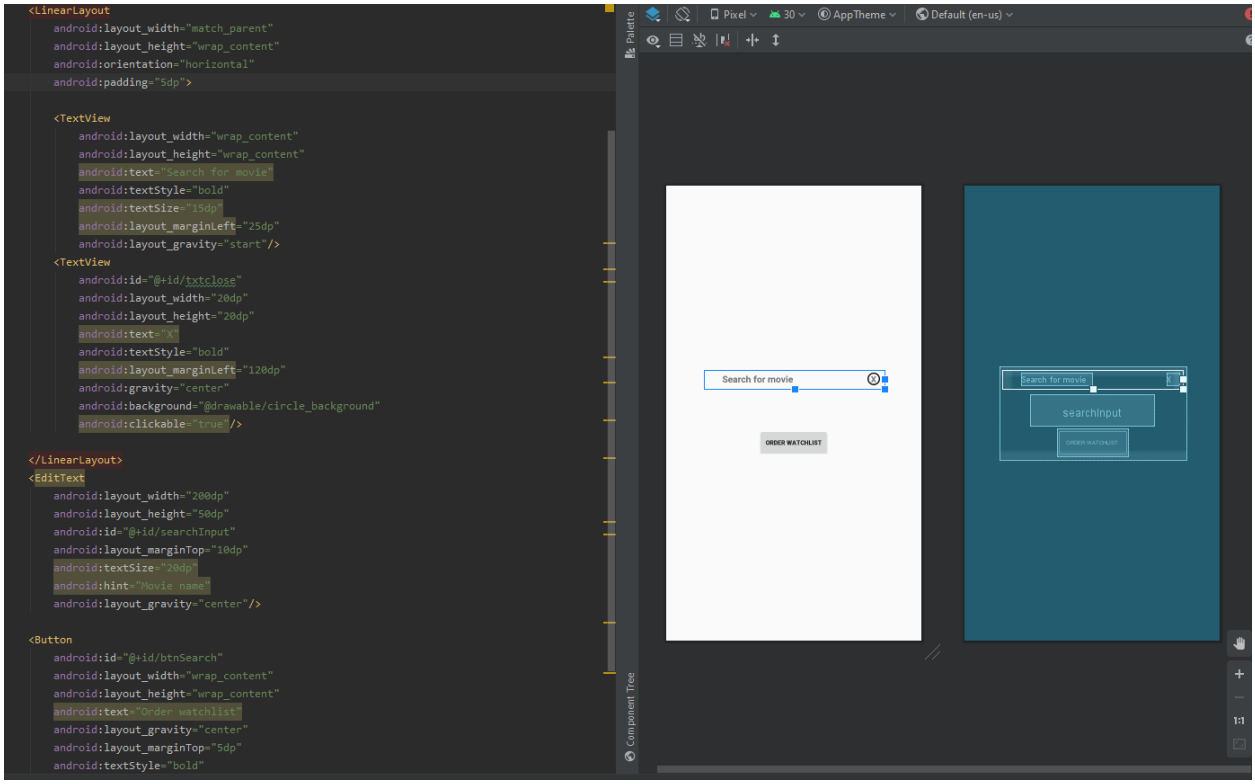

Figure 2. Dialog prompt to search for movies

```
public boolean searchPopUp(View v){
    final Dialog searchDialog = new Dialog(this);
    TextView txtclose;
    final EditText inputText;
    final Button searchBtn;
    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 3. Code for activating search

Task #3. When clicking a movie, show more information about it

In movie adapter each movie item gets a new on click listener (Figure 4), 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 4. Code for going to new intent 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 5)

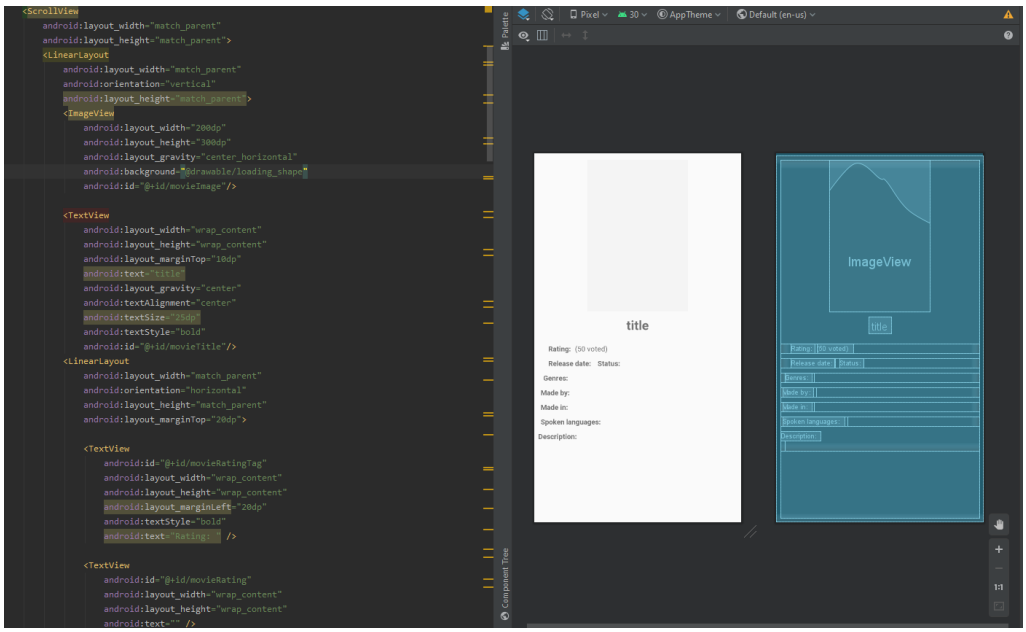

Figure 5. xml file for movie details layout

For getting information from movie database I used the same principle of getting information for most popular movies, top rated, etc (Figure 6). Just needed to create new service type for calling information. After getting needed information it is displayed on the phone (Figure 7).

```
private void getMovieDetails()
{
    String id = intent.getStringExtra("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) {
            details = response.body();
            fillInfo();
        }

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

Figure 6. Getting information from movie database

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

Figure 7. Service call to get information about a movie

Figure 8. Result of movie detail display

Task #4. 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 4). When trying to add movie to watchlist if user id is not empty button for adding movie to watchlist is displayed (Figure 8). To add movie in watchlist app needs database exact point where it needs to write in database (Figure 9). 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 10). 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 =
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 9. Code for setting up buttons

```
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 10. Code for adding to or removing from movie watchlist

Task #5. 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 9).

1. Check your current watchlist
2. Sort watchlist by alphabet, date, rating, etc.

Task #6. 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);

                            System.out.println(info.isWatched() + " po");
                            baseMovieList.add(info);

                            i = i -1;
                            if(i <= 0) {
                                pd.dismiss();
                                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 11. Code for getting information about user watchlist

Task #7. Sort watchlist 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 12), 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.

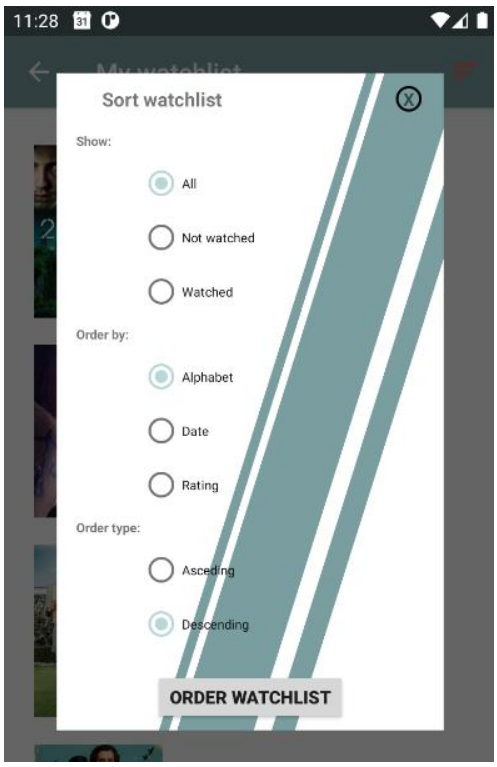

Figure 12. Dialog for sorting watchlist

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

```
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 13. 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 14. Base sorting function, which determins 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 15).

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

Figure 15. Check is movie is watched

To sort alphabetically it compares title strings (Figure 16). 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 17). Sorting by rating is the same as sorting by release date (Figure 18).

```
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 16. Sorts movies alphabetically

```
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 17. Sorts movie by release date

```
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 18. Sorts movie by user rating

Task #8. Add progress bar to watchlist, that shows how many movies from watchlist has been watched

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).

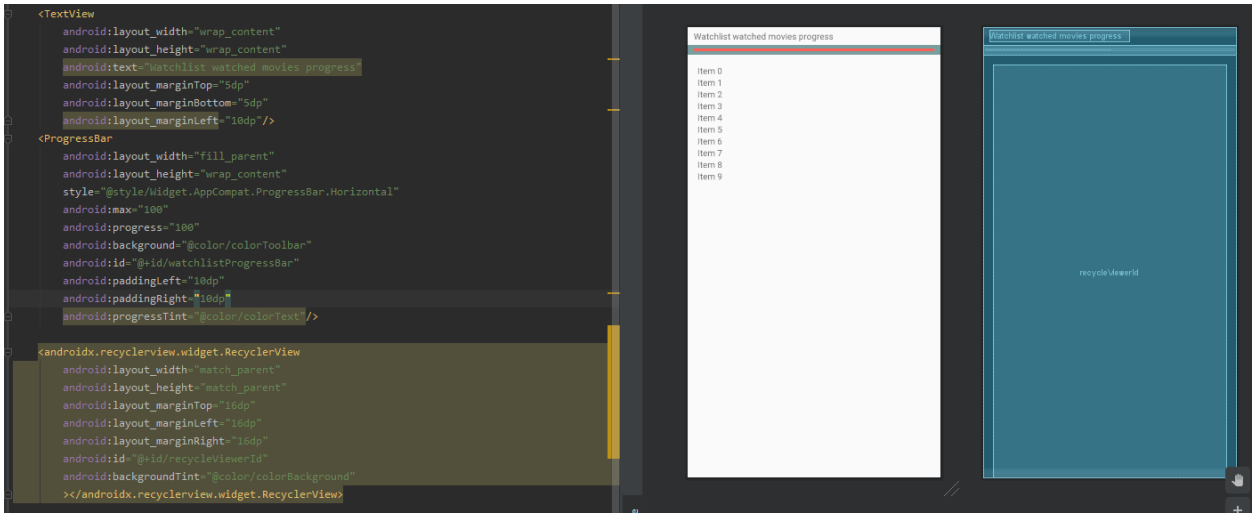

Figure 19. Watchlist layout with progress bar

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

Figure 20. Progress bar value counting