Introduction

MoodLogger is a diary application built specifically for the android platform. The purpose of MoodLogger is to enable users to record how they are feeling at any time throughout the day by creating a 'Mood Log'. Created Mood Logs are saved to Shared Preference and displayed on the 'Recorded Logs' page. The home page of mood logger also displays inspirational quotes about happiness. This is to help lift the mood and inspire the user to be more positive. These quotes are pulled directly from an API through an HTTP Request, and the quote is changed every 8 seconds.

Another unique aspect of Mood Logger is the 'Positive News' page. Given the consistent doom and gloom seen on the news, I thought it was important to try and share the many positive news stories that are currently happening in the world. The Positive News page uses an implicit intent with an ACTION VIEW to open a browser of the users' choice and read the news content.

Design Rationale

My design choices were centered around ensuring the core functionalities are implemented correctly. In this section, I will justify my choices on fragment use, layouts chosen and persistent data storage choice. I did not need to implement a Content Provider.

The layouts used throughout the application consist of Constraint Layout and Linear Layout. On the Home page, I also implemented a Scroll View to parent the sub-groups and give vertical scrolling functionality on smaller screen sizes. I used Constraint Layout throughout the application since I have positioned many views in a way that is relative to other views. Throughout the application, Constraint Layout was used as the parent of the View Group. This was to make it easier for responsive design on smaller devices. I used Linear Layouts whenever I needed to stack components vertically or horizontally with equal spacing.

Regarding persistent data storage, I chose to use Shared Preferences. This was due to the simplicity of its implementation given the time constraints of the coursework. Each record held a Log ID as the Key, and the concatenation of Log Title, Body and Date as the value. Each subsection of the concatenated string was encompassed in unique characters (e.g. {Log title}[Log body]@Log date@). This meant that when I retrieved the Log ID and its value from the Shared Preference, I could segment and extract the content of its value based on the unique character that encompassed it using the substring method. For example, to extract the title, I would create a substring based on the indexes of the first and last '{ , }' characters. Using this method, I was able to pull all stored logs from the Shared Preference and populate the Array List which was then passed into the Recycling View Adapter. This is how the Recycler View on the Recorded Logs page is populated.

For fragment use, I used one fragment throughout the application located in the View Log activity. View Log opens when a Log is tapped in Recycler View. This design choice was made since each View Log activity had the same design. Therefore, it made sense to use a fragment which simply loaded the selected logs unique contents into the fragment dynamically.

To eliminate any potential for ambiguity, below is an explanation of how each of the assessment criteria requirements has been implemented into my applications design.

Multiple screens that users can navigate through. As can be seen, this is implemented throughout my application, which has a total of 4 unique activities, one of which contains a dynamic fragment. If we include the implicit intent to open the news page on the browser, this is a total of 5 activities. Regardless, all these activities smoothly transition to one another using explicit and implicit intents and any associated data these intents need.

Use both explicit and implicit intents. Following on to the above point, I have used explicit and implicit intents throughout the application. For the most part, explicit intents are used for activity transitions. However, on the Positive News activity, a button is implemented to trigger an implicit intent on click.

Has menus. My application has an Options menu implemented in the top right of the navigation bar. Within this menu consists of the following menu items: Create Log; Recorded Logs; Positive News; Clear Recorded Logs. This menu acts both as a global navigation for the application, as well as a tool for data storage monitoring, as it enables the deletion of all records from the shared preference. It should be noted that in order to delete all Log data stored in Shared Preference, the Clear recorded logs menu item must be clicked, however, the changes will only take place once the app has been destroyed and restarted.

User recycler view. I have implemented a Recycler View on the Recorded Logs page. A dynamic array list containing the persistant logs is fed into the recycler view on every activity load, which then populates the Recycler View.

Use data storage. I have used Shared Preference to store the logs persistently. I have also implemented functionality to clear all logs from the Shared Preference via the options menu.

Use the internet. In the main activity I have implemented an inspirational quote generator that sits as an overlay above the image view containing a picture of mountains. The quote data is pulled via an HTTP connection and a GET request. Since the return value was a JSON object, I needed to parse it to a string before setting it to the quote text view. I re-pull from the API to generate a new quote every 8 seconds. All this functionality is offloaded onto a separate, anonymous thread.

The app opens and displays screens correctly without internet. The app will give the core functionality of my app without the internet. The only functionality that will be limited is the quote generator. In this case, I have set a default quote to display instead.

Novel Features

The novel feature on my application was the use of the inspirational quote that was being pulled from the API. This is a dynamic feature, that is offloaded from the UI thread, and updates the Text View every 8 seconds with a new quote. I think this was a unique and creative way to implement the internet usage aspect of the application.

Challenges Faced

The main challenge that I faced whilst developing this application was the retrieval of Logs from the shared preference, and the extraction of the value substrings to populate the Recycler View. The main reason I chose to use Shared Preferences was due to the simplicity of its implementation, which given the time constraints of the project was appealing to me. However, since Shared Preference only holds a key value pair for each record, this meant that to store the needed data for each log, I had to create a formatted string field in the Log class. Logs could then hold a concatenation of the Logs data (title, body and date). This would then be the value that I stored in the Shared Preference, and the Logs ID would be the key. This created unnecessary work when pulling the data from the storage, as I had to extract the data for each log using the substring method. In hindsight, and with more time, I would have set up a third-party database such as SQLite, as this would have enabled simpler database transactions.