Manga Feed

by

Andre Maurice Gregoire

B.S., Kansas State Unviersity, 2014

A REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department Of Computing And Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2017

Approved by:

Major Professor
Dr. Mitchell L. Neilsen

Copyright

Abstract

Mobile technology has advanced significantly in the last decade, devices have gotten smaller and much more powerful. Because mobile devices are so accessible, they have spread everywhere and are used extensively by an immense portion of the population for various tasks. Mobile devices are no longer a means to just communicate with another person, you can use them to take pictures, scan documents, create a Wi-Fi hotspot, or just be entertained.

E-Readers have been around for a while and implemented as a source of entertainment on many mobile platforms. However very few e-readers for Manga, which is a type of comic book, have been developed. Manga Feed, built on the Android platform, strives to help fill this void by providing a clean and simple to user interface for users to enjoy reading on the go. There are thousands of manga of various genres made available to users, offering entertainment to any individual interested in manga.

Table of Contents

List of Figures vii

List of Tables viii

1 Introduction 1

1.1 Project Description 1

1.2 Motivation 1

2 Android 2

2.1 Linux Kernel 3

2.2 Hardware Abstraction Layer 4

2.3 Libraries 4

2.4 Android Runtime 4

2.5 Java API Framework 5

2.6 System Applications 5

3 Android Components 6

3.1 Android Manifest 6

3.2 Activites 8

3.3 Fragments 9

3.4 Intents 11

3.5 Services 12

3.6 Data Storage 13

3.6.1 Shared Preferences 13

3.6.2 Internal/External Storage . . . 13
3.6.3 SQLite Database . . . 13
3.6.4 Network Connection . . . 13

4 Design . . . 14
4.1 Class Diagram . . . 14
4.2 MVP . . . 15
4.3 User Interface . . . 16

5 Implementation . . . 17
5.1 Application . . . 17
5.2 Main Activity . . . 17
5.2.1 Recent Fragment . . . 19
5.2.2 Library Fragment . . . 19
5.2.3 Catalog Fragment . . . 19
5.2.4 Drawer Menu . . . 19
5.3 Manga Activity . . . 20
5.4 Reader Activity . . . 22
5.5 Data Acquisition . . . 25

6 Testing . . . 27
6.1 Unit Testing . . . 27
6.2 Integration Testing . . . 28
6.3 Beta Testing . . . 28

7 Conclusion and Future Work . . . 29
7.1 Conclusion . . . 29
7.2 Future Work . . . 29

Bibliography . . . 31

List of Figures

2.1 Android Architecture[1] 3

3.1 Example Manifest File 7

3.2 Launcher Activity Manifest 8

3.3 Activity Life Cycle[2] 9

3.4 Fragment Life Cycle[3] 11

4.1 Manga Activity UML 15

5.1 Main Activity 18

5.2 Manga Activity 20

5.3 Status Filter 22

5.4 Reader Activity Portrait 23

5.5 Reader Activity Landscape 24

5.6 Chapter Data Retrieval 25

List of Tables

6.1 Integration Tests 28

Chapter 1

Introduction

1.1 Project Description

Manga Feed is an application built using the Android Framework for mobile hand held devices. It is an e-reader app developed to read Manga, a style of Japanese comic books and graphic novels. Users are able to build their own library and read up to date Manga as it is published and uploaded to the selected source website. A catalog of Manga for each source is present for the user to find something of interest as well as various ways to search, filter, and categorize the collection of Manga.

1.2 Motivation

The motivation of this application is to further develop knowledge in working with the Android Framework as well as provide a means for users to easily read manga on the go. Mobile technology, devices, and the part they play in peoples day to day lives has grown immensely. With this in mind as well as a personal desire for a mobile reader, led me to start work on this application.

Chapter 2

Android

Android is a mobile operating system developed by Google based on the Linux kernel.[4] This was built for mobile devices such as smartphones and tablets, where interaction is primarily based on touchscreen gestures such as: swiping, pinching, and tapping. It is also open source making it possible for a user or company to customize and produce a new flavor of android for personal or professional devices. Applications written for Android OS are generally written in java using the SDK, however you can also use native languages such as C/C++ with the NDK.[1]

Figure 2.1 below illustrates androids platform architecture. It is separated into five layers with six sections which are the Linux Kernel, Hardware Abstraction Layer, Libraries, Android Runtime, Java API Framework, and System Applications.[1]

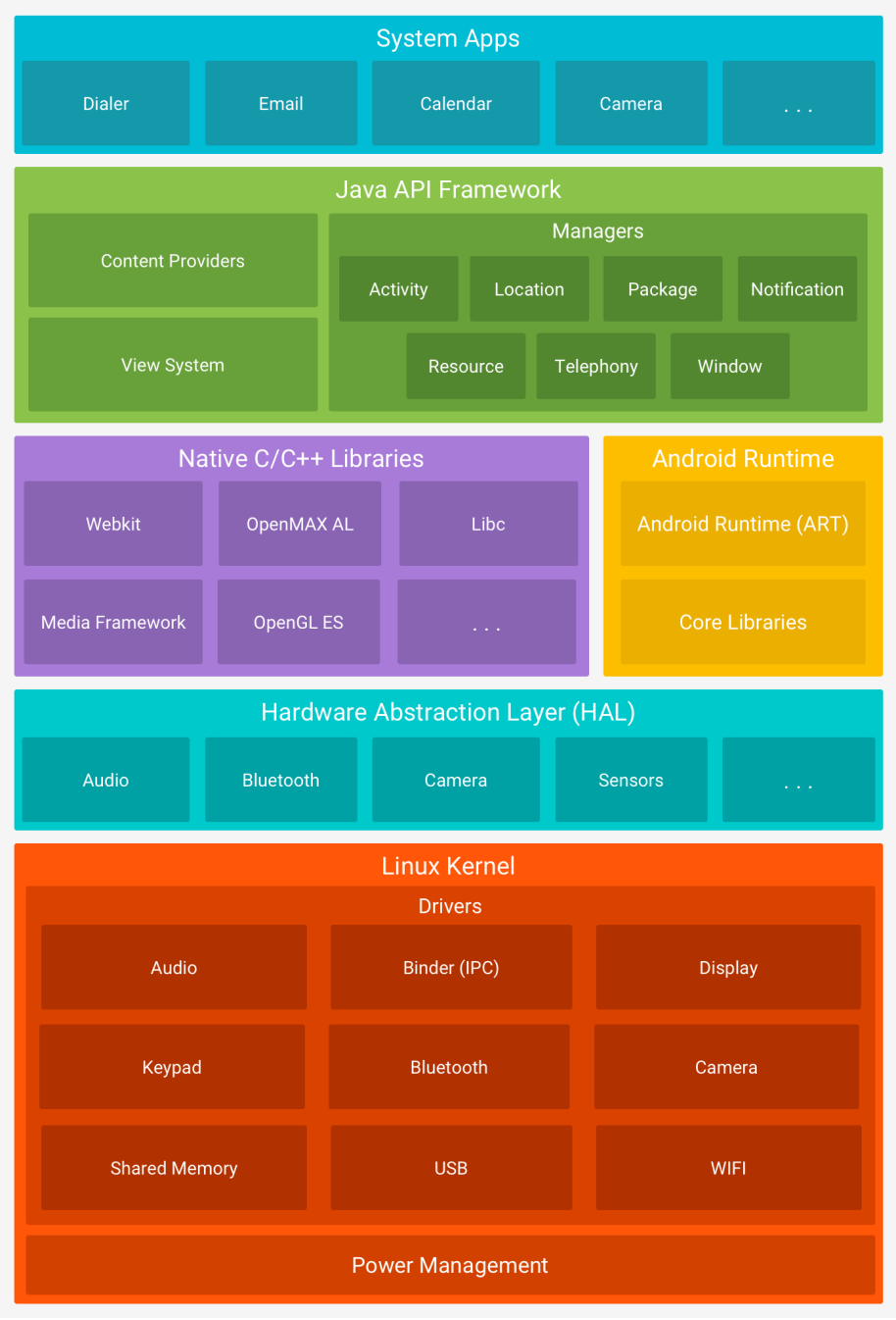


Figure 2.1: Android Architecture[1]

2.1 Linux Kernel

The linux kernel acts as a level of abstraction between the device hardware and contains the hardware drivers essential to the variety of devices that operate using Android including WiFi, Bluetooth and Camera drivers as well as several more.[5] The kernel also handles tasks

like threading and memory management for the Android Runtime (ART).[1]

2.2 Hardware Abstraction Layer

The Hardware Abstraction Layer (HAL) is a set of libraries that are used to provide access to the hardware capabilities of a device to the Java API Framework.[1] This can be something like using the camera on your smart phone, whenever you interact with an application that utilizes the camera of a device the application makes an API call to access the camera hardware and the library module for that hardware component is loaded.[6]

2.3 Libraries

The Native C/C++ libraries are used in core components used in Android such as the Android Runtime and the Hardware Abstraction Layer.[1] Some of these libraries have Java facing APIs such as the OpenGL library making it easy for application developers to draw and interact with 2D and 3D graphics.[5] Developers can also interact with these libraries through use of the NDK if they do not have a Java facing API or if they just need to squeeze out better performance.

2.4 Android Runtime

The Android Runtime holds two components: the Android Runtime (ART) — used to compile Android applications, and the Core Libraries — the libraries that enable developers to create applications using standard Java. Devices running Android 5.0 Lollipop and newer use ART as its means to compile and run applications.[7] In versions previous to Lollipop

the Dalvik Virtual Machine was used.[7] The switch from Dalvik to ART saw performance increases in CPU floating operations, RAM operations, and storage operations.

2.5 Java API Framework

The Java API Framework are the set of APIs that form the building blocks needed to create an application. It is comprised of a View System; which "can be used to build an app's UI, including lists, grids, text boxes"[1] and much more; a Resource Manager that provides "access to non-code resources such as localized strings, graphics, and layout files"[1]; a Notification Manager "that enables all apps to display custom alerts in the status bar"[1]; an Activity Manager "that manages the lifecycle of apps and provides a common navigation back stack"[1]; and Content Providers "that enable apps to access data from other apps"[1]

2.6 System Applications

The System Applications layer is the top layer of the Platform Architecture, it is the layer that holds all applications that come default on the phone such as Contacts, Calculator, Email, etc. as well as the applications developers will write themselves and distribute personally to customers on an application market. These default applications "have no special status among the apps the user chooses to install"[1] for example "the users default web browser, SMS messenger, or even the default keyboard"[1] can all be replaced.

Chapter 3

Android Components

Android applications make use of several key components that act as the building blocks of an android application.

3.1 Android Manifest

The AndroidManifest.xml file must exist in every application and must have that exact name. It holds essential information about the application that the android system requires in order to run any application code. The manifest file:

- "Names the java package for the application." [8]
- "Describes the components of the application that consist of: activities, services, broadcast receivers, and content providers." [8]
- "Determines the processes that host the application components." [8]
- "Declares the permissions that the application must have in order access the protected parts of the API." [8]
- "Lists the instrumentation classes that provide profiling and other information as the application runs." [8]

- "Declares the minimum level of the Android API that the application requires" [8]
- "Lists the libraries that the application must be linked against" [8]

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.teioh.m_feed">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:name=".MFeedApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:largeHeap="true"
        >
        <activity
            android:name=".UI.MainActivity.MainActivity"
            android:label="@string/app_name"
            >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".UI.MangaActivity.MangaActivity"
            android:label="@string/title_activity_manga"
            android:theme="@style/AppTheme">
        </activity>
        <activity
            android:name=".UI.ReaderActivity.ReaderActivity"
            android:label="ReaderActivity"
            android:theme="@style/ReaderTheme">
        </activity>
        <activity
            android:name=".UI.MainActivity.LoggingActivity"
            android:label="LoggingActivity"
            android:theme="@style/AppTheme">
        </activity>
        <service android:name=".RecentUpdateService" ></service>

    </application>

</manifest>
```

Figure 3.1: Example Manifest File

3.2 Activites

The Activity class is the entry point for users to android applications and provides the interface users will interact with.[2] In order for activities to be used by an application they must first be declared in the AndroidManifest file.[2] The first activity that is loaded when an application is launched is called the main activity, it is specially registered in the Android-Manifest file by means of an intent-filter.

```
<activity
    android:name=".UI.MainActivity.MainActivity"
    android:label="@string/app_name"
    >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figure 3.2: Launcher Activity Manifest

To create an Activity the activity class must inherit from Androids Activity class, and functions by a series of callbacks to handle the transitions between each stage of the activities lifecycle.

An activity usually implements one screen of an application, for example in the Contacts application that comes by default on all android devices, the first screen that shows a list of the users contacts would be an activity. After selecting a contact or creating a new contact, both could be separate activities depending how the application was designed.

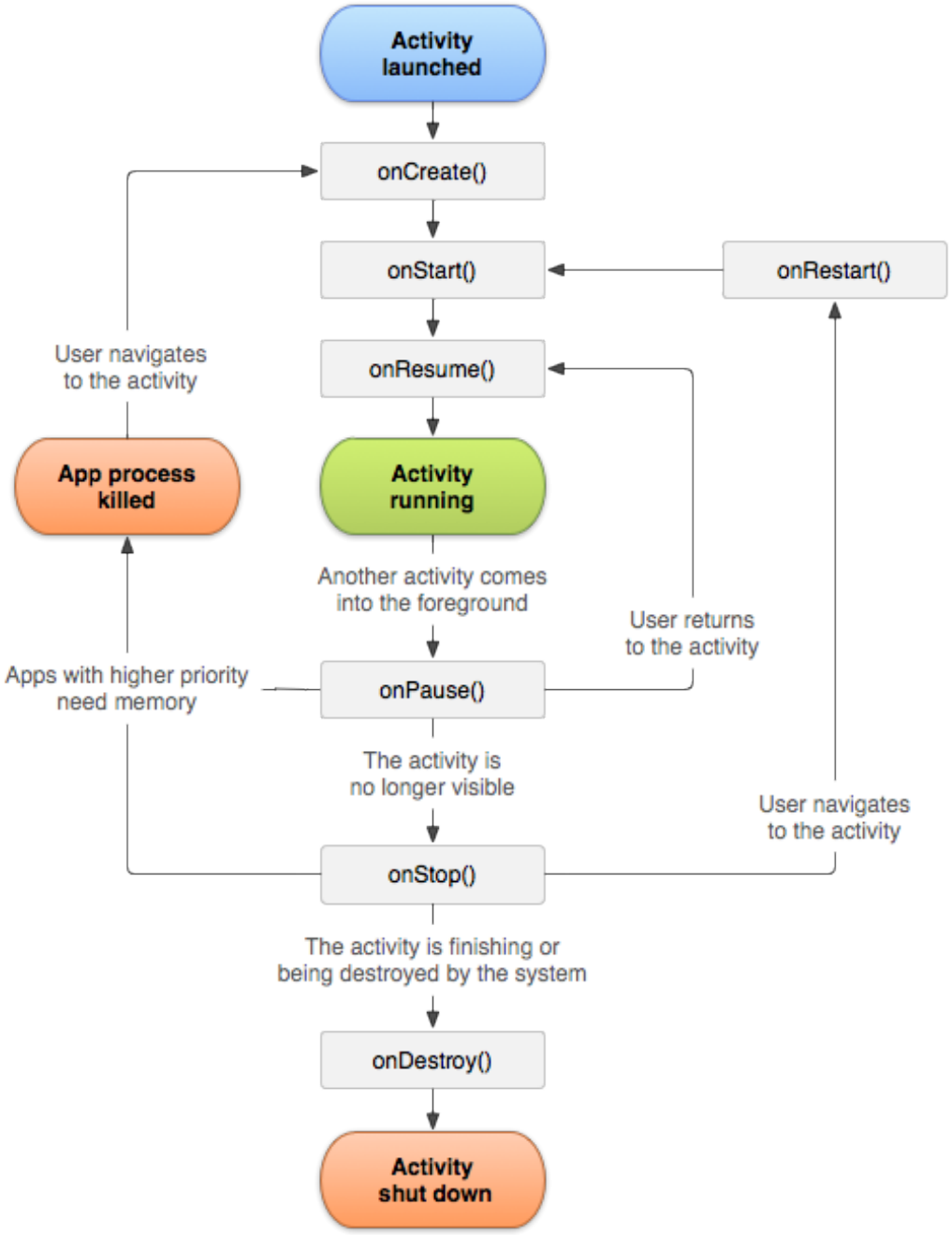


Figure 3.3: Activity Life Cycle[2]

3.3 Fragments

Fragments make up portions of an activities interface.[3] There can be one fragment that takes up the whole activity or multiple fragments to split up functionality for a single activity into sections. Each fragment has its own lifecycle and receives its own input events.[3]

"Fragments must always be embedded in an activity"[3], they can be added and removed from activities at any time the activity is running.[3]

Very similar to activities, fragments also rely on a callback system in order for them to handle its different states. However because fragments make up parts of an activity their life cycle is slightly different, as show in figure 3.4 below. When implementing a subclass for a fragment there are a few options: the base Fragment class; a DialogFragment, which will show a floating dialog; a ListFragment, which is used to hold and display a list of objects; and a PreferenceFragment, which is useful for building something like a settings page for an application.[3]

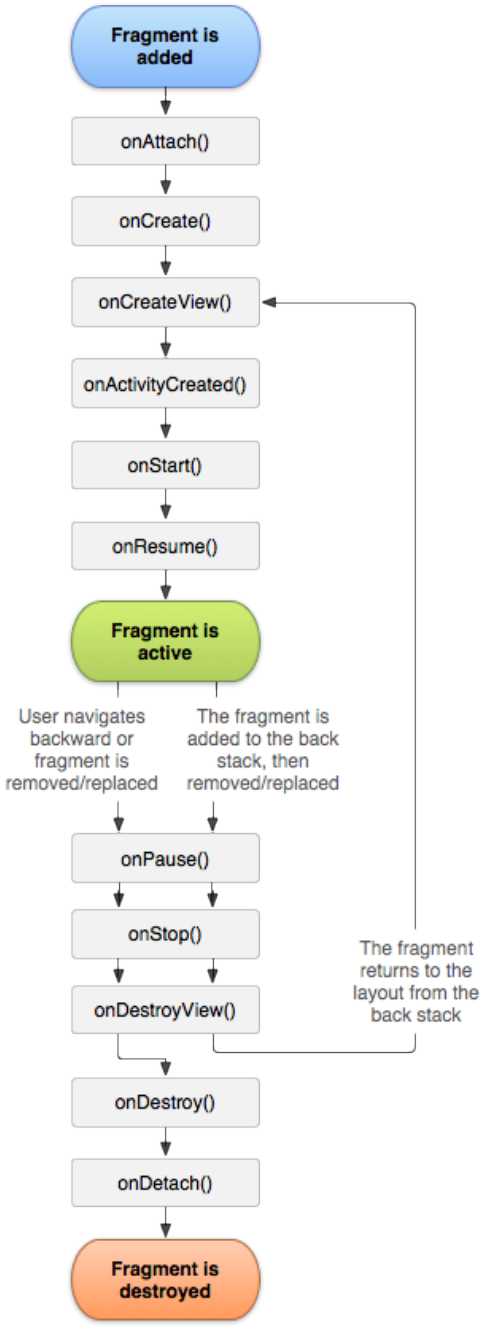


Figure 3.4: Fragment Life Cycle[3]

3.4 Intents

Intents have three primary uses, to start an activity, broadcast messages to broadcast receivers, and to start and communicate with services.[9] Intents can be thought of as structure that holds information about an action that is trying to be performed. Information that

an intent holds are its action, data, category, type, component, and the extra information the developer can pack into an intent.

There are two types of intents, implicit and explicit.[10] Implicit Intents are used to start an activity in another application.[10] They are called implicit because these kinds of intents do not require the developer to specify the app component that is used to start. Explicit intents on the other hand require the app component to be set which specifies the exact class to be run.[10]

3.5 Services

A service performs long-running operations in the background outside of the users view. This means unlike other application components like activities and fragments there is no user interface, however these components can start a service. Services are implicitly bound to its parents lifecycle, this means that if the component that starts a service exits, the service will continue to run until it is complete.[11]

There are three types of services: Scheduled, Started, and Bound. Scheduled services are services that are launched by an api such as the JobScheduler that launches takes in information such as networking and the time of execution and starts the service accordingly.[11] Started services usually "performs a single operation and does not return a result to the caller."[11] This can be a task like downloading a file. A bound service is similar to a started service except as the name implies it is bound to its caller and acts as a client-server interface so its parent can interact with it. This type of service will run as long as a component is bound to the service.

3.6 Data Storage

Android offers several options for developers to store persistent data. The correct storage medium chosen will depend how much data a developer is storing and the kind of data they are storing. The options available are: Shared Preferences, Internal/External Storage, SQLite Database, Network Connection.[12]

3.6.1 Shared Preferences

Shared Preferences allows the developer to store and retrieve key value pairs of primitive data types.[12] These are quite useful when storing user preferences that developers may make accessible in a settings page.

3.6.2 Internal/External Storage

This type of storage is used for general files such as images, PDFs, and so on. They will be stored on the device or memory card and can be retrieved or interacted with as needed. Files saved to internal storage are private to the application however when saved to external storage files are public.[12]

3.6.3 SQLite Database

Android provides full support for SQLite databases, any database that is created is private and only accessible by activities in the application.[12]

3.6.4 Network Connection

Network data storage is not on the device but somewhere that is accessible over a network connection. This can be cloud based storage, or a web backend that the application is connected to.

Chapter 4

Design

The design of this application is split into two categories: system design, and user interface (UI) design. The system design is important to solve the practical requirements of the application in a flexible, efficient, and secure way. It is also important for users to not feel frustrated while using an application especially if the application is suppose to be for entertainment purposes, or will be used often.

4.1 Class Diagram

A class diagram is a structure diagram that describes a system by the systems classes as well as each classes attributes, functions and their relationship among the other objects.[13] Figure 4.1 below shows the class diagram for the Manga Feed application using the Unified Modeling Language (UML), because of its size the attributes and functions of the classes were omitted. UML is a general purpose modeling language that provides a standard way to visualize how a system is designed.

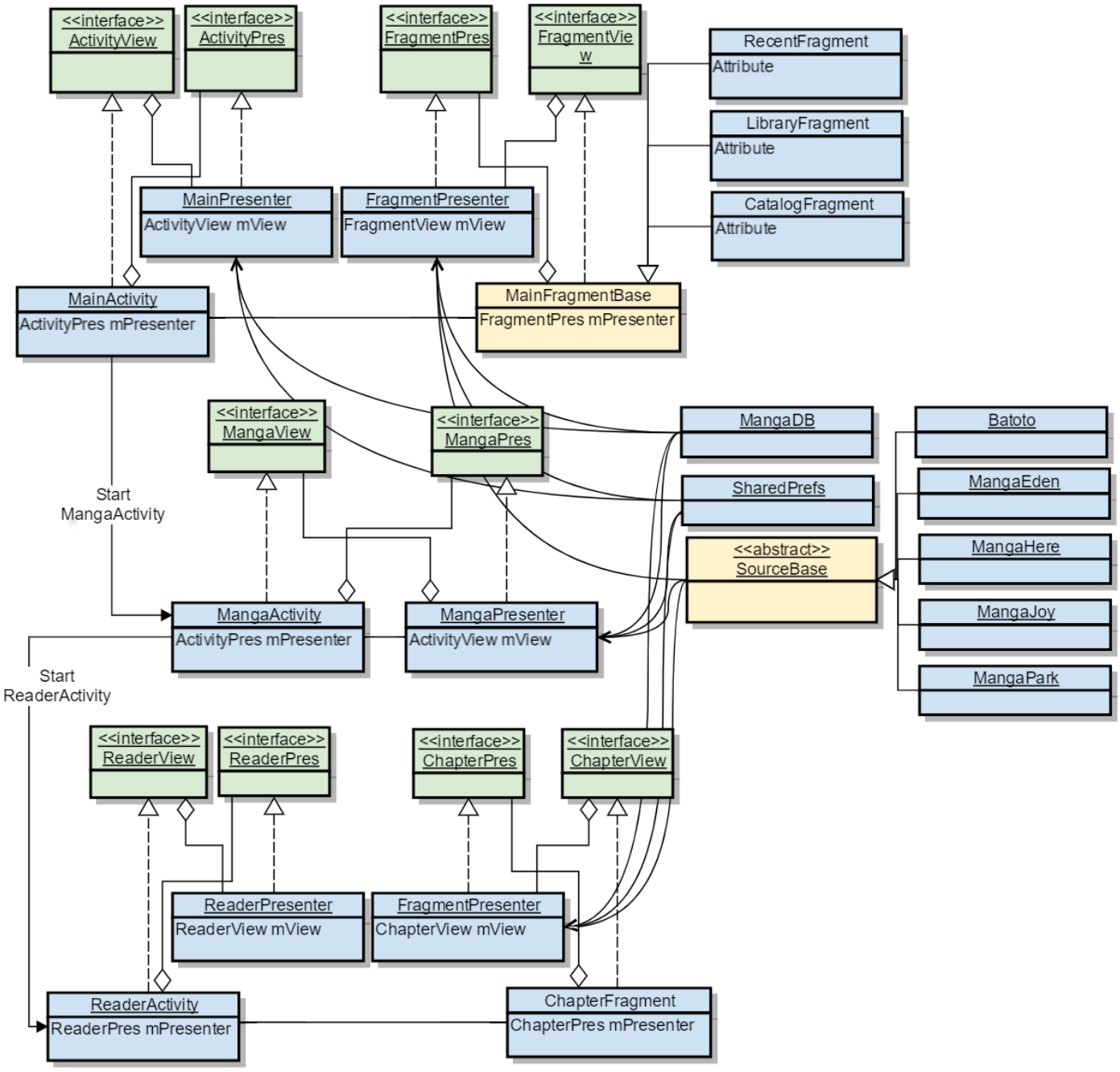


Figure 4.1: Manga Activity UML

4.2 MVP

The project follows the Model View Presenter(MVP) architectural pattern. MVP separates logic from the views and places it in the presenter, the idea is to decouple display logic from business logic. Interactions with the UI are received by the view and forwarded to the

presenter where any logic is taken care of. If the updates made within the presenter warrant the view to be updated, changes are then forwarded to the view where the UI is refreshed. The model acts as an interface to the data and should generally only be accessed from the presenter.

In order for the View and the Presenter to know how to communicate with each other, an interface is implemented by each of their respective classes. This interface will outline the specific functionality that each needs to know about each other. For example if there is an OnClick handler in the View, then the interface for the presenter would need to define a function that the View can then call to handle the logic behind the click event. If the logic done by the presenter then required the view to be updated or partially updated the views would need to define a function for updating the user interface.

4.3 User Interface

The user interface is the graphical interface that as the name suggests, users interact with to navigate and utilize it for its intended purpose. Creating an interface that appeals to a plethora of users is quite difficult, the interface of Manga Feed has changed and some some layouts have been completely re-designed since its inception. As a developer what you believe is intuitive, or simple, is not always the case and because of this outside input is very useful. User input can let the developer know what they've done right, what they've done wrong, or if an element just feels out of place. The users played a large part in many faucets of how the application currently runs, features that have been implemented, but especially the maturation of the UI design. How they were a part of the development will be discussed in Chapter 6.

Chapter 5

Implementation

5.1 Application

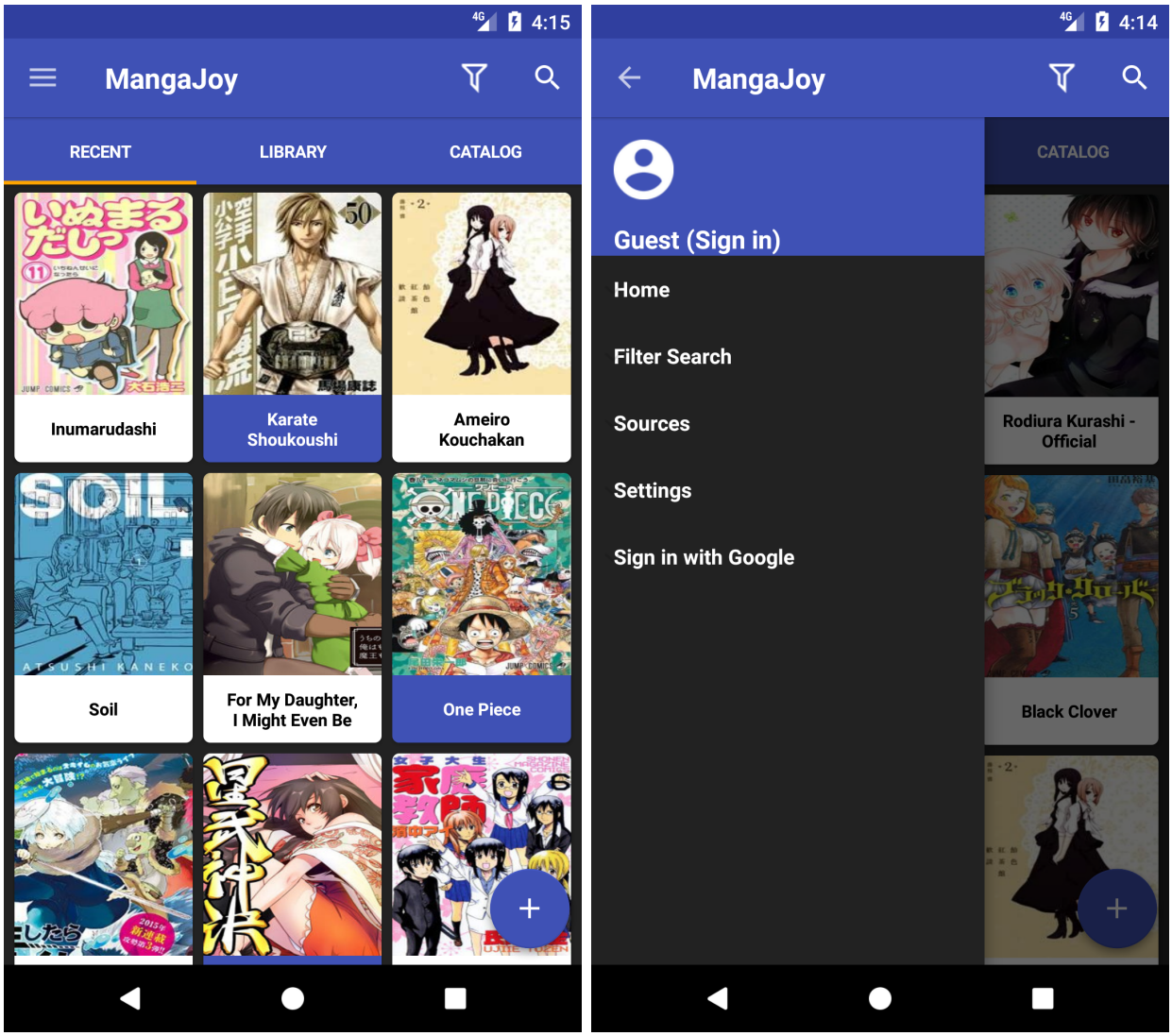
Starting an android application begins with creating a new android project in Android Studio. This will create a basic project structure for the source code, various resource files, and other assets such as images of a pre-built database. In the past android applications were built with Eclipse plus and Android Plugin however Android Studio is now the supported Integrated Development Environment (IDE).

After creating the project the developer can start laying out the ground work for how the application will be organized, look, and function. This chapter will provide an analysis of the application Manga Feed operates, how it was designed and organized.

5.2 Main Activity

The main activity is the first screen that is shown when the application is launched. This activity is recognized by the android operating system as the launcher activity based on the android manifest file, mentioned in the section 3.2. This activity has three fragments:

a RecentFragment, a LibraryFragment, and a CatalogFragment. The CatalogFragment, and LibraryFragment use the same layout resource and the RecentFragments layout is almost exactly the same with the exception of it containing a swipe refresh layout.



(a) Main Activity (b) Hidden drawer menu

Figure 5.1: Main Activity

5.2.1 Recent Fragment

The RecentFragment is responsible for retrieving updates from the default or user selected source. It makes an asynchronous call to query the sources website and retrieves the necessary HTML to parse and turn into Manga objects. These objects are returned to the view and shown to the user as can be seen in figure 5.1a above.

5.2.2 Library Fragment

The LibraryFragment queries the local SQLite database on the device and runs a query to find any manga items the user has decided to follow from the currently selected source, further details will be discussed in section 5.3. The list of manga from the query are then returned to the view and shown to the user.

5.2.3 Catalog Fragment

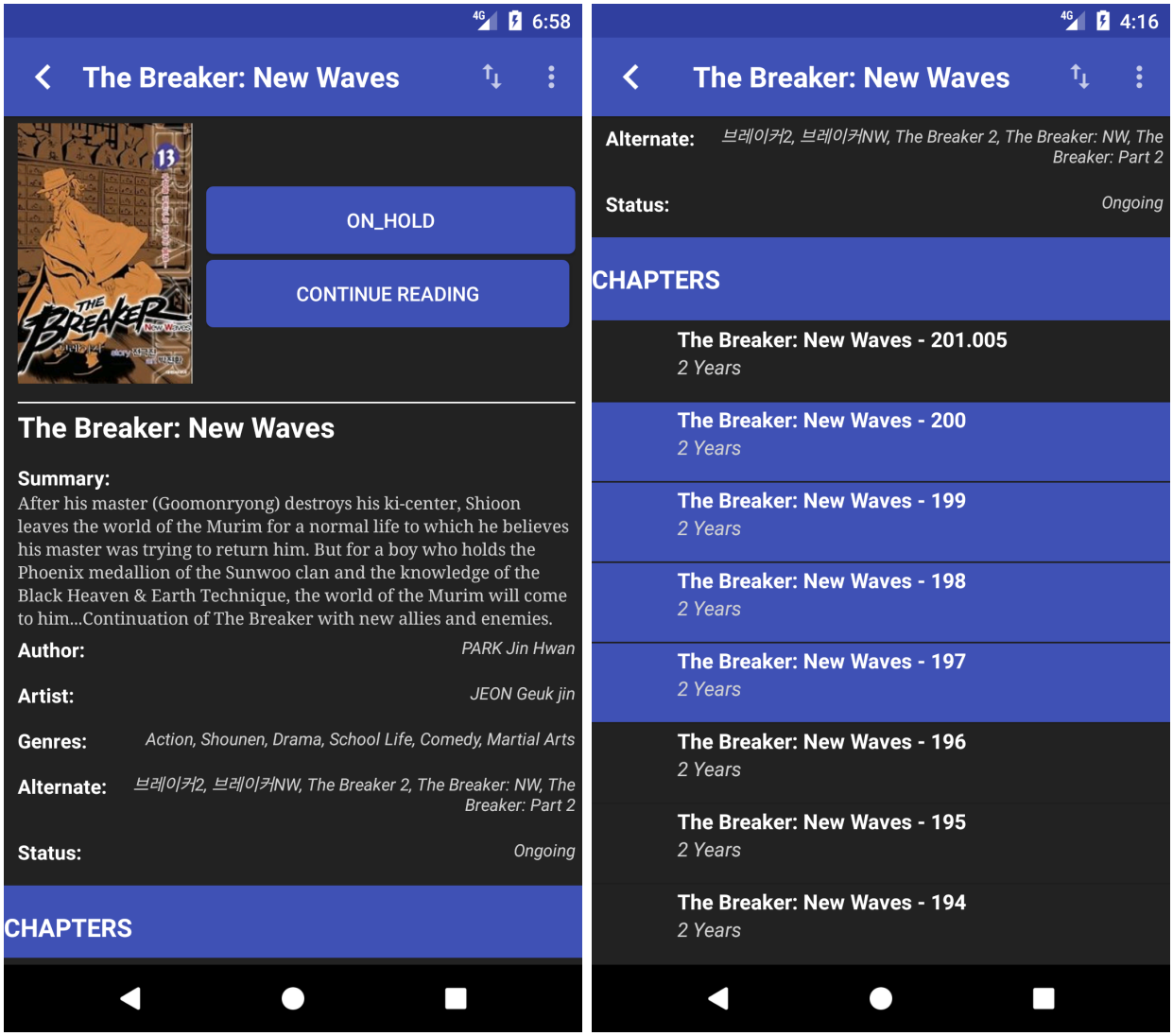
The Catalog Fragment functions exactly the same as the LibraryFragment however instead of retrieving a list of manga followed by the user, it will return a comprehensive list of manga that the currently selected source provides.

5.2.4 Drawer Menu

The drawer menu contains several options for the user to choose from, shown in figure 5.1b above. There is a home button, to return the user back to the main activity, this will hide the drawer menu; There is a filter search that brings up a non simple dialog for the user to choose different criteria to filter the shown manga in the Recent, Library, and Catalog fragments; The sources section expands and collapses when selected to show the different sources Manga Feed supports; The settings button will bring up the settings fragment that has a few personal preferences and support functionality for the user; and finally there is an option to sign in using a google account.

5.3 Manga Activity

The manga activity is the next activity that is accessible to the user by selecting an item from one of the three fragments in the previous activity. At the start of activity the application acquires the URL to the selected manga, this is used as a unique key to query the local database and retrieve the various available information.



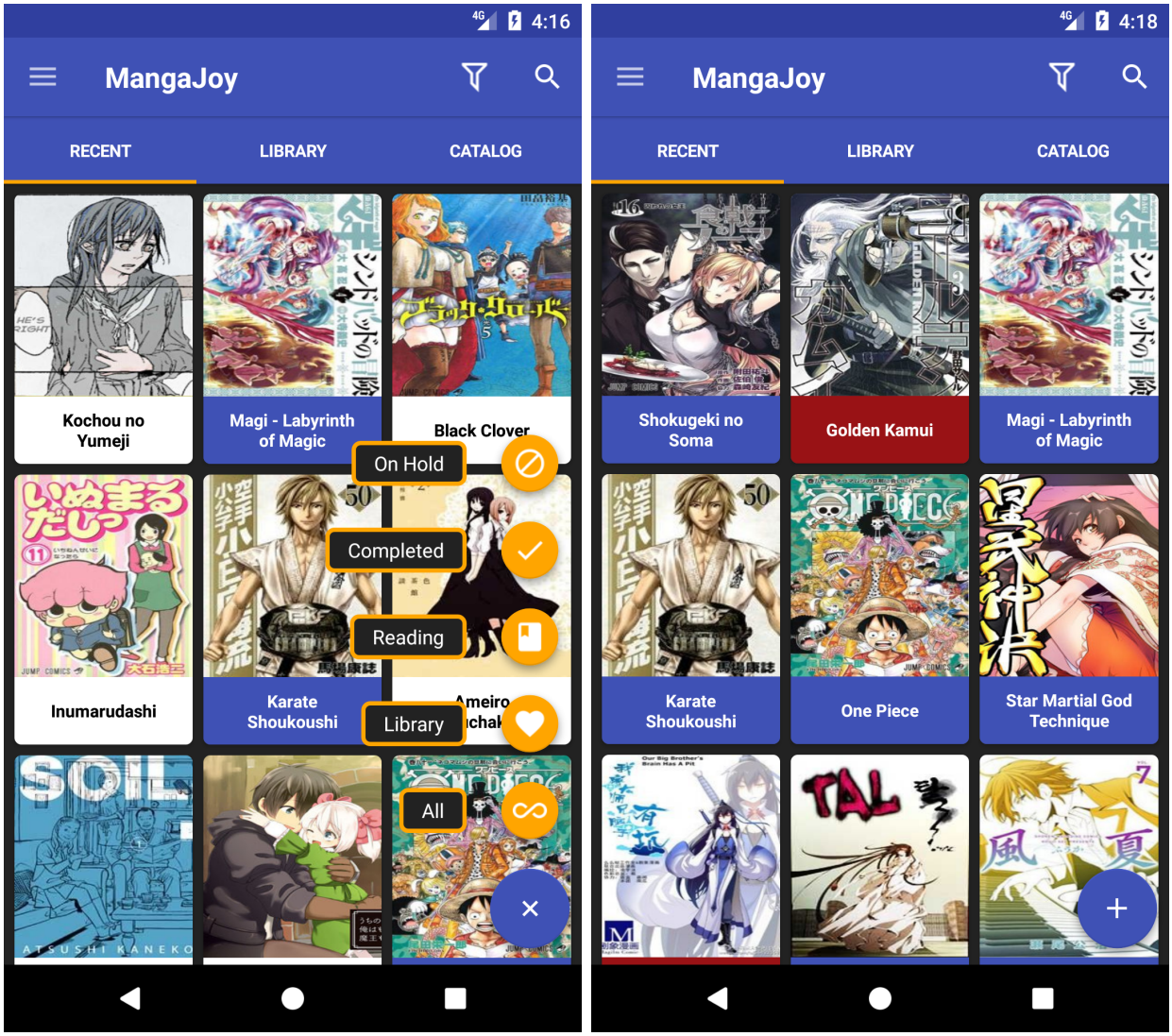
(a) Manga Information (b) Manga Chapter List

Figure 5.2: Manga Activity

It is comprised of a simple list view and a custom header that is attached to said list view. The header presents information about the manga that the reader may find useful such as; a summary, or brief introduction to the story; the name of the author; the name of the artist; the various genres it has been labeled; alternate names it may be called; and the status of its publication. All of these aforementioned bits of information are available on a case by case basis due to the fact that all information is made available by the various sources where they may be incomplete.

The actual list portion of the activity shows the various chapters that are available for the selected manga. Each item will show the name of the chapter and if available the time it was posted. If the user has added the manga to their library they will see something similar to figure 5.2a above. When a chapter has been selected the next activity will load, this will be discussed in section 5.4. The selected chapter will also be highlighted with the primary blue color of the application, see figure 5.2b above, this indicates that a user has read or at least started to read the selected chapter.

There are a few things the user can do such as add the manga to their library, and switch the chapter order from descending to ascending. If the user chooses to add the manga to their library the activity will begin to keep track of the chapters the user has interacted with, as was mentioned before. A few more options also become available, such as clearing the list of chapters the user has interacted with; the user can remove the manga from their library; and the user can also choose between three options to categorize the manga: Reading, Completed, and On Hold. This property can be utilized in the main activity as a filter to search for a specific category and each category is displayed with a different color, see figure 5.3 below.

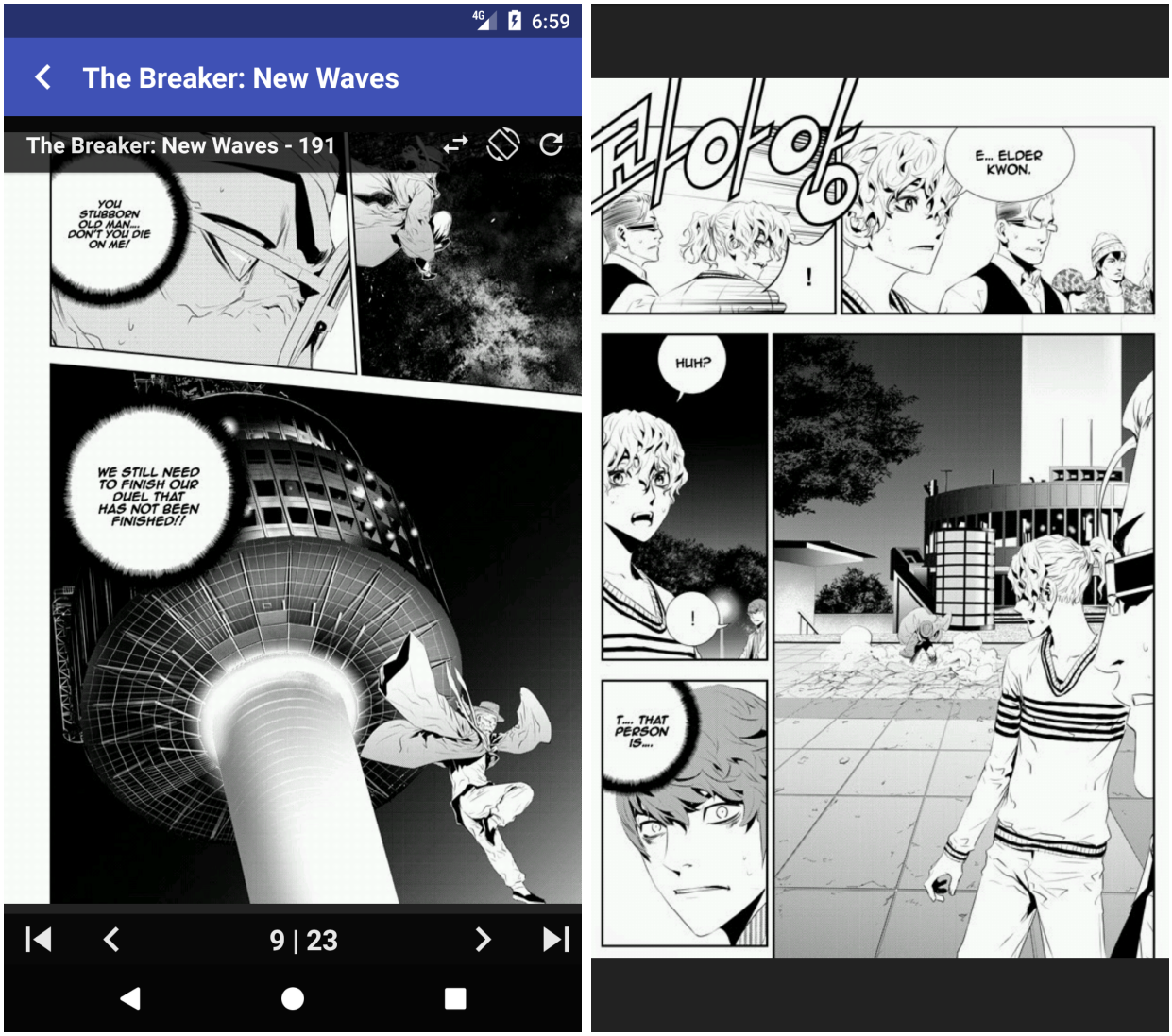


(a) Status Filter Menu (b) Result of selecting the Library filter

Figure 5.3: Status Filter

5.4 Reader Activity

The reader activity is the last activity the user can navigate to by selecting an item from the chapter list in the Manga Activity. It is comprised of a custom view pager, titled as NoScrollViewPager; two headers and one footer.

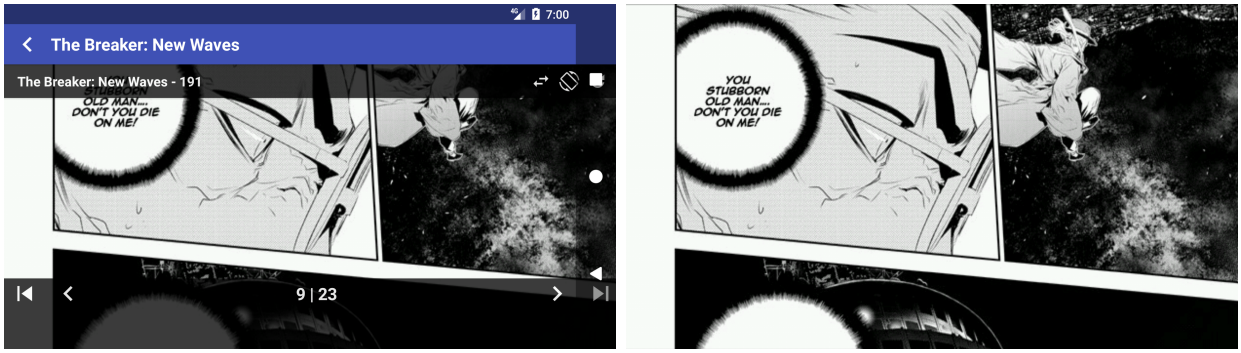


(a) Reader (b) Reader - hidden

Figure 5.4: Reader Activity Portrait

The NoScrollViewPager disables the scrolling functionality of a view pager, and contains a set of fragments that represent a specific chapter from the Manga Activity that the user will be able to scroll through and read its content. This is done by each chapter fragment implementing another custom view pager, titled GestureViewPager to handle on screen gestures such as zoom, scrolling, and flings. Creating the NoScrollViewPager was necessary in order to allow the GestureViewPager to consume the users interactions, otherwise it would have consumed the interactions because it is the parent view.

The two headers contain the information such as the title of the manga, the title of the current chapter, and various features users may find useful while reading. As seen in figures 5.4a above and 5.5a below. The top header contains the title of the manga and an option to go back to the Manga Activity. The second header contains the chapter title, as well as these three options: the user may toggle the swipe direction to change pages from left and right to up and down; change and lock the screen orientation from portrait to landscape; and an option to refresh the current chapter, sometimes pages will not load for various reasons such as bad internet connection.



(a) Reader Activity (b) Reader - hidden

Figure 5.5: Reader Activity Landscape

The footer contains basic navigation buttons that can be seen in figures 5.4a and 5.5a above. The user can go to the previous chapter, go back one page, go forward one page, or skip to the next chapter. It also keeps track of which page the user is currently on and shows the total amount of pages the chapter contains.

The headers and footer are part of the reader activity so each chapter fragment can seamlessly transfer communication rights and update the various portions that change with each chapter such as the chapter title, current page and total number of pages. Because it belongs to the reader activity it overlays on the image viewer, these portions of the

reader activity can be toggled by single tapping the image and they will be hidden from view.

5.5 Data Acquisition

Data for the application consists primarily of images for the book covers and chapter pages, as well as information about the manga such as its various titles, author(s), artist(s), and genre(s). This information is obtained by scraping the various sources webpages and presenting it accordingly in the application. Sources generate there webpages dynamically with the same template for each item, because of this parsing the html tags directly and retrieving the relevant pieces of data was very straight forward. This is done by taking a snap shot of the webpage where its raw html is returned and parsed accordingly. Figure 5.6 below shows part of this while extracting the chapter elements.

```
private List<Chapter> scrapeChaptersFromParsedDocument(final Document aParsedDocument, final String aTitle)
{
    List<Chapter> lChapterList = new ArrayList<>();
    Elements lChapterElements = aParsedDocument.select("ul.chapter-list").select("li");
    int lNumChapters = lChapterElements.size();

    String lChapterUrl, lChapterTitle, lChapterDate;

    for (Element iChapterElement : lChapterElements)
    {
        lChapterUrl = iChapterElement.select("a").attr("href");
        lChapterTitle = iChapterElement.select("span").first().text();
        lChapterDate = iChapterElement.select("span").get(1).text();

        lChapterList.add(new Chapter(lChapterUrl, aTitle, lChapterTitle, lChapterDate, lNumChapters));

        lNumChapters--;
    }

    return lChapterList;
}
```

Figure 5.6: Chapter Data Retrieval

There are four queries that are made to retrieve information in this application. The first one is in the main activity where it retrieves a list of a sources recent updates. The second and third are after the user enters the manga activity, here the application will make separate queries to retrieve a manga's information and another to build its list of chapters.

The last query is when entering the reader activity, this is where a list of image urls is built and the chapter images are rendered for the user to read.

Chapter 6

Testing

Testing involves the execution of software and system components to evaluate properties of interest such as: meeting design requirements, responds to user input accordingly, and functions within an acceptable amount of time. To evaluate these and various other properties different testing techniques are adopted.

6.1 Unit Testing

Unit Testing is the process where the smallest testable units of a system are individually analyzed to verify correct performance. These units generally consist of verifying methods or a singular class. These types of tests are commonly written during development, and can later be a portion of the systems regression testing, which is used to verify updates made to the system do not negatively impact the separate modules of the system. The unit tests for this application are coupled with the set integration tests talked about in section 6.2 next.

6.2 Integration Testing

Integration testing is the testing used to ensure the group of modules and components that make up the system are combined and produce the desired output. This is also where hardware and software component interactions are verified to function as intended.

Table 6.1: Integration Tests

#	Test Case	Expectation	Result
1	Main Activity Item Select	Manga Activity is launched	Pass
2	Main Activity Text Search	Adapters perform text search	Pass
3	Main Activity Genre Search	Adapters perform genre search	Pass
4	Main Activity Hidden Drawer Toggle	Nav Drawer opens and closes	Pass
5	Main Activity Hidden Drawer Options	Nav Drawer options work	Pass
6	Main Activity Status Filter	Adapters perform status filter	Pass
7	Manga Activity Toggle Status	Manga status changes	Pass
8	Manga Activity Toggle Follow	Manga is followed	Pass
9	Manga Activity Toggle Chapter Order	Chapter order inverses	Pass
10	Manga Activity Item Select	Reader Activity is launched	Pass
11	Reader Activity Previous Chapter	Goes to previous chapter	Pass
12	Reader Activity Skip Chapter	Goes to next chapter	Pass
13	Reader Activity Toggle Screen Orientation	Orientation changes/locks	Pass
14	Reader Activity Toggle Vertical Scroll	View pager scroll direction changes	Pass
15	Reader Activity Toggle Refresh Chapter	Chapter data is refreshed	Pass
16	Reader Activity Page Forward	Goes to next page	Pass
17	Reader Activity Page Backward	Goes to previous page	Pass

6.3 Beta Testing

Versions of the application are released to a group of users in order to ensure as few faults and bugs are present when it is released to the public. This can also overlap with usability testing where users are asked to complete tasks and in general just use the application and give feedback about their experience. This portion of testing is how users would give input about bugs, slowness, or their thoughts and suggestions on the design and application itself.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Manga Feed has been designed and implemented to gain and further develop knowledge in Java Programming and the use of the Android Framework. It has been tested on several mobile devices, and is currently used by several people, myself included. This project has taught me about the importance of up front design, especially when new features are requested and having to consider how they will relate and interact with the current system design. Working with the users was also an interesting aspect of the project, seeking the opinions of the people using the application provides useful feedback for creating a better application.

7.2 Future Work

There are a few things I have been planning to implement, as well as requests from the users. Developing a web backend to store user libraries will provide multi device support to users, this can also be continually expanded after it is built such as being able to sync mobile databases with the server to keep devices up to date. A popular request from the users, has been the implementation of an offline reader, this will require the design of a download ser-

vice to manage and organize chapter information on a device's internal or external storage. I will continue to provide maintenance and take feature requests into consideration as time permits while moving forward.

Bibliography

[1] Platform Architecture, 2017. https://developer.android.com/guide/platform/index.html.

[2] Activities, 2017. https://developer.android.com/guide/components/activities/index.html.

[3] Fragments, 2017. https://developer.android.com/guide/components/fragments.html.

[4] Ville-Veikko Helppi. Android beyond the handset, 2010. http://www.techdesignforums.com/practice/technique/android-beyond-the-handset.

[5] Android Architecture, 2017. https://www.tutorialspoint.com/android/android_architecture.htm.

[6] Android Interfaces and Architecture, 2017. https://source.android.com/devices/.

[7] ART vs Dalvik, 2013. https://infinum.co/the-capsized-eight/art-vs-dalvik-introducing-the-new-android-runtime-in-kit-kat.

[8] App Manifest, 2017. https://developer.android.com/guide/topics/manifest/manifest-intro.html.

[9] Intent, 2017. https://developer.android.com/reference/android/content/Intent.html.

[10] Common Intents, 2017. https://developer.android.com/guide/components/intents-common.htm.

[11] Services, 2017. https://developer.android.com/guide/components/services.html.

[12] Storage Option, 2017. https://developer.android.com/guide/topics/data/data-storage.html.

[13] Class Diagram, 2017. https://en.wikipedia.org/wiki/Class_diagram.