DEPARTMENT

CCP3730: Mobile Computing

Spring 2024

Mobile Application

Submission Deadline: 18/05/2024

Actual Submission Date: 18/05/2024

Student Registration Number
CSY21048
CSY21054

Declaration: *I have completed and submitted this work by myself without assistance from or communication with another person either external or fellow student or any AI type of content generator. I understand that not working on my own will be considered grounds for unfair means and will result in a fail mark for this work and might invoke disciplinary actions. This piece of assessment will be continuously checked for its academic integrity until my graduation and the mark will be revised if it is found to breach the unfair means policy. It is at the instructor's discretion to conduct an oral examination which will result in the award of the final grade for that particular piece of work.*

Technical Document: Movies App

Ylber Galica CSY21048

This is a technical documentation for the project coursework on the class Mobile Computing 2024, we were tasked to create a mobile application using android studio, this application would let users browse a selection of movies from an api provided to us furthermore it would allow the user to favorite movies and browse offline these movies. This document will explain how the frontend and general folder architecture were handled.

1. Folder Structure

/java/com/york/moviesapp

This is the main source set for the Java code. It follows the standard package naming convention, which helps in organizing and uniquely identifying code. This is also where the MainActivity lies, the entry point of the application. The subdirectories of this folder are as follows:

- **/database:** Contains all the files related to the local database and cache management.
- **/helpers:** We gathered static classes with reusable and modular methods to one folder. Currently the folder only contains one class related to API request handling.
- **/recyclerview:** Contains adapters and view holders for all the RecyclerViews used in the project.
- **/ui:** Home to the MainActivity class. Contains the user interface components of the app, these are bundled into their own subfolders. The project makes use of four different main fragments with their own class and model file, these fragments include the Home, Dashboard, Details and Favorites, which will be explained more in detail further below.

/res

This directory contains all the non-code resources used in the application. These include files for the layouts and menus, drawables such as images and icons, values for colors and themes, etc. These are all bundled into their own respective subfolder for better folder structure and clearer development.

2. Frontend

The application contains 3 main screens:

- Home
- Dashboard
- Favorite Movies

Home Screen: This screen greets the user to the app. For the sake of making the app more user friendly and not clutter the home screen with too much information, it only shows the title of the application and an image that gives a clear idea as to what the app is about.

Dashboard: The dashboard displays the large list of movies provided by the API. The movies are all grouped into genres with each genre being a horizontal scrollable view, the items display the poster of the movie, paired with the title and the release year. The user may then click on one of the movies to navigate to the details page of the movie.

Details Page: The details page displays the movie details. It contains information such as the title, release date, rating, genres, a short synopsis of the movie, length, as well as cast and crew. The user also has the option to favorite the movie from this screen via a star button at the top left. The user may choose to unfavorite a movie at any time via the same button.

Favorite Movies: This screen displays all the favorited movies by the user, in a grid displaying their poster and title.

The user is able to navigate through the main three screens via the navigation menu at the bottom of the screen. The navigation bar is created using the component "BottomNavigationView" as well as other Navigation classes provided by the android studio as boilerplate code.

2.1 Fragments

The screens in our application are not separate activities, instead we use fragments. Our application is fragment based instead of activity based. It essentially resembles a single page web-application, where the activity is never changed, we simply swap between rendering different fragments. In order to add a new fragment to the application, we add it to the navigation XML file responsible for handling the navigation of the main screens.

A problem we encountered during the development of the application was the issue of passing data from one page to another. In an activity based application, intents are used to switch to another activity and pass data as parameters. However, fragments involve a different method compared to intents.

Inside the navigation XML file containing the fragments, we add to said fragments, actions and arguments. Actions define navigation that may occur while within the fragment to another fragment, it involves giving it a destination fragment and later using a NavController to navigate to another fragment. We assign said NavController method to an OnClickListener of a movie item inside the dashboard fragment to display the Details page.

Fragments are also given arguments in the aforementioned XML file, when we need to pass parameters from one fragment to another. Arguments can be of any type and defined as the developer desires. When actually switching from the Dashboard fragment to the Details

fragment, we pass a Bundle containing the ID of the movie. This is given in the OnClickListener method of the movie item.

2.2 RecyclerViews

An important part of the project was the RecyclerViews, which are used for scrollable fields with dynamically rendered items. They provide a more powerful and memory efficient version of the ScrollView, as they dynamically change the content of the items that move past the visible screen and are recycled to be used as the next item coming into the screen.

To make use of RecyclerViews we implemented Adapters for each one we needed as well as ViewHolders, and of course one additional xml file describing the item that will be dynamically rendered many times by the RecycleView.

A RecyclerView Adapter is what controls what and how the items get rendered in the RecyclerView. In each of our cases, given the data of the items that we want to render, the adapter will parse them and render them onto the layout frames.

2.3 Image Loading with Glide

We faced an issue where simply setting the “src” property of an ImageView through code would not result in the image being rendered. Thus we resorted to using a library called Glide, which provides image loading and caching functionalities that are very efficient in both performance and memory. Using glide also enabled the images to still be loaded even when the user’s device is not connected to the internet, as the images will automatically be cached by the Glide library.

Using Glide to render an image onto an image view is very simple, here is a sample code snippet showcasing the main functionality:

```
Glide.with(context).load(“sourceURI”).into(imageView);
```

2.4 Themes

During development, we faced an issue where colors would not update according to the theme of the android device of the user. The background would update to black or white whenever the device’s theme was dark or light respectively. However, the text and other colors would stay the same, making it unreadable at times.

In order to fix the issue, we created a new subfolder inside “/res/values” called themes, which contains two XML style files containing different value usages for colors and textColors. However, one XML file is used for when the light theme of the device is active, while the other is for the dark theme.