## Task 1 Sign in (with-up option)

1. 1. The sign-in screen will require a username and a password (Figure 1).

2. If user cannot connect to the database, user will be informed that you cannot connect to the database (Figure 2).

3. The email and password must be filled out, otherwise it will be prompted (Figure 3 and Figure 4).

4. If the user input the wrong information, the user will be prompted that you input the incorrect email and password. (Figure 5)

5. The user can click the eye which is in the right of password edit box, then he can see the password that he inputed.

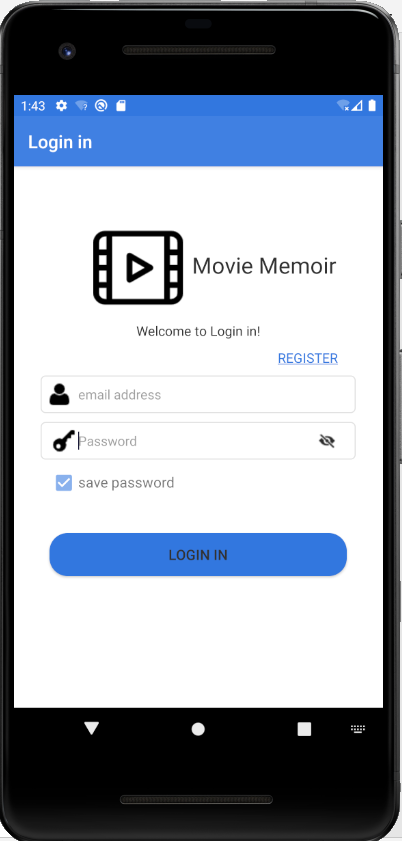
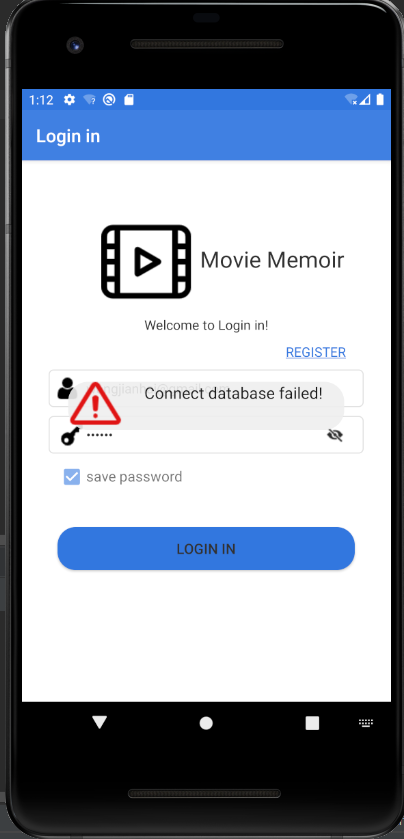
 

Figure 1 Figure 2

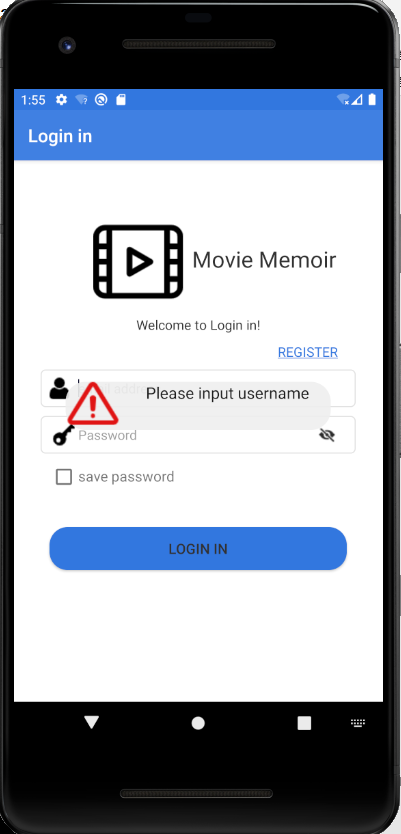
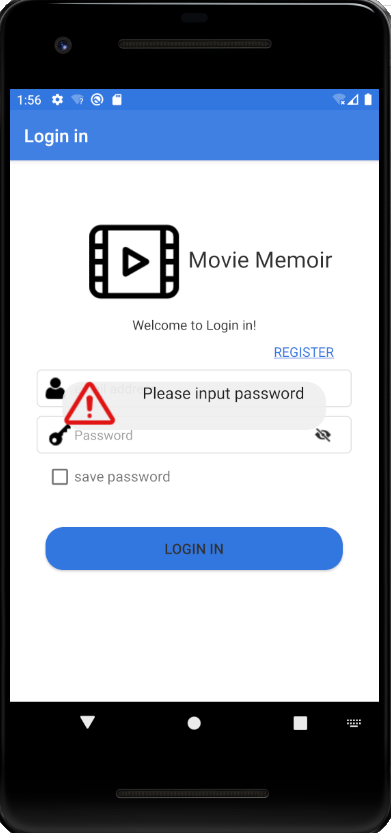
 

Figure 3 Figure 4

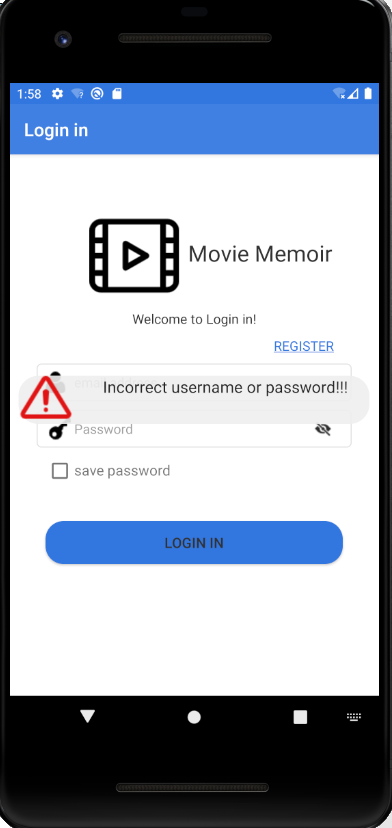
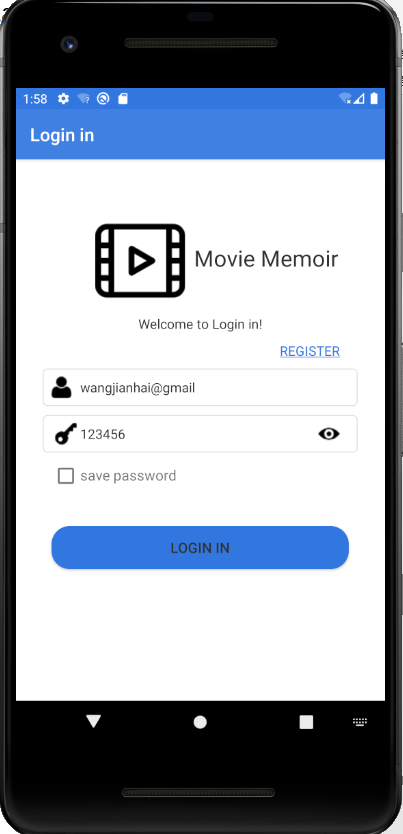
 

Figure 5 Figure 6

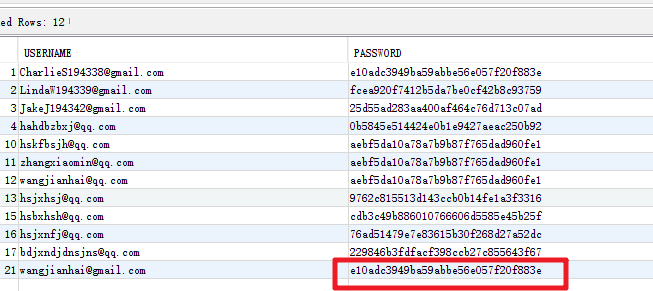
1. Hash the password on the client-side using the right libraries in Java. The codes are as follows.

|  |
| --- |
| public class Encode {  public static String encodeMD5(String inStr){  MessageDigest md5 = null;  try{  md5 = MessageDigest.getInstance("MD5");  }catch (Exception e){  e.printStackTrace();  return "";  }  char[] charArray = inStr.toCharArray();  byte[] byteArray = new byte[charArray.length];  for (int i = 0; i < charArray.length; i++)  byteArray[i] = (byte) charArray[i];  byte[] md5Bytes = md5.digest(byteArray);  StringBuffer hexValue = new StringBuffer();  for (int i = 0; i < md5Bytes.length; i++){  int val = ((int) md5Bytes[i]) & 0xff;  if (val < 16)  hexValue.append("0");  hexValue.append(Integer.toHexString(val));  }  return hexValue.toString();  }  } |

The code to send to the server is shown below:

|  |
| --- |
| PostLogin postLogin = new PostLogin(login\_email.getText().toString(), encodeMD5(login\_password.getText().toString()));  postLogin.execute(); |

The server-side data is shown below:



## Task 2 Sign up form (User Profile) Screen

1. The register form are shown in the following figure (Figure 7, Figure 8, Figure 9)

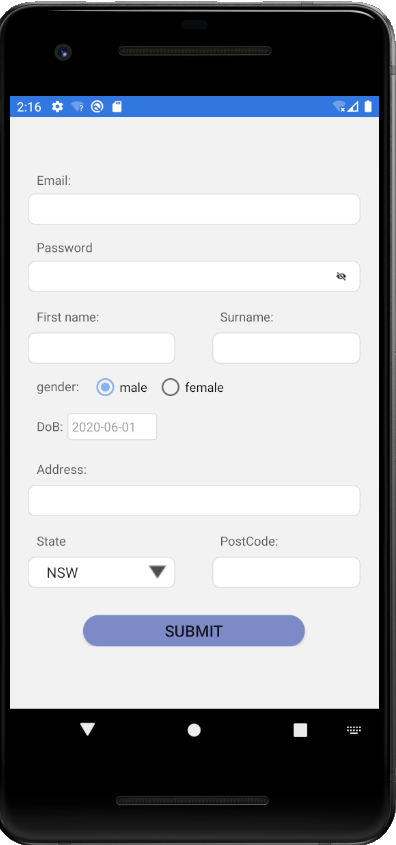
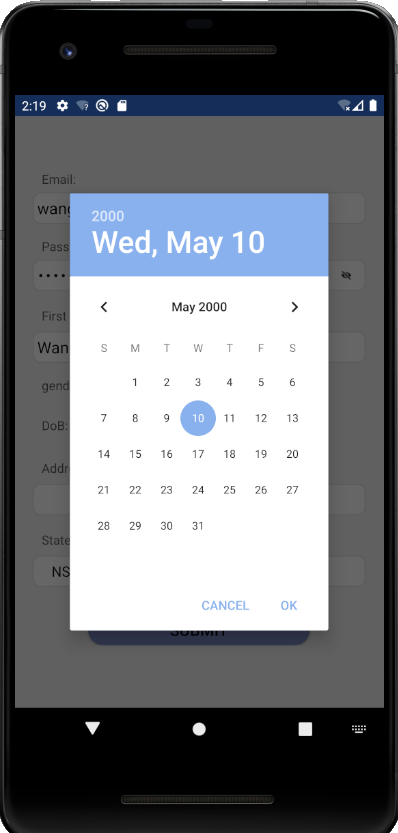
 

Figure 7 Figure 8

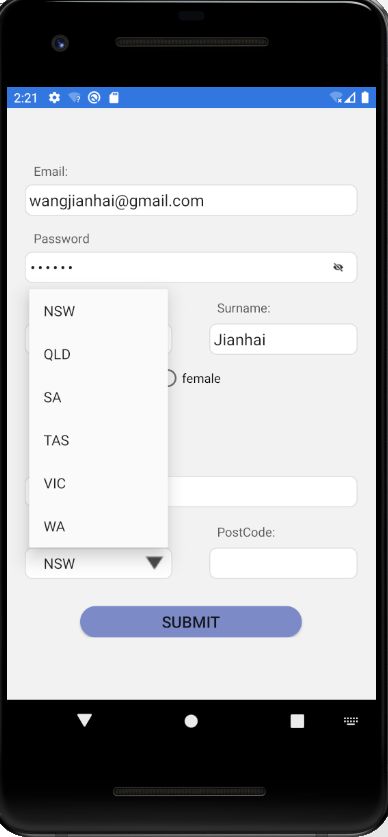


Figure 9

1. Data entry validation and error messages will be informed. (Figure 10)

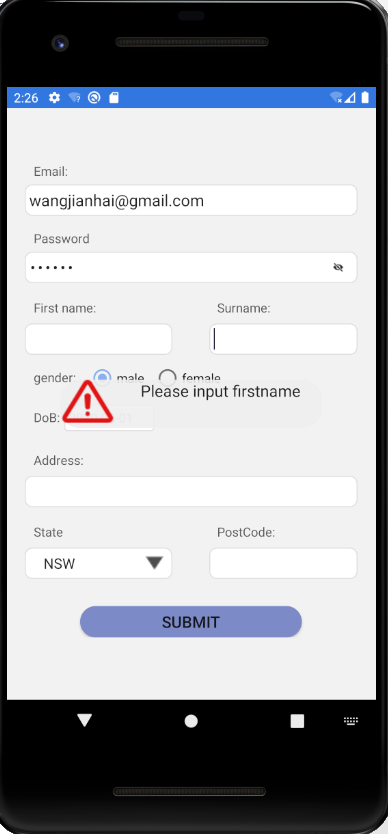
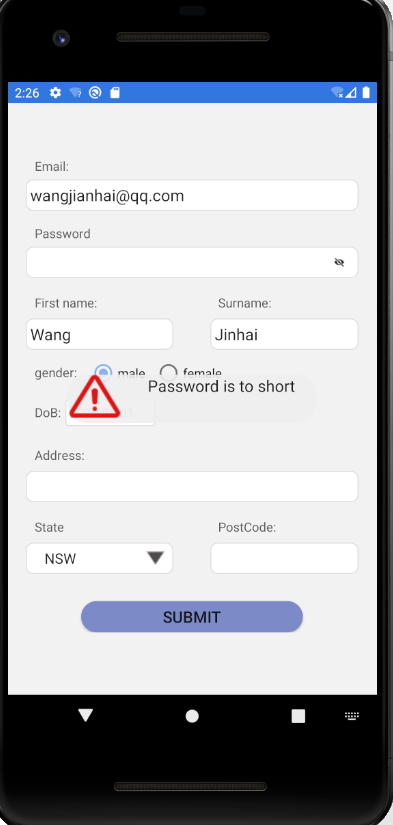
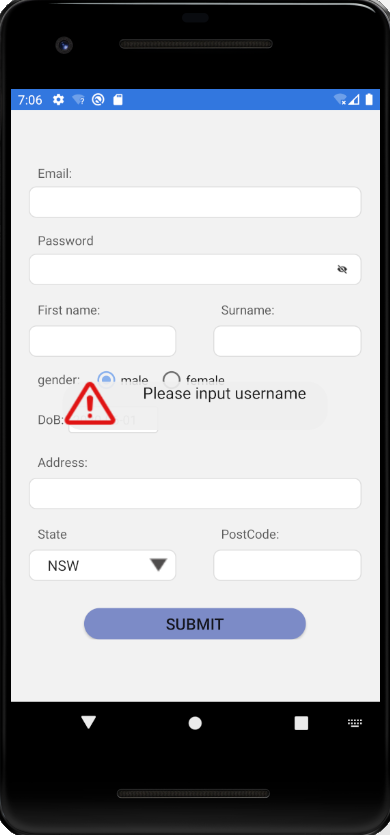


Figure 10

1. Check whether the email exists o rnot. (Figure 11)

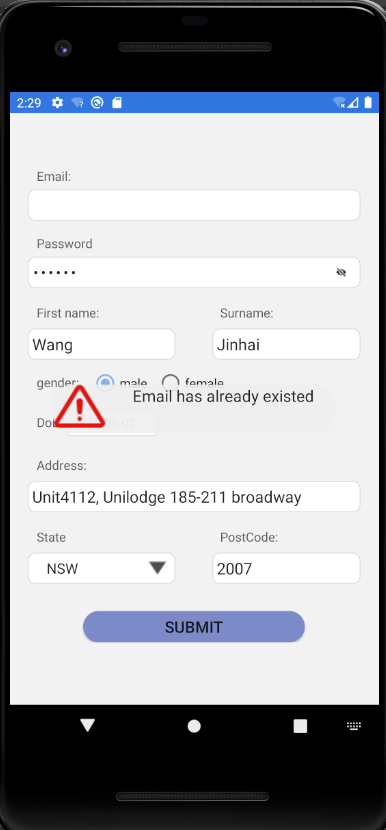


Figure 11

1. The form  follows at least four form guidelines .(Figure 12)

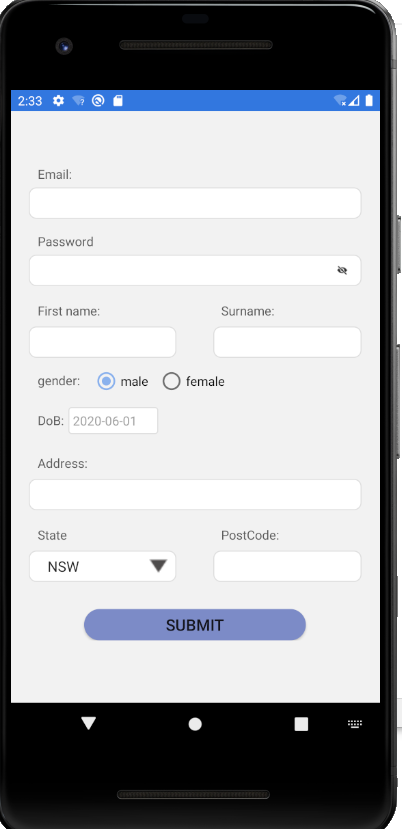
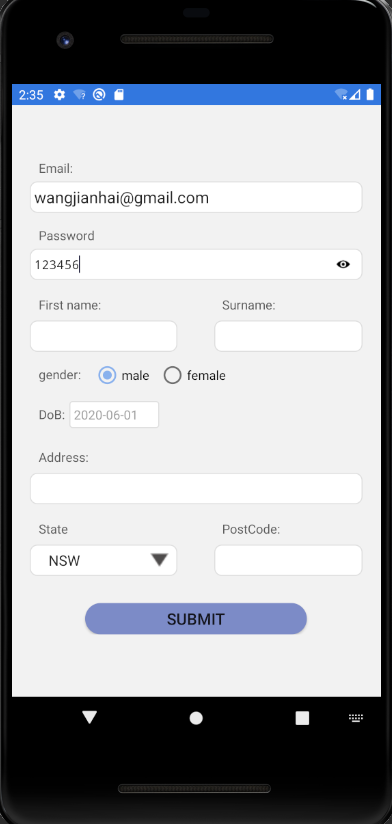
 

Figure 12

1. The code is as follows.

|  |
| --- |
| public String doRegister(Credentials credential) {  final String methodPath = "moviememoir.credentials";  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");  RequestBody requestBody = new FormBody.Builder()  .add("username", credential.getUsername())  .add("password", credential.getPassword())  .add("signupdate", credential.getSingupdate())  .add("name", credential.getUserid().getUsername())  .add("surname", credential.getUserid().getSurname())  .add("gender", credential.getUserid().getGender())  .add("dob", credential.getUserid().getDob())  .add("address", credential.getUserid().getAddress())  .add("state", credential.getUserid().getState())  .add("postcode", credential.getUserid().getPostcode())  .build();  Request request = new Request.Builder().url(BASE\_URL + methodPath).post(requestBody).build();  System.out.println(credential);  String strResponse = null;  try {  Response response = client.newCall(request).execute();  strResponse = response.body().string();  } catch (Exception e) {  e.printStackTrace();  }  System.out.println(strResponse);  return strResponse;  } |

## Task 3 Home Screen

1. The home screen welcomes the user by their first name and display the app title, a related and meaningful picture, the current date, and the top five movie names, his release dates and rating scores given by the user for the current year (2020).

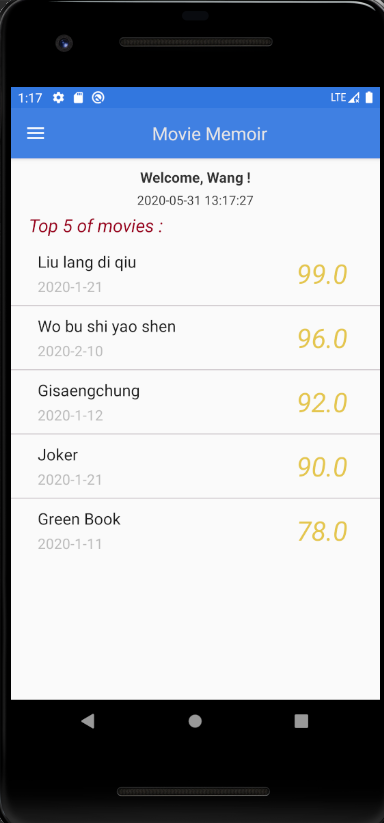
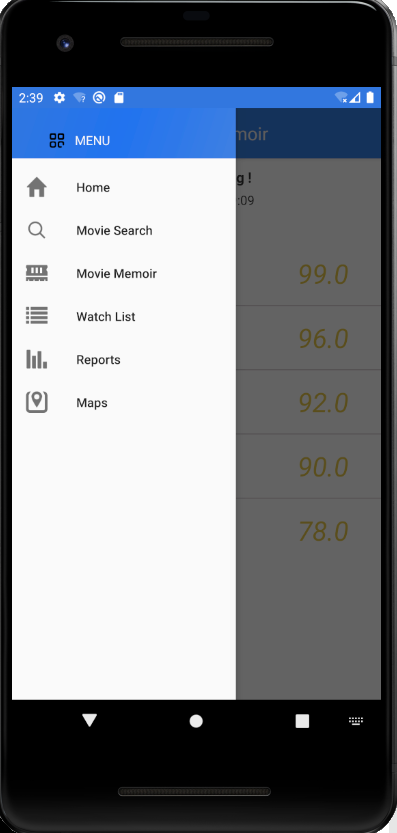
 

Figure 13

## Task 4 Movie Search

The user to enter a movie name (case insensitive), and will display a list of matching movies with some basic information including the name, the release year, and an image.

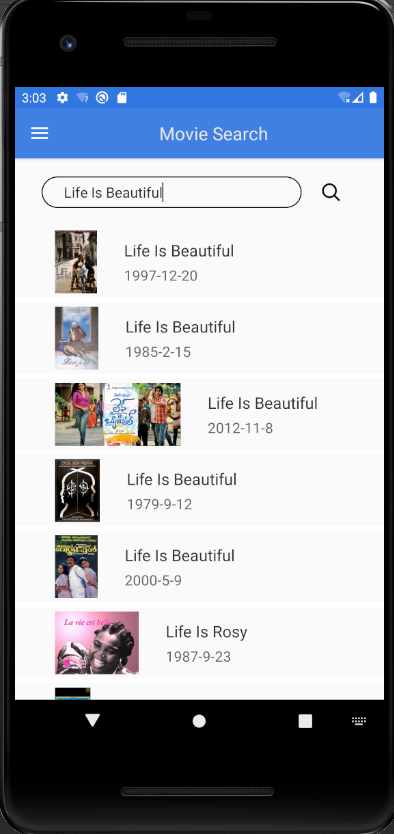
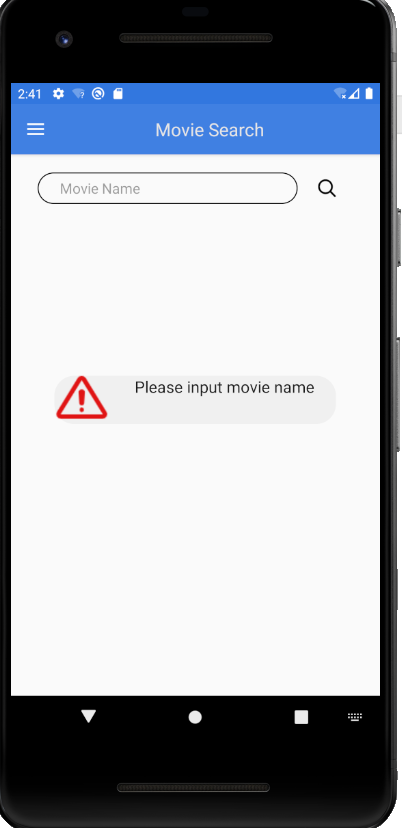


Figure 14

## Task 5 Movie View

The movie view screen will display the following information for the movie selected in the Movie Search screen: the genre, the cast, the release date, the country, the name of the director(s), a synopsis/plot summary/storyline , rating scores given by the public (from a movie API), and displaying scores as stars instead of numbers (supporting both half and full stars).

This screen also provides two options/buttons for adding the movie to Watchlist or the Memoir. When the user clicks the button to add the movie to Watchlist, first you need to check if the movie already exists in the local database. If it exists, it should provide a message about it. If it does not exist, the movie will be added to the Watchlist table and display a relevant message. When the user clicks on Add to Memoir (it means the user has watched it), the user will be directed to the Add to Memoir screen to enter more information.

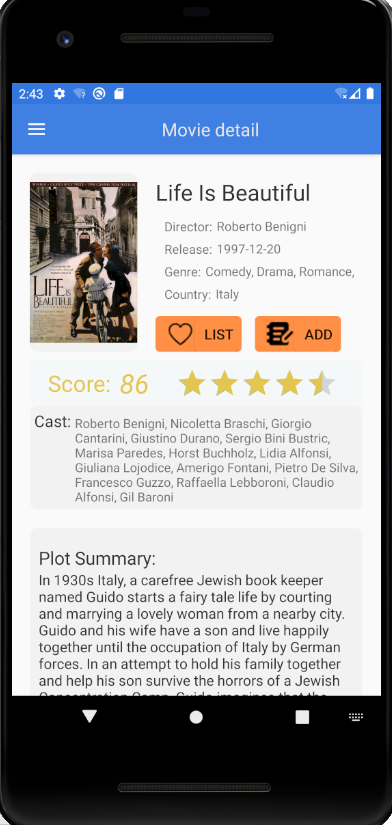


Figure 15

## Task 6 Database, Notification and Watch Screen

1. I use LiveData to hold the Watchlist data and implement the ViewModel and Repository classes.
2. The Watchlist screen shows all the movies stored in the local database (the movie name, release date, and the date and time added to Watchlist) in a list such that they can be selected. The data in the list will be updated after any change through implementation of the observer pattern.The screen should providet wo options for the selected movie from the list: 1) Delete and 2) View.

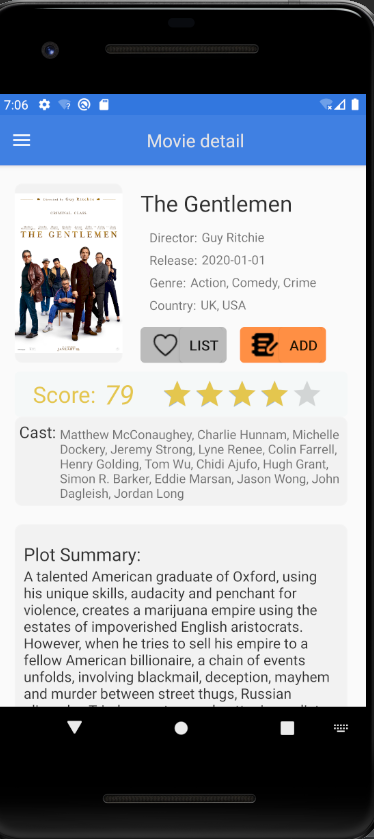
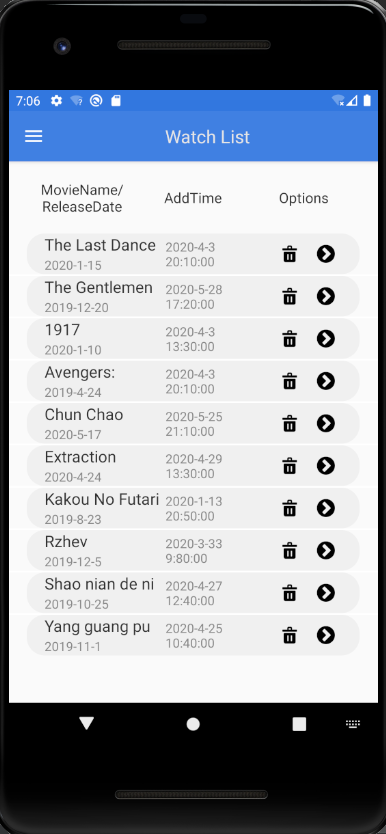


Figure 16

1. In the Watchlist screen, after selecting a movie from the list, if the View button is clicked, the user will be redirected to the Movie View screen where it will show the details of the selected movie. In the Movie View screen, the Add to Watchlist button must be disabled because the movie already exists in the local database. The purpose here is to allow the user to add the Watchlist movie to their memoir if they watched it.
2. In the Watchlist screen, after selecting a movie from the list, if the Delete button is clicked, the movie can be deleted. Provide a confirmation message before delete. After the user confirmation, the selected movie can be deleted from the local database, and the watchlist screen will be refreshed.

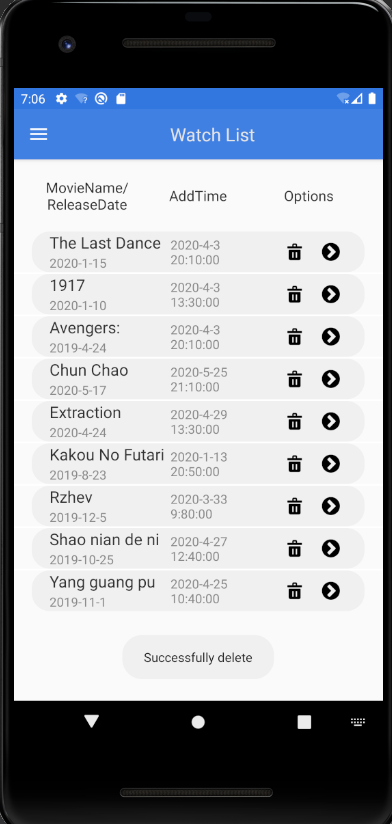
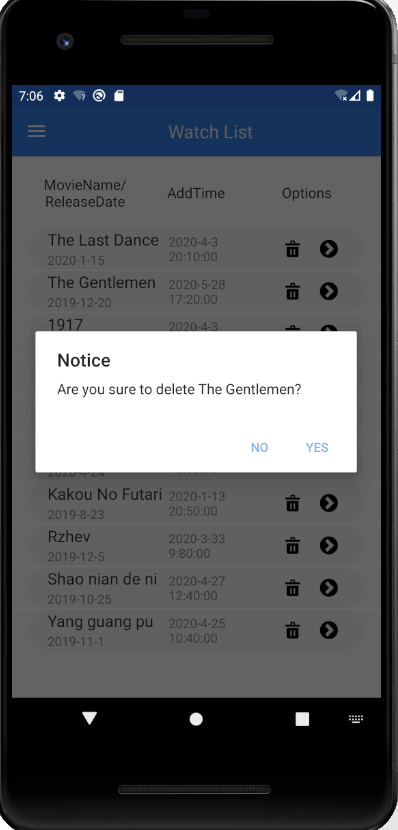


Figure 17

## Task 7 Add to Memoir

1. This screen shows some data passed from the previous movie view screen including the movie name and its release date, plus a related image
2. The screen allows the user to enter more information about their movie memoir. This includes: 1. the date that the user watched it (use a date picker) and time. 2. The cinema name and its suburb or postcode I use an additional screen to obtain the cinema details from the user. After the cinema is added (requires using a POST method), the spinner list needs to be updated. 3. A short comment about their memory, opinion and how they felt after watching the movie A rating score for the movie provided by the user using stars. I use a creative, dynamic and efficient way to support full and half stars for rating.

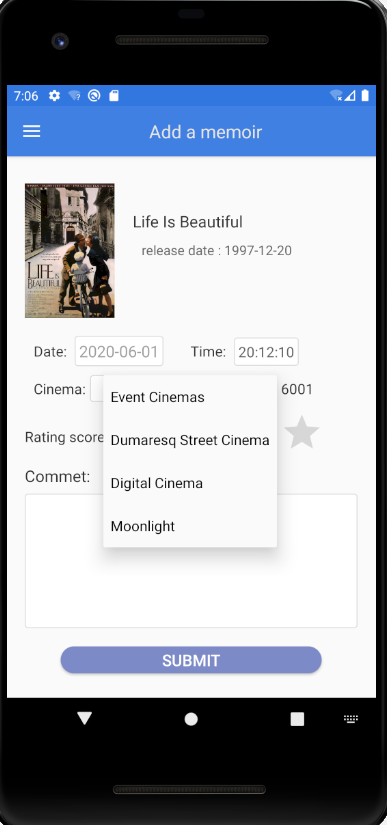
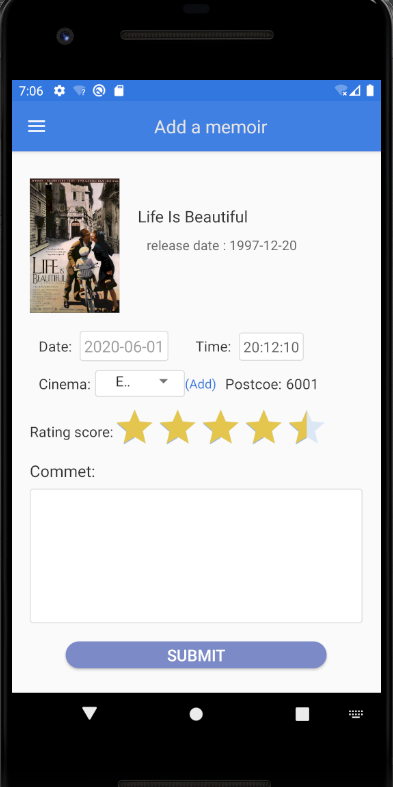


Figure 19

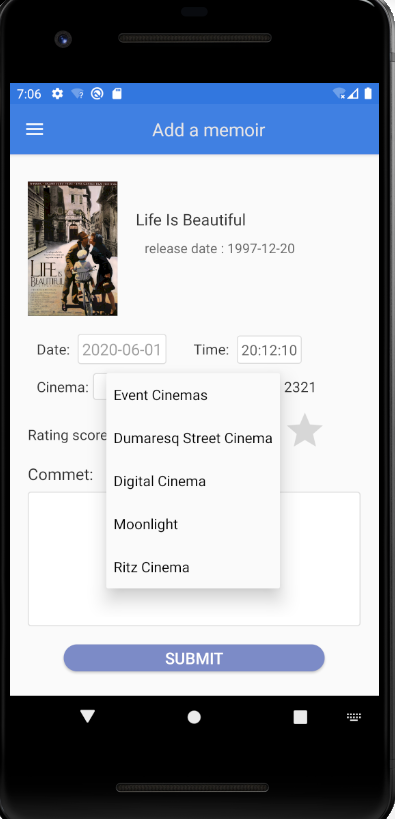
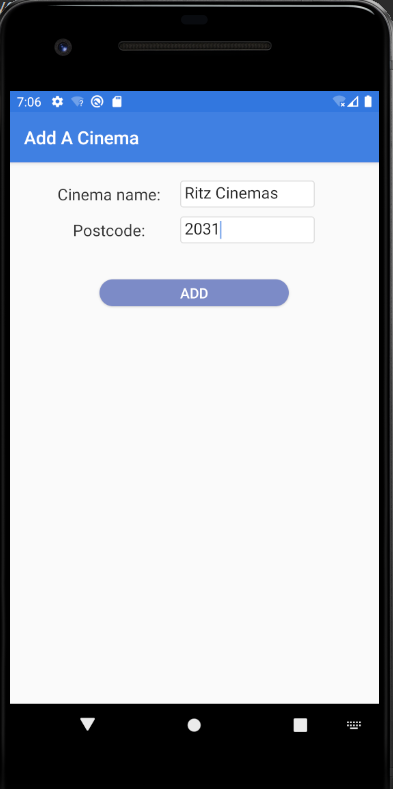


Figure 20

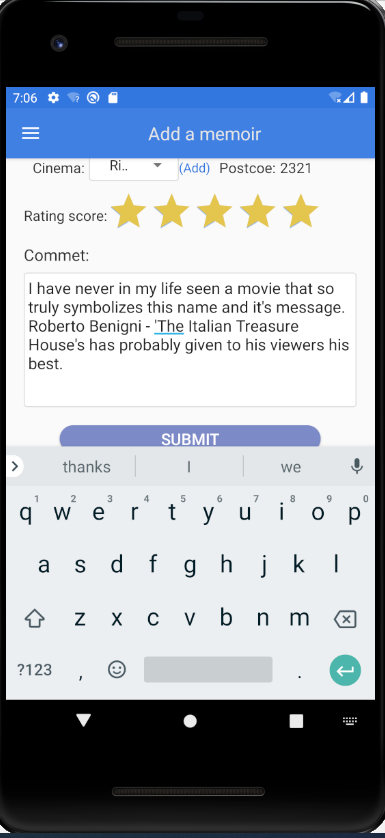


Figure 21

1. After all the information is entered, and the user clicks the submit/add button, the movie memory details and cinema id details must be added to the momoir and cinema table at the server side.

## Task 8 Movie Memoir

a) This screen displays a list of all movies stored on the server-side database. This list will show the movie name, the release date, an image , and the date that the user watched it, cinema suburb/postcode, user comment/memory, and the user’s rating score as stars.

b) The screen will provide three sorting options (i.e. based on the date, the user given rating scores as stars,and public review scores (from a movie API) as stars. The filter options can include different genres (use a spinner)

c) This screen is scrollable. Each movie can be selected. When a movie is selected, further details about that movie should be retrieved from the APIs including the official trailer. The details will be similar to the results of Movie View .

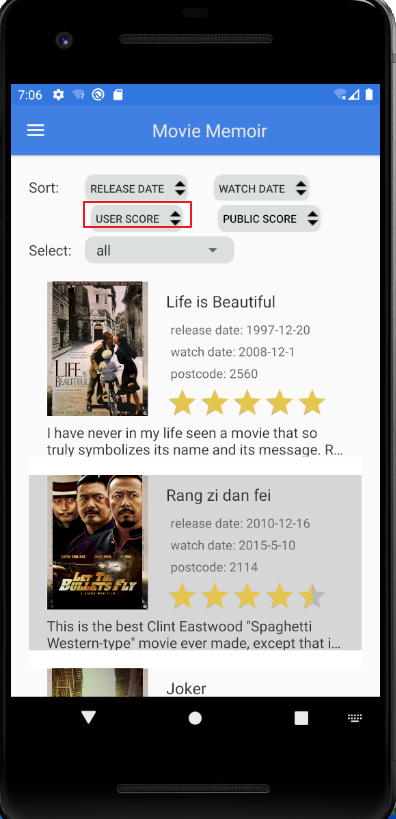
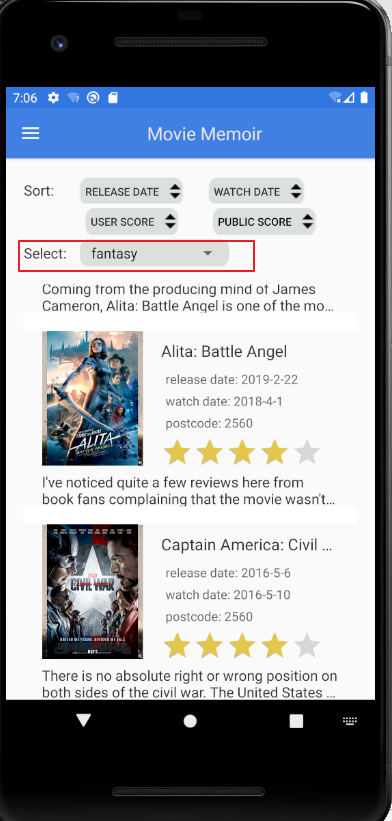


Figure 22

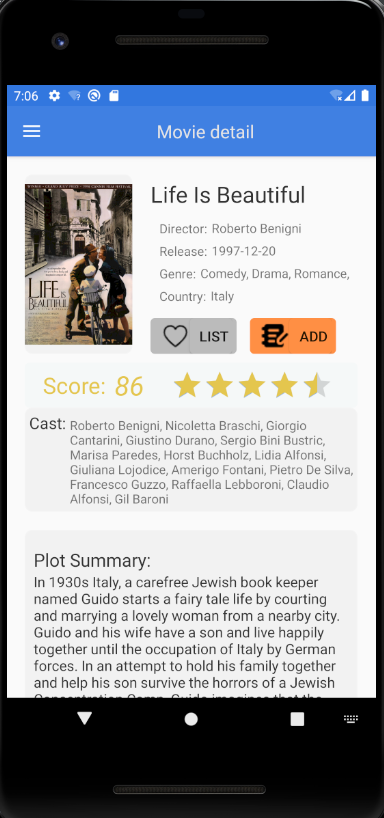
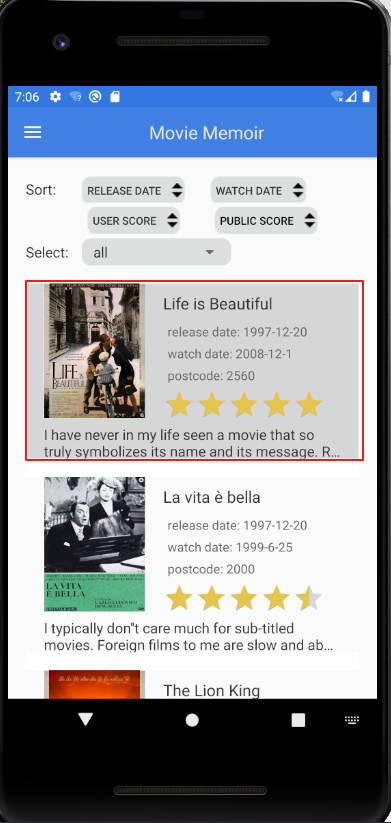


Figure 23

## Task 9 Report

a) Pie chart: the top section of the screen will include a date picker to allow the user to enter a starting date and an ending date, and a pie chart that will display the total number of movies watched per suburb/postcode (as percentage %) for the selected period. The labels and percentages should be shown on the pie chart.

b) Bar graph: the bottom section of the screen will include a spinner to allow the user to select a year (2015 to 2020), and a bar graph to show the total number of movies watched per month for the selected year.

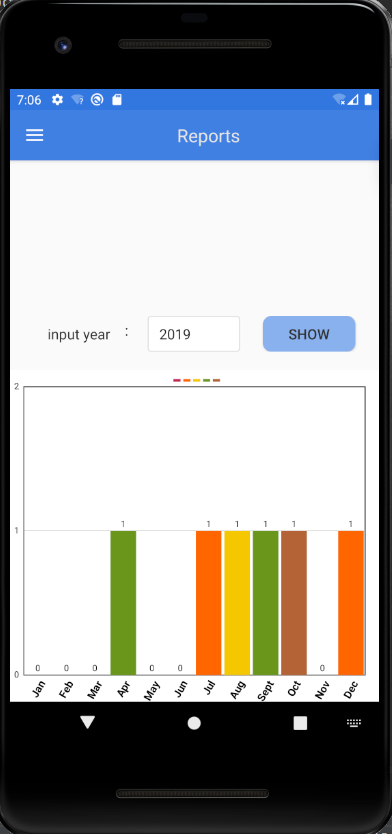
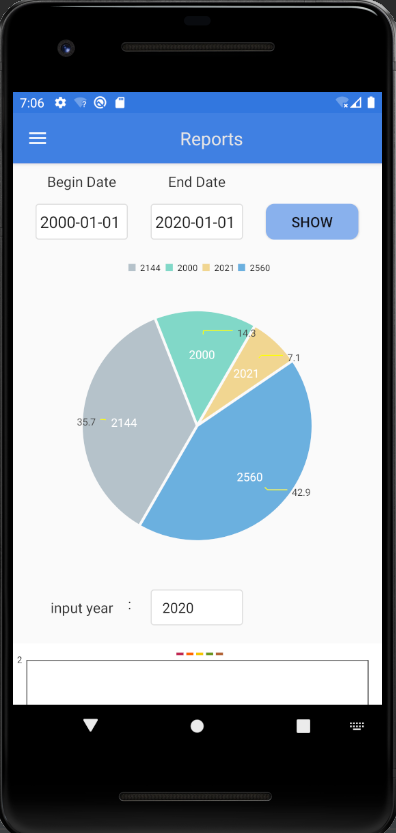


Figure 24

## Task 10 Map

1. The map screen will show the user’s home location. You need to programmatically convert the full address of the user based on their address and postcode into latitude and longitude values. Then use the latitude and longitude values for displaying the location on the map.
2. Convert the cinema suburbs/postcodes into latitude and longitude values and show

them on the map. This requires a GET method to retrieve the user information from the server-side database.

c) When the user taps on a cinema marker on the map, display its name.

d) The marker for the user location and cinemas should have a different color.

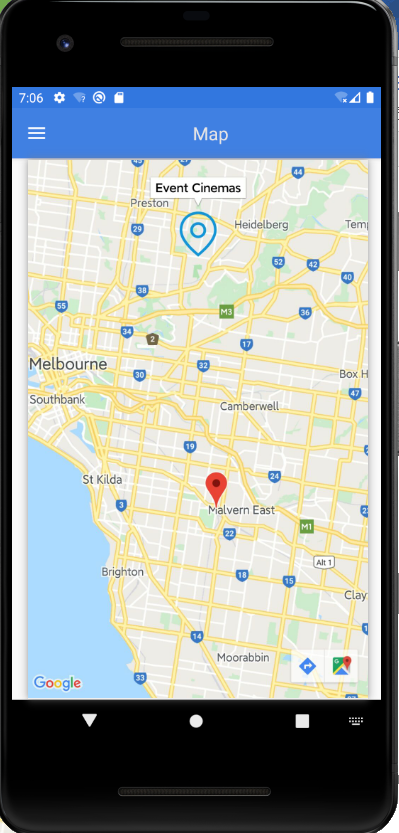
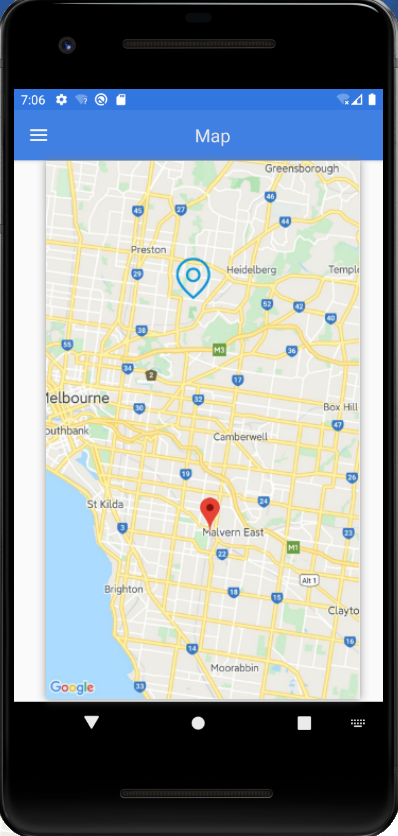


Figure 25