Technical Specification Document

Anas Alagtal

Movie Center

14511433

18/May/2019

Table of contents

0. Abstract

This project involved building a mobile android application that would allow users to explore movies. A python recommender system is implemented that would retrieve a list of the preferred movie for users to display them within the application.

1 Introduction

1.1 Overview

Watching movies is a very common hobby in modern day with the availability of online streaming platforms. These platforms can be enhanced with the presence of applications that displays all the necessary information regarding past and upcoming movies. This is what “Movie Center” provides.

Movie Center is an android application built to promote the exploration of new movies for users. It pulls different genres of movies from “The movie Db” through their API and displays them within the application.

Users can register with a valid email and password from within the application to be able to use its features. Once logged in users can browse movies from the available tabs; Upcoming movies, Now playing movies, Popular movies and Recommended movies. The tabs will contain movies in a scrollable view to allow users to explore more then what's originally displayed.

Clicking into a movie poster will lead to a movie details page that displayed further information regarding a movie and allows the user to rate a movie out of 7. If the app sees that the users rating means that he enjoyed the movie, it will push that movie to firebase. Every time a user updates that database with a new movie, a python script reads the database and updates a recommended list in the base with a list of movies. The movies are found once running the users “enjoyed” movies with a dataset of over 2000 popular movies from 1990 onwards. These recommended movies are then displayed in the Recommended movies tab.

1.2 Glossary

Java: Object-oriented programming language used to write the functionalities of the Android application

XML: Stands for Extensive markup language. A self-descriptive language that Android studio incorporates for design and display of items in the user interface in android.

Python: High-level programming language that was used to develop the recommender system

Firebase: Mobile development application platform. Used for my user authentication and database.

Android Studio: Googles Integrated development environment for the Android operating system.

API: Application program interface. It is a set of functions that allow the creation of an application through the access of functions or data from a service.

GUI: Graphical User Interface. An interface that allows the user to interact with an application through the use of icons and images rather than text and commands.

The movie Db: A third party website that offers an API which allows access to a full range of movies database and information regarding these movies.

Google Cloud Platform: Cloud computing service for hosting applications.

JSON: (JavaScript Object Notation) is a lightweight data-interchange format that uses human-readable text to transmit data objects

2 General description

2.1 Motivation

From applications that are already out there, IMDb would be considered the most popular. However, in my opinion, it is more complex than it needs to be. It contains a lot more information then what people usually want to know like new about certain actor and interviews etc. It lacks the simplicity that I wanted from a movie application that is supposed to display movie information.

Recommendations within IMDb are not also up to the standard I would have wished. If you enjoy a movie, it will recommend other movies where the main actor has also featured in. It does not take an overall view of your preferences.

That's where the inspiration to create my own application with a simplistic UI and a tailored recommended system came from.

2.2 Research

Seeing as most of these sort of applications are rarely used in their web app for, I decided to implement the idea through android studio so that it can be used in android mobile devices. This left me little choice in languages as XML is the default markup language for the UI and the code will be either in JAVA or Kotlin and seeing as I know Java, that is the language I decided to use.

Doing some research into recommender systems, I realised python would be the best language in implementing it. It has some very powerful libraries to easily read in CSV files, write to CSV file, pull JSON data from an API, read and write from a database, calculate the similarity between strings, etc. Python is also a language I'm familiar with so I decided to go with it.

Once I figured the "what", I needed to figure out the "how" here are some examples:

Android Studio: This was the main point of research for me. I have never built an application in android before. In fact, I've never built a mobile application. Extensive research was done both at the beginning and throughout the duration of the project. The emulator which I was to use to run the project on needed installation along with some configuration in the BIOS settings. XML was the language needed to create,

design and display different features of the interface including background images, buttons, and text view.

Firebase: Firebase was also totally new to me. I researched a bit into it for my proposal and seen how powerful its features was especially within android so I decided to use it. Once implementation began I had to find out how to incorporate the connection of the database both within my application using java and to my recommender system in python.

2.3 Business Context

This application is targeted really to a market involving anyone who is old enough to use a smartphone and has any interest in movies. It will, of course, enhance any users movie watching experience. It is an android application so can only be used on Android smartphones by downloading from the google play store.

Revenue regarding the application could be generated by directly selling the application at a cost or by charging fees for promoting various upcoming movies.

2.4 Product / System Functions:

Once the application is downloaded and installed into the Android device, the user then launches the application and registers within the application. Once registered, the user is logged in and their user information is recorded into my database. They can then use any feature within the application. The user can then browse a range of movies whether they are upcoming movies, movies now playing in the cinemas, popular movies, they could read more into a movie, give their own rating into a movie or even view a set of tailored recommended movies.

2.5 User Characteristics

The target audience is very broad. The application is intended for all users who are able to operate an android mobile device.

The user needs to have basic knowledge of how to install an android application. The entire application including title, plot, trailers, etc will be in English so they need to be familiar with the language. They would need to have a personal email to be able to register or log in. From then on everything should be straightforward.

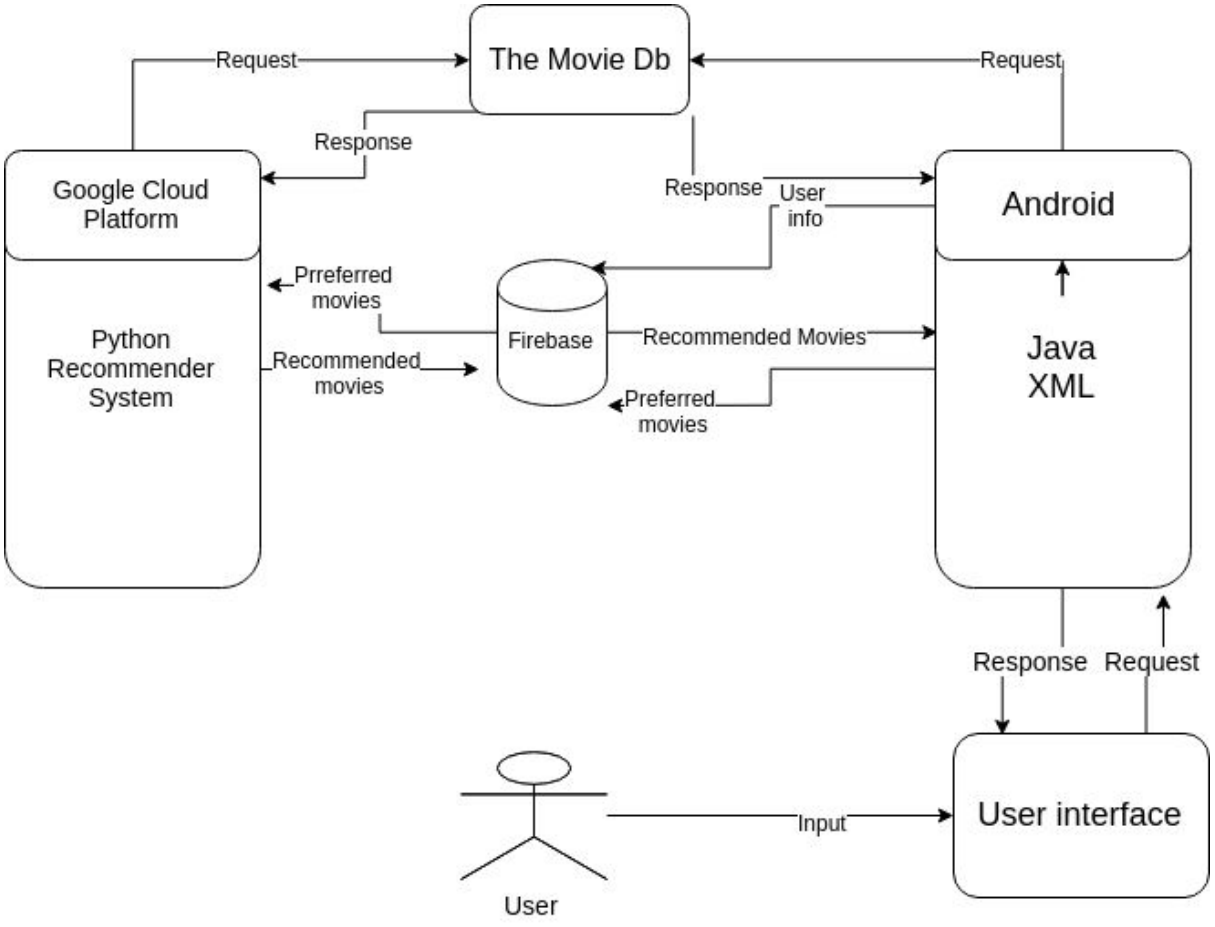
2.6 User Objectives

The objective from the user point of view would be a simple app with easy access and easy navigation. The app should display movies in an organized and clear manner. Information regarding any movie would be easily attainable. A returned list of recommended movies should be fairly accurate to what the user sees as enjoyable.

3. Design

Fortunately, the application did not require any great deal of modification from the functional spec. There were very minor changes I decided not to implement but were justified. Logging in through Google was not implemented seeing as you were required to have an email regardless. Writing a review was not implemented seeing as users can enter a rating. Good reviews are accompanied with good rating and vice versa therefore having both together was not need and it is a lot easier for the user to rate a movie rather than give a review. This being along with the fact that reviews are seldom written lately.

3.1 System Architecture

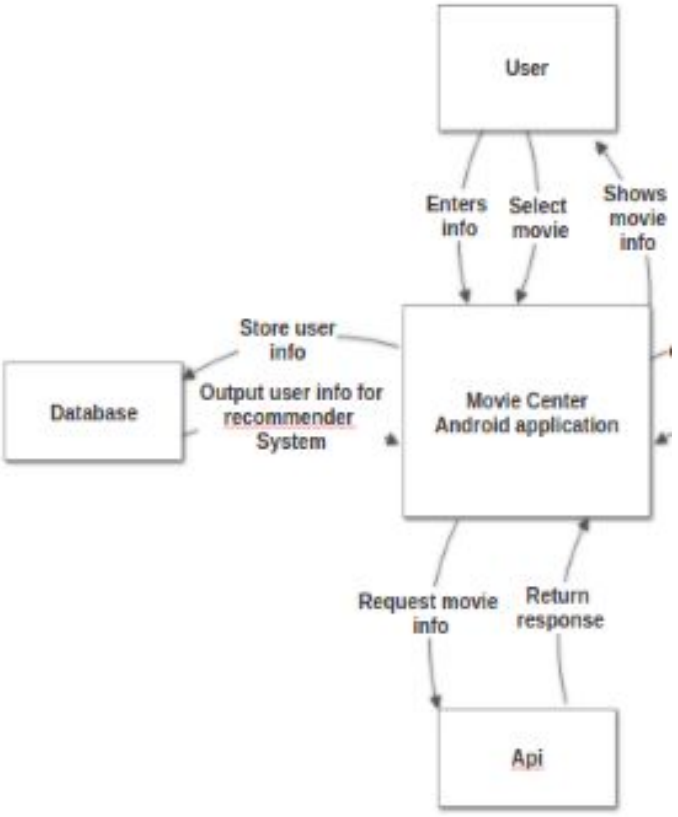


Like said precious, Movie center is an android application implemented to display movies and their information. Once a user registers, their information is stored in firebase. While doing so the movie details are being pulled from The Movie Db API. The user can interact with the GUI to add or remove movies from his/her preferred list of movies. Anytime the database is updated with a new preferred movie, a python

script that is constantly running on the Google Cloud Platform is triggered to create a new list of recommendations. This is done by running this list of preferred movies across a CSV file containing the most popular movies from 1990 till today pulled for The movie Db. The result of this system i.e a list of recommended movies is then pushed onto the firebase database. This list is pulled from within the application and displayed to the user.

3.2 High-Level Design

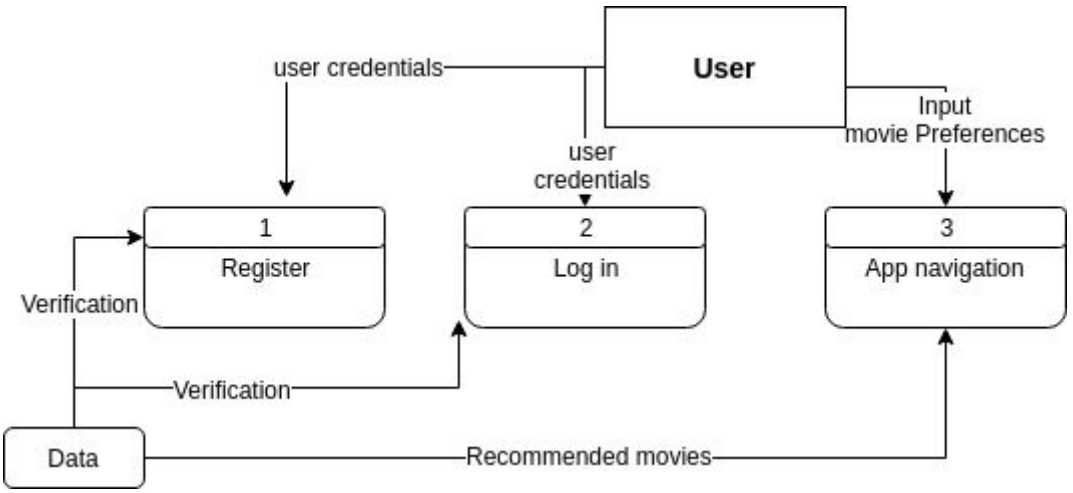
The following diagram illustrates the high-level design of the system and its functionality.



Once the user launches the application, He is prompted to register or log in after which his details are stored in the firebase database. A thread running in the background automatically pulls information from “The Movie Db” through their API. These movies are then displayed and can be rated by the user. Any movie that the user rates well will be pushed onto the database. A trigger in the python script allows the recommended movie list in the database to be updated any time the user rates a new movie. Another thread running in the background will pull the recommended movie list and display it in the recommended tab.

3.2 Data Flow Diagram

Below is a first level data flow diagram.



4 Implementation and Sample Code

4.1 Java

The functionalities of the Android application were built in Java. Android only allows for Java and Kotlin but I chose not to use Kotlin as I am inexperienced in the language which would require extra time for research. Here are a few examples of what I implemented in Java:

User Authentication:

The first thing was the login and registration pages. Using XML, I created a template login and registration activities with button and text views. The java classes read the input taken from the user for the email and password variables during registration. Using a built-in email checker checked whether the email was of the correct format. Using Firebase authentication, I checked whether the email entered was a valid email. A small function would check whether the password enter contained enough characters. As the app is not one that requires ultimate confidentiality, I allowed for passwords to contain any sequence of characters as long it is of length 6 or greater.

```
private void attemptLogin() {

    String email = editTextEmail.getText().toString().trim();
    String password = editTextPassword.getText().toString().trim();


    if (email.isEmpty()) {
        editTextEmail.setError(getString(R.string.error_field_required));
        editTextEmail.requestFocus();
        return;
    }
    if(!isEmailValid(email)){
        editTextEmail.setError(getString(R.string.error_invalid_email));
    }
    if (password.isEmpty()) {
        editTextPassword.setError(getString(R.string.error_field_required));
        editTextPassword.requestFocus();
        return;
    }
    if(!isPasswordValid(password) ){
        editTextPassword.setError(getString(R.string.error_invalid_password));
        editTextPassword.requestFocus();
        return;
    }
    progressBar.setVisibility(View.VISIBLE);

    mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            progressBar.setVisibility(View.GONE);
            if(task.isSuccessful()){
                Intent intent = new Intent(LoginActivity.this, HomeActivity.class);
                //intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                startActivity(intent);
            }else {
                Toast.makeText(getApplicationContext(), task.getException().getMessage(), Toast.LENGTH_SHORT).show();
            }
        }
    });


}
```

Network Request:

I created a network request class to grab URL information. Within this is class is a “fetchMovieData” method that takes in the “The Movie Db” URL with the API and attempts an HTTP Request to grab the JSON Information from the site. It calls another function that creates a URL Object from a URL String. The result is a JSON object containing movie information from that request.

```
    //Create URL format
    private static URL createURL(String stringURL) {
        URL url = null;
        try {
            url = new URL(stringURL);
        } catch (MalformedURLException e) {
            Log.v(LOG_TAG, "Error: Problem occurred in creating URL");
        }
        return url;
    }

    /**
     * Fetch Movie Details
     * @param requestJsonUrl Json URL
     * @return List of movie objects
     */
    static List<MovieDetails> fetchMovieData(String requestJsonUrl) {

        String jsonResponse = "";

        URL url = createURL(requestJsonUrl);

        try {
            jsonResponse = makeHttpRequest(url);
        } catch (IOException e) {
            Log.v(LOG_TAG, "Error: Problem occurred in fetching movie details through the request");
        }

        return JsonMovieDetails.extractJsonMovieData(jsonResponse);
    }


}
```

Extracting Movie information

Before Displaying movie information, each movie needs to be extracted separately along with its details. To get the JSON object from the network request class, I created an Async thread that would work in the background to call and retrieve the JSON info. Once done so, I iterate through the JSON to find the Tittle, Release date, Poster path, etc from each movie.

```
public class MovieDetailsLoader extends AsyncTaskLoader<List<MovieDetails>> {

    private final String url;

    public MovieDetailsLoader(Context context, String url) {
        super(context);
        this.url = url;
    }

    @Override
    protected void onStartLoading() {
        forceLoad();
    }

    @Override
    public List<MovieDetails> loadInBackground() {
        if (url == null) return null;
        return NetworkRequest.fetchMovieData(url);
    }
}
```

```
   try {
       JSONObject base = new JSONObject(json);

       JSONArray resultsArray = base.getJSONArray(KEY_RESULTS);

       for (int i = 0; i < resultsArray.length(); i++) {

           JSONObject object = resultsArray.getJSONObject(i);

           if(object.has(KEY_ID)){
               id = object.optString(KEY_ID);
           }
           if (object.has(KEY_TITLE)) {
               title = object.optString(KEY_TITLE);
           }
           if (object.has(KEY_RELEASE_DATE)) {
               releaseDate = object.optString(KEY_RELEASE_DATE);
           }
           if (object.has(KEY_POSTER_PATH)) {
               poster = object.optString(KEY_POSTER_PATH);
           }
           if (object.has(KEY_VOTE)) {
               vote = object.optString(KEY_VOTE);
           }
           if (object.has(KEY_OVERVIEW)) {
               overview = object.optString(KEY_OVERVIEW);
           }

           MovieDetails movieDetails = new MovieDetails(id, title, releaseDate, poster, vote, overview);

           movieDetailsList.add(movieDetails);
       }

   } catch (JSONException e) {
       Log.v(LOG_TAG, "Error: Problem occurred during extracting Json MovieDetails Data");
   }

   return movieDetailsList;
```

Displaying Movie Info

To display movie information, I have a Fragment class for each tab. A movie list containing each movie is attached to a recyclerview to display it one by one. A column fitting class takes the posters and displays them 3 in each row.

```
@Override
public void onBindViewHolder(MoviesHolder holder, int position) {
    MovieDetails movieDetails = moviesList.get(position);
    if(movieDetails.getPoster().isEmpty() || movieDetails.getPoster().equalsIgnoreCase("null")){
        holder.iv_movie_image.setImageResource(R.drawable.no_poster_available);
    }else{
        String createPosterPath = POSTER_PATH + movieDetails.getPoster();
        Context context = holder.iv_movie_image.getContext();
        Picasso.with(context).load(createPosterPath).into(holder.iv_movie_image);
    }
}
```

```
public class ColumnFitting {

    public static int calculateNoOfColumns(Context context) {
        DisplayMetrics displayMetrics = context.getResources().getDisplayMetrics();
        float dpWidth = displayMetrics.widthPixels / displayMetrics.density;
        return (int) (dpWidth / 120);
    }

}
```

Storing Preferred movie list

Once a user rates a movie, a method “on rating changed” that listens to any change in rating gets called and checks the rating of the user. If it is greater than 4/7 then the movie pushed to the database.

```
    rating_bar.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {
        @Override
        public void onRatingChanged(RatingBar ratingBar, float v, boolean b) {
            float x = ratingBar.getRating();
            if(ratingBar.getRating() > 4){
                Toast.makeText(getApplicationContext(),String.valueOf(x),Toast.LENGTH_LONG).show();
                myRef.child(movieDetails.getTitle()).setValue(movieDetails.getTitle());
            }else{
                Toast.makeText(getApplicationContext(),String.valueOf(x),Toast.LENGTH_LONG).show();
                myRef.child(movieDetails.getTitle()).removeValue();
            }
            saveRating();
        }
    });

    retrieveRating(); //Restore the rating if some
}
```

Retrieving Recommended Movies list

```
//Listener to retrieve and keep up to date with any changes in the Firebase database to update
private void attachDatabaseReadListener(){
    recommendedMovieList.clear();
    if(childEventListener == null){
        childEventListener = new ChildEventListener() {
            @Override
            public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String s) {
                //recommendedMovieList.clear();
                // Get the data of the new insertion in the parameter, it will deserialize the data from the database
                //It does that because the fields matches what is in the database
                Recommended recommended = dataSnapshot.getValue(Recommended.class);
                recommendedMovieList.add(recommended);
                adapter.notifyDataSetChanged();
            }
            @Override
            public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable String s) { }
            @Override
            public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) {}
            @Override
            public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable String s) {}
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) { }
        };
        databaseReference.addChildEventListener(childEventListener);
    }
}
```

4.2 Python

Preparing data:

To be able to recommend movies I needed to pull a range of movies to recommend from. In Python, it was much simpler then JAVA. I used the requests library in python to extract the JSON from the API. Firstly I created a CSV named “movie.csv”. I then used the JSON from the API request to iterate through and store each move and the relevant data to store in the CSV

```
def updateDF():
    df = createCSV()
    api_key = "3a99e9b91f001ce52e943ec17f668e45"

    for year in range(1990, 2020):

        for j in range(1, 4):
            response = requests.get('https://api.themoviedb.org/3/discover/movie?api_key=' +  api_key + '&append_to_response=credits&primary_release_ye
            #print (response.json())

            jsonData = response.json() # store parsed json response
            #print (jsonData)

            #https://api.themoviedb.org/3/discover/movie?api_key=3a99e9b91f001ce52e943ec17f668e45

            jsonData_films = jsonData['results']

            # for each of the highest revenue films make an api call for that specific movie to return the budget and revenue
            for film in jsonData_films:
                # print(film['title'])
                film_revenue = requests.get('https://api.themoviedb.org/3/movie/'+ str(film['id']) +'?api_key='+ api_key+'&append_to_response=credits&l
                #print (film_revenue.json())
                film_revenue = film_revenue.json()
                #print(locale.currency(film_revenue['revenue'], grouping=True ))
                #df.loc[len(df)]=[film['title'],film_revenue['revenue'], film_revenue['overview'], film_revenue['genres']] # store title and revenue in
                genres = []
                actors = []
                #print (film_revenue['genres'])
                for i in range (len(film_revenue['genres'])):
                    genres.append(film_revenue['genres'][i]['name'])

                for i in range (len(film_revenue['credits']['crew'])):
                    if (film_revenue['credits']['crew'][i]['job'] == 'Director'):
                        director = (film_revenue['credits']['crew'][i]['name'])


                if (len(film_revenue['credits']['cast']) < 4):
                    continue
                else:
                    for i in range (4):
                        actors.append((film_revenue['credits']['cast'][i]['name']))

                df.loc[len(df)]=[film['title'],','.join(genres),director,  ','.join(actors), film_revenue['overview'], film['id'], film['release_date']
    return df
```

I then removed any movies that were missing any information because they would not be recommended correctly and finally wrote the data frame to the CSV file.

```
def removeEmptyCells():
    df = updateDF()
    #replace any empty strings in the column with np.nan object
    df['Genre'].replace('', np.nan, inplace=True)
    df['Director'].replace('', np.nan, inplace=True)
    df['Plot'].replace('', np.nan, inplace=True)
    df['Actors'].replace('', np.nan, inplace=True)
    #drop the null values i.e all rows:
    df.dropna(subset=['Genre'], inplace=True)
    df.dropna(subset=['Director'], inplace=True)
    df.dropna(subset=['Plot'], inplace=True)
    df.dropna(subset=['Actors'], inplace=True)

def writeToCSV():
    df = removeEmptyCells()
    with open('movies.csv', 'a') as f:
        df.to_csv(f, header = False)
    #df.to_csv(r'test.csv')
```

Recommender class:
The following recommender class would first take the “movies.csv” file to prepare the data. It would only take the Title, Genre, Directors, Actors and Plot columns from the data. It would convert the genres to a list of genres e.g Crime, Drama would go to [Crime, Drama]. It would convert the director's name from first name second name to one full string and no spaces. This is to reduce the false increase of similarity between two movies just because the director's first names are the same. E.g Martin Scorsese to MartinScorsese.

```
def dataPreProcessing():

    df = pd.read_csv('movies.csv')
    #df = pf.read_csv('PreProcessingTest.csv')
    df = df[['Title','Genre','Director','Actors','Plot']]

    #discarding the commas between the actors' full names and getting only the first three names
    df['Actors'] = df['Actors'].map(lambda x: x.split(',')[:4])

    #putting the genres in a list of words
    df['Genre'] = df['Genre'].map(lambda x: x.lower().split(','))

    df['Director'] = df['Director'].map(lambda x: x.split(' '))

    for index, row in df.iterrows():
        row['Actors'] = [x.lower().replace(' ','') for x in row['Actors']]
        row['Director'] = ''.join(row['Director']).lower()

        # initializing the new column
        df['Key_words'] = ""

    return df
```

I used the Rake() function from NLTK library in python to extract keywords from the plot. Words like “the” and “and” would be removed from the cells. This is to give a better more accurate similarity score based on good words and not common stop words. I then create a new column that would combine the genre, director, actors and plot before dropping the columns and leaving the one big one called “movieDetails”. I then instantiate a matrix of each movie across each movie and find the cosine similarity between each of them with each of the other. I then return the matrix to be used for recommendations.

```
def similarityFunction():
    df = dataPreProcessing()
    for index, row in df.iterrows():
        plot = str(row['Plot'])

        # instantiating Rake, by default it uses english stopwords from NLTK
        # and discards all puntuation characters as well
        #print plot
        r = Rake()

        # extracting the words by passing the text
        r.extract_keywords_from_text(plot)
        # getting the dictionary whith key words as keys and their scores as values
        key_words_dict_scores = r.get_word_degrees()
        row['Key_words'] = list(key_words_dict_scores.keys())
    #print (key_words_dict_scores)

    # dropping the Plot column
    df.drop(columns = ['Plot'], inplace = True)

    df.set_index('Title', inplace = True)
    df['movieDetails'] = ''
    columns = df.columns
    for index, row in df.iterrows():
        words = ''
        for col in columns:
            if col != 'Director':
                words = words + ' '.join(row[col])+ ' '
            else:
                words = words + row[col]+ ' '
        row['movieDetails'] = words

    df.drop(columns = [col for col in df.columns if col!= 'movieDetails'], inplace = True)
    # instantiating and generating the count matrix
    count = CountVectorizer()
    count_matrix = count.fit_transform(df['movieDetails'])

    indices = pd.Series(df.index)

    # generating the cosine similarity matrix
    global cosine_sim
    cosine_sim = cosine_similarity(count_matrix, count_matrix)
    return df, indices
```

The recommendations method take in a list of movie names in as a parameter. It goes through the list of movies that user would enter as their preferred movies one by one and compare the movie with the rest of the dataset. The top 10 movies and their similarity scores are stored in a dictionary. Once the iteration through every preferred movie is complete, I return the dictionary in order of rating. This returns a list of movies best suited to the overall list of movies rather than return a set of movies recommended to one movie than the other then the other etc.

```
def recommendations(titles):

    df, indices = similarityFunction()
    allMovieDict = {}
    topDict = {}

    recommended_movies = []
    top_indexes = []


    for title in titles:
        existing = False
        for i in indices:
            if (title == i):
                existing = True
        if (existing == True):
            # gettin the index of the movie that matches the title
            idx = indices[indices == title].index[0]

            # creating a Series with the similarity scores in descending order
            score_series = pd.Series(cosine_sim[idx]).sort_values(ascending = False)

            for x, y in score_series.items():
                allMovieDict[x] = y

            # getting the indexes of the 10 most similar movies
            top_10_indexes = list(score_series.iloc[1:11].index)
            #print (top_10_indexes)
            for i in top_10_indexes:
                topDict[i] = allMovieDict[i]
                top_indexes.append(i)
            #print (topDict)
            #print top_indexes
            # populating the list with the titles of the best 10 matching movies
            for i in top_10_indexes:
                #print (df.index[i])
                topDict[df.index[i]] = topDict.pop(i)
                recommended_movies.append(list(df.index)[i])

            #topDict[list(df.index)[i]] =
    return (topDict)
```

Database Listener

A database class connects to firebase and is triggered whenever a new movie is added. The recommended child to the database is updated with a new list containing a list of movie that was the result of calling the recommendations method again with the new movie.

```
def stream_handler(message):
    recomm = {}
    movieDict = {}
    favMovies = []
    nodeExists = True

    print("******" , message)
    try:
        MovieDict = db.child().get().val()['Movies']
    except:
        nodeExists = False

    if(nodeExists == True):
        for x, y in MovieDict.items():
            favMovies.append(y)

        recomm = recommendations(favMovies)

        for movie, score in recomm.items():
            id = (df2.pipe(lambda item: item[item['Title'] == movie]['id']))
            id = id.__str__().split()[1]
            recomm[movie] = id
        print (recomm)

        db.child("Recommended").remove()

    # for movie, id in recomm.items():
    #     try:
    #         db.child("Recommended").child(movie).set(id)
    #     except:
    #         print ("Error", movie)
        for movie, id in recomm.items():
            testDict = {}
            testDict["title"] = movie
            try:
                db.child("Recommended").push(testDict)
            except:
                print ("Error", movie)

my_stream = db.child("Movies").stream(stream_handler,None)
```

5 Problems and Resolutions

Pulling data: When pulling information for my “movies.csv” some data was coming our broken. A lot of data was missing and some movies were in Chinese. This caused my program to stop running whenever it reached a movie of this sort. I fixed this by choosing the English language in the API URL, choosing only the most popular 40 movies of each year and removed any movie that was missing any data.

Databse trigger: Although I knew that I must get a trigger working to call a recommender function, I did not know how. I was going to have my python code on the Django or Flask framework to connect to android so that the user rating a movie would trigger it but then I found the Pyrebase library in python which allows for triggers like that. Now, my anytime the database is updated the python system runs.

Displaying posters: When I retrieved the JSON information in Android, It was displaying one by one covering the entirety of the screen. I found a column fitting method online that walked me through how to display it in sections. I ended up displaying the movies ⅛th e size of the original now they display 2 in each row and 3 rows in each page which is scrollable.

Displaying Recommended Movies: Towards the end of the project, I was working on trying to display the posters of the recommended movies. Unfortunately, I ran into many errors with little time to fix. So I resolved this by simply displaying a list of movies.

These were the major problems that I had to overcome throughout the project. Other than that the problems were minor ones.

6. Validation and Testing

6.1 UI Testing

Tests in android were written to see if parts of the UI worked as it should. I tested wether the main activity launches upon clicking into the app. I used the espresso library to see if clicking a button, text view etc brings you to where it is supposed to. I tested the fragments (Tabs) to see if they displayed once entering the home Activity like they should. You can find examples of these tests are in the src/app/src/androidTest/java/com/example/moviecentre folder.

This first test is to see if the LoginActivity is displayed upon launching the application

```
@Rule
public ActivityTestRule<LoginActivity> mActivityTestRule = new ActivityTestRule<LoginActivity>(LoginActivity.class);

private LoginActivity mLoginActivity = null;


//testing if sign up texview bring you to sign up activity
Instrumentation.ActivityMonitor monitor = getInstrumentation().addMonitor(SignUpActivity.class.getName(), null , false);

@Before
public void setUp() throws Exception {
    mLoginActivity = mActivityTestRule.getActivity();
}

@Test
public void testLaunch(){
    View view = mLoginActivity.findViewById(R.id.email_sign_in_button);
    assertNotNull(view);
}
```

This test is to see if clicking the signUp text view launches the signUp Activity

```
@Test
public void testLaunchOfSignUp(){
    assertNotNull(mLoginActivity.findViewById(R.id.textViewSignup));
    onView(withId(R.id.textViewSignup)).perform(click());

    //wait 5 seconds until an activity is launched else return null
    Activity signUpActivity = getInstrumentation().waitForMonitorWithTimeout(monitor, 5000);

    assertNotNull(signUpActivity);

    signUpActivity.finish();
}
```

This test is to see if the HomeActivity is launched when the user clicks the log in button

```
@Test
public void testLaunchOFHomeActivity(){
    assertNotNull(mLoginActivity.findViewById(R.id.email_sign_in_button));
    onView(withId(R.id.email_sign_in_button)).perform(click());

    //wait 5 seconds until an activity is launched else return null
    Activity HomeActivity = getInstrumentation().waitForMonitorWithTimeout(monitor, 5000);

    assertNotNull(HomeActivity);

    HomeActivity.finish();
}
```

6.2 Unit Tests

I wrote some unit tests using the JUnit framework to test different functionalities of the application and different methods. Example of these tests are the ones write in the Login and Sign Up classes. I tested whether the email and password field accepted what they are supposed to accept and reject what they are supposed to reject. All their tests passed.

```
@Test
public void testLaunchOFHomeActivity(){
    assertNotNull(mLoginActivity.findViewById(R.id.email_sign_in_button));
    onView(withId(R.id.email_sign_in_button)).perform(click());

    //wait 5 seconds until an activity is launched else return null
    Activity HomeActivity = getInstrumentation().waitForMonitorWithTimeout(monitor, 5000);

    assertNotNull(HomeActivity);

    HomeActivity.finish();
}
```

I tested the entire NetworkClass function by testing the fetchMoveData method. It returns true if an instance of a JSON object s returned once running the class. It returns false if it catches an error.

```
@Test
public void testFetchMovieDataTrue() {
    List<MovieDetails> movieList = NetworkRequest.fetchMovieData("https://api.themoviedb.org/3/movie/now_playing?api_key=3a99e9b91f001ce52e943ec17f668e4
    MovieDetails md = movieList.get(0);
    assertTrue(md instanceof MovieDetails);
}
@Test(expected = NullPointerException.class)
public void testFetchMovieDataFalse() {
    List<MovieDetails> movieList = NetworkRequest.fetchMovieData("https://api.themoviedb.org/3/movie/now_playing?api_key=8484848");
    MovieDetails md = movieList.get(0);

}
```

I also test the recommender system using the unittest library in python. I tested the pdf data processing method in the following manner:

```
def test_PreProcessingMethod_EQUAL(self):


    expected = pd.DataFrame(
                {
                'Title':
                    ['GoodFellas'],
                'Genre':
                    [['drama', 'crime']],
                'Director':
                    ["martinscorsese"],
                'Actors':
                    [['rayliotta', 'robertdeniro', 'joepesci', 'lorrainebracco']],
                'Plot':
                    ["The true story of Henry Hill, a half-Irish, half-Sicilian Brooklyn kid who is adopted
                'Key_words':
                    [""]
                })

    input = recommender.dataPreProcessing()
    assert_frame_equal(input, expected)

def test_PreProcessingMethod_NEQUAL(self):
    expected = pd.DataFrame(
                {
                'Title':
                    ['GoodFellas'],
                'Genre':
                    [['drama', 'crime']],
                'Director':
                    ["martinscorsese"],
                'Actors':
                    [['rayliotta', 'robertdeniro', 'joepesci', 'lorrainebracco']],
                'Plot':
                    ["The true story of Henry Hill, a half-Irish, half-Sicilian Brooklyn kid who is adopted
                'Key_words':
                    ["0"]
                })
    input = recommender.dataPreProcessing()
    try:
        assert_frame_equal(input, expected)
    except AssertionError:
        # frames are not equal
        pass
    else:
        # frames are equal
        raise AssertionError
```

6.3 Usability Testing

About two-thirds of the way into my project, I asked some classmates, friends and family members to try out my application and give some feedback to see how usable is my application to everyday users. Of course, I had to get their approval for participating by reading and signing a plain language statement and a consent form.
One feedback I got was that the background of the home page was not appealing and made viewing the email and password textbox difficult.
Another feedback was the color scheme for the tabs. Both of these feedbacks were taken into account and changed.
One negative feedback I got was that the posters covered the entire screens which distorted the image since then I added the column fitting class to fix the issue.
Fortunately for me, all the feedback related to UI issues that did not make the application any less usable.

6.4 Final User Testing

At the end of my project, I ran users test to determine the overall success of my project.
Here is the set of questions I asked the users:

Q1: How did you hear about the application ?

Q2: What movie app do you currently use(if any) ?

Q3: How would you rate the speed of the application ?

Q4: What did you like most about the app ?

Q5: What did you like least about the app ?

Q6: What do you think is an improvement on my competitors ?

Q7: How would you rate you recommended list?

Q8: Would you consider using the app instead of my competitors?

Q9: What is your opinion on the UI and the overall UX

Q10: What is anything you like to add/ change or remove (Feedback)

Generally, people said that the application was very easy to use, very fast in performance and good GUI with overall enjoyable user experience. The recommendations were also a huge success with almost every user saying they enjoyed the overall return result.
Many users who tested the project earlier in its lifecycle said their issues were resolved. A lot of people also mentioned how they would have liked to click through the recommended list to further explore movies but as I mentioned earlier I ran into many errors and could not find the time to fix that.

6.5 Acceptance Testing

Since there is no development team involved with the project, all the acceptance was down to myself and the user. I generally did not move onto the next phase until I accept what was completed. Any functionality that was added was seen was it acceptable by me before following up with the same users as above.
The recommender system, for example, went through an acceptance test. Once I completed it I ran a few movies off a basic dataset to see if the returned set of recommendations was relevant. Here is an example of what my project supervisor entered.

```
print (recommendations(["Shutter Island", "Fargo"]))
```

And here is a list of the recommendations.

```
owner@anas-hp:~/Recommender$ python3 recommender.py
['No Country for Old Men', 'The Departed', 'Rope', 'The Big Lebowski', 'Reservoir Dogs', 'T
he Godfather', 'The Godfather: Part II', 'On the Waterfront', 'Goodfellas', 'Arsenic and Ol
d Lace', 'Vertigo', 'Blood Diamond', 'The Sixth Sense', 'Rear Window', 'Memento', 'Chinatow
n', 'Gone Girl', 'Sleuth', 'The Revenant']
```

7. Results

7.1 Results Overview

Overall I am happy with the result of the implementation. The application, for the most part, followed what I said I was going to do in the functional spec. It allows users to register/log in, view a range of movies, find out further details on the movies, give their own rating on the movies and find a list of recommended movies.

The recommendations were also very successful to the users, my project supervisor and myself.

There were some aspects that I would have changed of course but overall I am pretty pleased.

7.2 Future Work

In my opinion, as simple as the application is, a lot could be added without losing simplicity. Further movie information could be displayed like critic reviews, all crew, and cast, trailers, more images regarding the movies. If there was more time available I would implement a sign out button as well as having the ability to view movies that you have rated already.

This project gave me the confidence to work in mobile development in the future. I really enjoyed implementing Async threads that work in the background while users are using the application. I also enjoyed the ability to manipulate textviews, buttons, images within the UI

For the future, in any application, I would like to get a pipeline on git working so that testing is automatically triggered by commits.

7.3 Conclusion

I am very glad we were made to do a final year project. I learned a lot both on the technical and nontechnical terms. I gained a vast amount of knowledge on android, learned about Firebase, testing, recommender systems. I also cemented the knowledge of documentation I got last year with further documentation this year. I learned about the software development process as well and what should come first second etc as well as what would need to be done if you want to change anything. Looking forward to applying what I learned here into the industry.