Coursework Report

Ross Langan
40276526@napier.ac.uk
Edinburgh Napier University - Mobile Applications Development (SET08114)

Abstract

This paper contains the planning, implementation and testing of an android application designed to store collections of items. In each section I describe a different aspect of the project, starting with an overview of my application choice and inspirations. I cover the design and implementation, as well as an evaluation on both the implementation and my performance. It is hoped that this paper will fully cover every aspect of my project and the both the successes and challenges faced along the way.

Keywords – Android, Application, Collection, API

1 Introduction

The application I have designed for this project is a gallery designed to store images of collections of items (e.g. vinyls, coins), and includes an built-in camera function. Users are able to create and name folders, browse through existing collections, add images to any collection and expand and zoom each image.
The original concept for my app was a catalog of album art, representing the vinyls in the users collection. A separate section would contain a 'wish list', and include vinyls not owned. On the click of a button, the user would be able to search quickly for any album by any artist, and download the album art to the appropriate folder (owned/wish list). This feature would require the internet and it would download data from an api. Ideally this would suggest an album as the user is typing, like Google's autofill to maximise the speed of a user selecting an option, as well as minor spelling errors not being an issue. Once each image is saved, it is displayed in a gallery-style screen, with two columns and potentially several rows of square images. This would be a scrollable screen, with expandable and zoomable images. Below the expanded image would appear the name of the artist and vinyl, and the tracklist for the album.
This concept was aimed at people of all ages and locations. The market for vinyl records has been on the rise since the mid 2000s, meaning a huge portion of users will be young adults and teenagers who have started collecting, as well as older generations who have had vinyls from before the rise of CDs.
The idea of keeping all my vinyls in one place on my phone was inspired by the time consuming process of flicking through all my vinyls showing friends and guests what I owned. With all my vinyls stored as images in one place I could show them off easily and quickly meaning my guests could choose a vinyl quickly and with ease. I took some inspiration from Instagram's scrolling grid view of images, with an expandable image that contains a description.
Before I made my decision I researched ideas for my application. I read several articles on vinyl shopping applications including the discogs app [1]. I also found a blog[2] written by a person who wanted to document his collection of 850 vinyls. He tested 5 applications and reviewed each based on features, ease of use and ease of adding vinyls to the storage. I read an article by AppCrawler[3] rating similar apps for iOS.

2 Software Design

The planning of my project proved to be a process which did not end until the application itself was finished. With features changing throughout the implementation, I had to change and edit plans many times during the project, however application does remain very similar to my plans.
I began with designing a user interface for each activity in my app. I identified the four main activities which the user will interact with as the album screen (main page), the add album screen (also main page), the display of images inside each album and the camera activity.
The homescreen design (as seen below in Figure 1)includes a basic gridview with square album previews and a number for each album. It also includes a fab at the bottom of the screen for launching each of the two activities that is required- add album on the left and launch camera on the right. I also included the action associated with the back button, which in this case would quit the application and return to the android home screen.

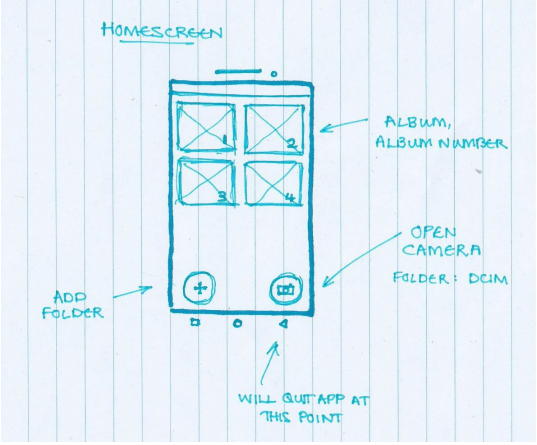

Figure 1: **Homescreen**- Page displayed on launch of application containing albums

To add an album, I designed a screen that is very simple

to use and easy to understand. I originally planned a pop-up window with an editText and two buttons, however after consideration I thought it might be easier to attach this to the current view. I began by adding an editText at the top of the screen, a keyboard which I should be able to make appear and mini fabs below the larger ones. To this plan I also added some basic pseudocode (seen top left of Figure 2) covering the creation of a boolean allowing me to share the current album name when required between classes. Furthermore, I planned the use of 'mkdir' to create a directory within the DCIM, and noted the tasks associated with each new fab.

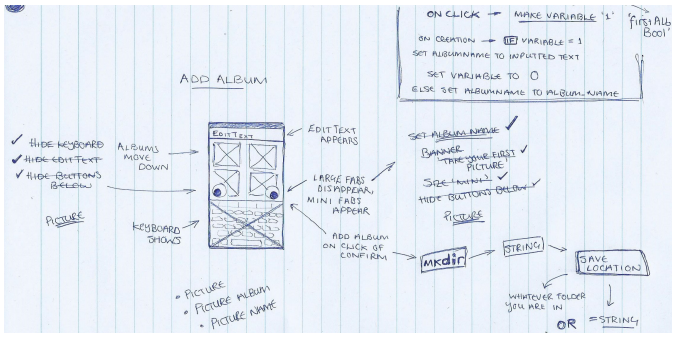

Figure 2: **Add Album**- Page displayed on click of add album fab

The camera activity I drew is inspired by Instagram's camera activity on iOS. I added a large circle button as the capture button (sourced here[4]). This also removes the need to have any kind of text inside the button. As a user of an iPhone, I was inclined to add a back button to the screen rather than relying on the back button of my Samsung, both out of habit and ease of switching between activities.

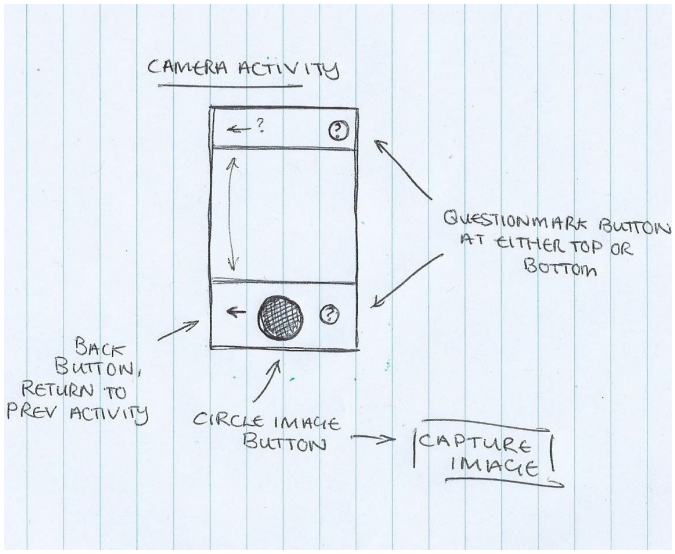

Figure 3: **Camera Activity**- Page displayed on click of camera fab

At the beginning of the process and after I had my basic ideas had developed, I made a GANTT chart to keep track of the time I spend on each task, and to make sure I finish the project in plenty time. Furthemore, I marked tasks which were achieved in green, partially achieved in orange and not achieved in red as the application progressed. As time went on I updated the bars on the chart if one task was going to take me longer than expected. This means I could still tell whether my application was going to be finished on time or not, and make appropriate changes if needed.

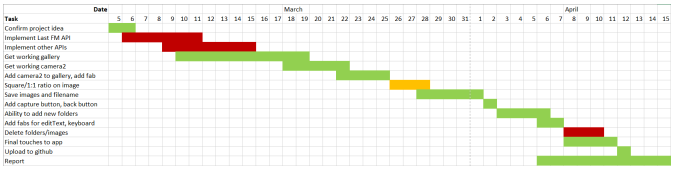

Figure 4: **Gantt**- The gantt chart I used

3 Implementation

The implementation of my current and final application (leaving out all previous failed attempts) started with following the YouTube video and tutorial on AndroStock[5].I found in my planning. This created a basic gallery app for me to begin adding the main features to and changing to suit my needs.This example uses the the Glide image loading library and Zoomable ImageView library (Licenses included in project).
I began by removing the actionbar to make it fullscreen, keeping the notification bar at the top containing the time and battery. I changed each album preview to a square to suit the needs of my original vinyl idea, removed some of the writing that appeared on each preview and changed the background colour of the application to #383838 (Grey22) for a darker theme, all seen in Figure 5.

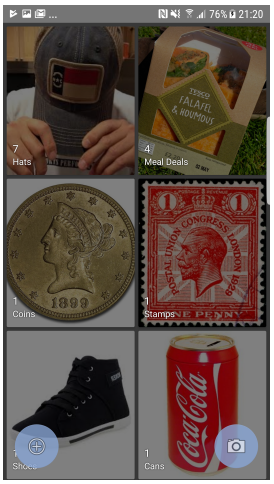

Figure 5: **Homescreen Layout**

After failing to implement any of the useful apis, I searched for documentation and programs relating to the new camera2 api. I found a basic example by Google[6] and added parts of the code to my existing application. To bridge between the new camera activity and the album activity I added a floating action button (fab) seen in the lower right-hand corner of the homescreen.
I changed the appearance of the camera activity to show a square window for the image. To match the desired look for my application I changed the colour scheme of the bar

at the top and the bottom of the screen to match the grey theme. Next, I added an image I found on Google that was licensed for reuse as the camera button[4], matching more the style of circular capture button found on Instagram and Snapchat, and I added an information and back button on either side of the screen as seen below in Figure 6. The information button contains instructions for the camera and the back button returns to the previous activity.

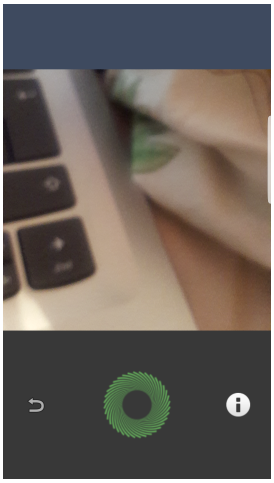

Figure 6: **Camera Activity**

To enable the user to add new folders, I added a fab on the left-hand side of the homescreen. On click, an editText box appears at the top of the screen, the keyboard shows and both the new folder and camera fab disappear. I added two smaller fabs which become visible when this event happens, representing a cancel and confirm button. On cancel, the editText and keyboard disappear, as do the cancel and confirm fabs. On confirm, a new folder is created using mkdirs(). (Figure 7)
Returning to the camera, to save the captured images in the DCIM rather than in AppData, I had to retrieve the path of the external storage seen in the first part of Listing 1 below. To build the name of the image, I first created the variable 'sname' which holds the name of the selected albumactivity, and timestamp which contains the current date and time as a string. Each time an image is available, I concatenate the DCIM path, sname, which represents the folder, '/item', timeStamp and the extension '.jpg'. If the user launches the camera activity from the homepage, sname remains empty and the image is saved in the DCIM directory.

3.1 Code Listing

Listing 1: File Naming in Java

```
1 mFile = new File(Environment.↩
      getExternalStoragePublicDirectory(Environment.↩
      DIRECTORY_DCIM), sname + "/item" + timeStamp + "↩
      .jpg");
```

Finally to finish my app, I changed several appearances of the word 'vinyl' in variables and on the display to more suit the idea of my new app. I changed the name of the application from 'Vinyl Collector' to 'Collection Master'.

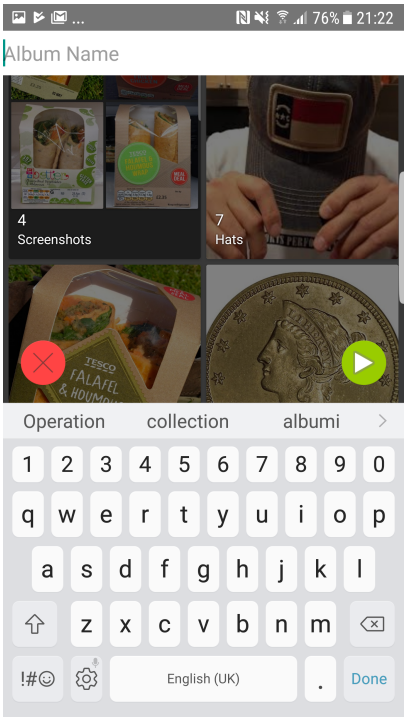

Figure 7: **Adding an Album**

4 Evaluation of Implementation

My finished application 'Collection Master' is in many ways different to the original concept I designed. The main element of my app that does not match the original idea is the use of the camera to capture images and populate each folder. The initial idea was to include a search function, where the user would enter the name of an artist or album and the api would return an album cover. The high-quality image would then be saved in the app data and displayed in the grid view. However issues with licensing, depreciated apis and lack of documentation meant that the closest I could have to that is a camera where the user could take a picture of each vinyl. Furthermore, the camera caused further issues as I was unable to achieve a square photo with a 1:1 ratio like Instagram or the 'square' photo option on iOS. I read several online forums about this issue including an article on stackoverflow[7], however the solution did not seem to fix my problem, as the preview would look square, but the picture after taken would remain a larger size vertically than horizontally. Testing my camera on a vinyl, I discovered that the large image would not display correctly in my gridview with a vinyl, so this prompted me to change my application to a library of items rather than vinyls, removing the need for a square image.
With potentially several collections of things being owned by a specific person, and no one thing like vinyls to focus on, I had to implement the feature of adding separate albums for each different collection. This means that the user can add several albums and name them on creation, differing from my original fixed 'owned' and 'wish-list' albums.
Another small difference between idea and finished problem is I was unable to attach text, like a description or caption to each individual image. With vinyls out of the picture, I decided it would not be practical to have the user type in a description of each image on capture. The idea of the

lastFM/Discogs api was to download this data for the user, saving time and energy, and without this I thought there was no point. In order to make each filename unique, I had to use 'item' plus the current date and time. This differs from my original idea of '/vinyl' and a the number of the stored vinyl e.g '/vinyl50.jpg'. This would allow easy tracking of the number of vinyls stored at any time, or even the order they were purchased in.

Compared with other applications in the genre, my app is lacking in extra features, however provides enough functionality for it to be an effective application at documenting collections. The first application I took inspiration from, Discogs[1], is by far the most popular and feature-packed application for documenting vinyls. It has features my application does not including the option to purchase vinyls in your wish list and change the view of your vinyls. Each item can be shared to social media and information is displayed about each vinyl on click. It even has the ability to value your entire vinyl collection based on the price of each vinyl on their website.

A similar gallery app for android is 'Simple Gallery' by Simple Mobile Tools[8]. This application has a similar level of functionality to mine, providing the ability to have several named folders with images in each. It can play videos in the app, and supports pinch zooming and a cropping feature. Its simple design is very similar to mine, however the buttons in this app are in the action bar, rather than floating action buttons and it does not have an inbuilt camera. I received feedback from several potential users of my app both during and after production. Before starting my app, several people told me that the ability to search for vinyls and download album art was an important feature to have. However when this was deemed unachievable, I asked again what they thought of the camera. Many of the users suggested that the low camera quality means that the gallery will look less striking and appealing, with poor quality images. Concerns were expressed at the sheer amount of time it would take to photograph potentially several hundred vinyls, and that once this had been done, there is no option to re-order any of the images. As well as this, during production it was clear that because of the location of my stored images, the DCIM folder containing all the images stored in the phone's memory would appear.

Towards the end of building my app and nearing a finished product, the user-feedback was very different. As the purpose of the application changed, many people said it looks clean, and that the action buttons on the front page were useful and in good positions on the screen. Positive feedback was given about the expandable and zoomable images being displayed at a good size on the screen, and many people liked the fact that you can view your own images in the DCIM folder - some even suggesting that with improvement this app could be used as a replacement for the phone's own gallery. Negative feedback included no option to delete images after capture and gallery does not update unless the user returns to the album page of the application.

Many improvements could be made to my app to develop it further and ready it for release on the app store. One of the major improvements would take me closer to my original vision, and would obviously be adding in a search function and downloadable images. Another small but major improvement would be the inclusion of a toggle for the flash on the camera activity, as in many situations the phone cannot recognise when a flash may be better. Many much smaller improvements would also better the user-experience and include being able to choose an image to display as the album cover rather than the most recent image added to the folder and an option to add a photo from a source other than the camera (e.g 'DCIM' to folder 'Coins'). The option to share individual/groups of photos on social media is also a feature that is worth including, especially considering the younger members of the target market.

Further improvements include:

- Re-order albums and images within albums
- Rename albums and images within albums
- Crop images
- Change theme of app (Dark/Light/Day/Night)

5 Personal Evaluation

During the process of designing and making my project I encountered many problems and challenges, some of which drastically changed the end-product. The first major challenge I faced was the implementation of the api I decided to use. The Last FM api[9] appeared to be exactly what I needed, providing data on artists, albums and everything music related. I tried endlessly to get the api working with my program, however a lack of documentation and worked examples meant although I could connect to the service, I was unable to download any data including album art. I referred several times to two example programs on github[10][11] however I could not implement code either of these sources in my own project. After giving up on the last FM api, I attempted to use the discogs api to achieve the same thing. This api resulted in the same problems for me though, and I was also unable to use it. I was unaware whether I would be able to download the album art from this source and save it to the phones external memory too. After exhausting both my desired apis, I studied the use of the Google image search[12] and Bing image search[13] apis only to find that the Google images api was depreciated in 2011 and that bing images was a paid service- purchasing this being a risk I would not take if I was not certain it would work. Furthermore I was again unsure whether I would be able to save the desired image to the phone's memory.

Another problem I had with depreciated apis was the camera api[14]. After following the example in the Napier workbook, I quickly found out that with Android 5.0 (api 21 and higher) to use the camera the camera2 api[15] must be used. This proved to be a huge issue for me as there are few worked examples online, and although documentation was available on the android developers website, I was unable to understand how to implement this in my own program. I had to refer closely to a basic example published by Google on github[6], and in my final application ended up using a large portion of the code from here to add the camera feature to my application. Several aspects of the provided code did not suit my needs, and I had to change several parts to fix issues, of which a few appeared in the issues tab on github.

A major problem that surfaced late on in my development was the gallery not updating to show new images. Once an image is captured, whether in an app or on the phones default camera, the gallery must be updated in order to show the new images, otherwise the phone is unaware that a new image has been captured and should be displayed. The small section of code I used to perform this update initially only worked after a second image had been taken, prompting the first to appear. In order to fix this so that an image is shown immediately after captured, I had to move the code around several areas in my project. I tried updating the gallery on the launch of the application, on the launch of a new activity, on the changing of a layout and on the click of the capture image button. After moving the code around several places, and a lot of trial and error, I achieved a gallery update on availability of an image (right before it is saved). However even still, the gallery currently only updates when the user returns to the main album page, I believe because when the back button is used it returns to the previous screen in the previous state. Ideally this should happen as soon as an image is taken.

Other issues I had which caused me problems included some lines of code overriding others when programming my floating action buttons (FABs), imports missing in my classes due to not having auto imports on and issues with the action bar, which I was was unable to make work as desired. Due to bad weather, I was also unable to get support during my scheduled classes, which meant I had to change slightly to avoid an issue I was having.

Overcoming each challenge, both individually and when I have several issues at once could take days of working and problem solving. To fix many of these problems, I started with writing a basic plan of what I needed to fix or achieve, and some basic pseudocode which I could try and follow. I referred to issues other people have had on stack overflow and this was a good way to fix many of the smaller issues with imports, version control and basic bits of code. This was also the case with Napier's workbook, as many of the worked examples included contained clues or sections of code which could help me. For more advanced issues referring to full projects on github was good at sparking fresh ideas about how to fix issues and making me think of different approaches to achieve tasks. I watched YouTube videos from app developers and websites, and read comments referring to these. The most effective way I used to fix issues in my program though was basic trial and error. By debugging on Android Studio, I was able to pinpoint the line of code which caused an error, even though the program was compiled with no errors. This refers back also to moving sections of code around to several places, as I had to do this more than once to find the best place for code.

Thinking about what kind of application I wanted to make very early on, I think I started my project early and left enough time to produce a working application. I added the basic features that I wanted and adapted well when something became unachievable. However major issues I faced during the implementation that I did not expect caused me to lose a lot of time. Part of the planning that I could have done better involves taking into consideration that problems could occur, and leaving more time to compensate for this. It would also have been beneficial for me to seek help at university rather than changing my program when I cannot get something to work. I feel like my performance on this coursework has been good based on the amount of time and effort I have dedicated to it, and the thought that went into my concept.

In planning and designing my first android application I have learned a lot about the importance of making sure resources are available to you, in a way you can utilise before you start your project. I believe it's important to decide beforehand on an application that is achievable in the timescale that is available, and also matches your experience to an extent, however a challenge can be good. In facing many problems and obstacles, I have learned that sometimes I need to change and adapt certain parts of my project in order to make it work. I think after this experience I now understand the importance of feedback from your peers who can help determine points of improvement, and the great feeling that comes when your app is complete.

References

[1] "Discogs application, https://www.discogs.com/app,"

[2] A. Winistorfer, "I tried out 5 vinyl collection management apps, http://www.vinylmeplease.com/magazine/i-tried-out-5-vinyl-collection-management-apps/,"

[3] "App crawler, http://appcrawlr.com/ios-apps/best-apps-vinyl-collection,"

[4] "https://pixabay.com/en/camera-diaphragm-film-icon-lens-2766307/,"

[5] Shajib, "Create a photo gallery app in android - android studio, https://www.androstock.com/tutorials/create-a-photo-gallery-app-in-android-android-studio.html,"

[6] googlesamples, "android-camera2basic, https://github.com/googlesamples/android-camera2basic,"

[7] lopez.mikhael, "How to get an android camera2 with a 1:1 ratio like instagram, https://stackoverflow.com/questions/34638651/how-to-get-an-android-camera2-with-11-ratio-like-instagram,"

[8] Simple-Tools, "Simple gallery, https://play.google.com/store/apps/details?id=com.simplemobile

[9] lastfm, "Last fm api - https://www.last.fm/api/intro, http://ws.audioscrobbler.com/2.0/,"

[10] c99koder, "lastfm-android, https://github.com/lastfm/lastfm-android,"

[11] tgwizard, "Simple lastfm scrobbler, https://github.com/tgwizard/sls,"

[12] Google, "Google image search api (deprecated), https://developers.google.com/image-search/,"

[13] Microsoft-Azure, "Bing image search api, https://azure.microsoft.com/en-us/services/cognitive-services/bing-image-search-api/,"

[14] Android-Developers, "Camera api, https://developer.android.com/guide/topics/media/camera.html,"

[15] Android-Developers, "Camera 2 api, https://developer.android.com/reference/android/hardware/camera2/package-summary.html,"