
# My Wellbeing Kit

## Design Report

Sam Fahey Project Manager
Aidan Vos Client Liaison
Changlai Zhao Lead Artist/Interfacer
William Stephenson Lead Programmer
WenHao Wang Lead Programmer
Weibo Chen Lead Designer
Sa Ma Lead Designer

**Introduction** **4**

**Data Storage & Management** **5**
SQLite Database 6
Database ER Model 6
Detailed Database Structure 6

**XML Version, Schema and Namespace** **8**

**Interface Design** **10**
Stage 1 10
Stage 2 10
Stage 3 11
Stage 4 11

**Functional Prototypes** **15**
Card Flipping and Scrolling 15
Approach A 15
Approach B 16
Approach C 16
Selected Approach 17
Data Structure for Dictionary 17
Approach A 17
Approach B 18
Approach C 18
Selected Approach 19
Data Storage 19
Approach A 19
Approach B 20
Approach C 20
Selected Approach 20

**Appendix I: Use Case Scenarios** **21**
Use Case 2: UC2_User_Selects_CardsMenuOption 21
Use Case 4: UC4_User_Flips_Card 23
Use Case 8: UC8_User_Writes_CardDescription 24
Use Case 9: UC9_User_Duplicates_Card 25
Use Case 11: UC11_User_Favourites_Card 26
Use Case 15: UC15_User_Creates_CustomCard 27
Use Case 27: UC27_User_Edits_CustomCard 29

Use Case 31: UC31_User_Creates_JournalEntry 31
Use Case 38: UC38_User_Edits_JournalEntry 33
Use Case 43: UC43_User_Selects_Keyword 34
Use Case 46: UC46_User_Sorts_JournalEntries 35
Use Case 55: UC55_User_Selects_EmotionalBank 36
Use Case 59: UC59_User_Enables_DescriptorPrompt 38
Use Case 62: UC62_User_Views_HelpfulResources 40
Use Case 65: UC65_User_Adds_HelpfulResource 41
Use Case 70: UC70_User_Edits_HelpfulResource 42

**Appendix II: OO Modelling 44**
Package Diagram 44
Class Diagrams 45
Sequence Diagrams 47

**Appendix III: Interface Design and Iterations 55**
Prototype 1 - Version 1 55
Prototype 2 - Version 1 57
Prototype 3 - Version 1 59
Prototype 4 - Version 1 (also see appendix 8.8) 61
Prototype 5 - Version 1 (also see appendix 8.9) 65
Prototype 6 - Version 1 73
Prototype 7 - Version 1 75
Prototype 5 - Version 2 (also see appendix 8.5) 77

**Appendix IV: Database Design 80**
Attribute description Table 80
Database Tables - Local SQLite Database 80
Database Overview 80
Card_Data table: 81
Journal_Data table: 82
Feeling_Words table: 83
Journal_Pass table: 83
Resource_Data table: 84

**Appendix V: Example XML Files 85**
Screen: viewCards 85
Screen: viewJournal 87
Screen: viewResources (edited) 89
Screen: Menu Sidebar 92

**Appendix VI: Functional Prototypes 94**
Card Animations 94

Flipping 94
Scrolling 95
Data Structure for Dictionary 97
Data Storage (user data) 100

**Appendix VII: Glossary 103**

**Appendix VIII: References 104**

# 1. Introduction

The purpose of this document is to describe the implementation of the My Wellbeing Kit mobile application as specified in the Software Requirements. This document provides guidelines for reference by the My Wellbeing Kit project development team during implementation of the software.

Section 2 of the document, Data Storage & Management, is divided into two sub sections, 2.1 SQL Database, and 2.2 XML.

Section 2.1 provides examples of the database ER model used by the application to store persistent data, with short descriptions for each table and their corresponding columns.

Section 2.2 provides an outline of the XML schema to be used by the application and its purpose, with example XML code for guidance.

Section 3 of the document, Interface Design, describes the process undertaken by the development team in designing the user interface for the application, providing examples with reasoning for each aspect core to the interface.

Section 4 of the document, Functional Prototypes, focuses on the possible methods for solving some of the key software implementation challenges that the project presents. Each functional prototype specifies several approaches that can be taken and specifies the preferred solution with reasoning.

This document contains several appendices relating to each section in the body of the document for reference by the project development team. The purpose of the appendices is to provide greater clarification on design related topics and iterations, including use case scenarios, object oriented modelling, interface design and iterations, database design, XML schema and files, functional prototypes, a glossary and references.

This document is a susceptible to change as project development progresses.

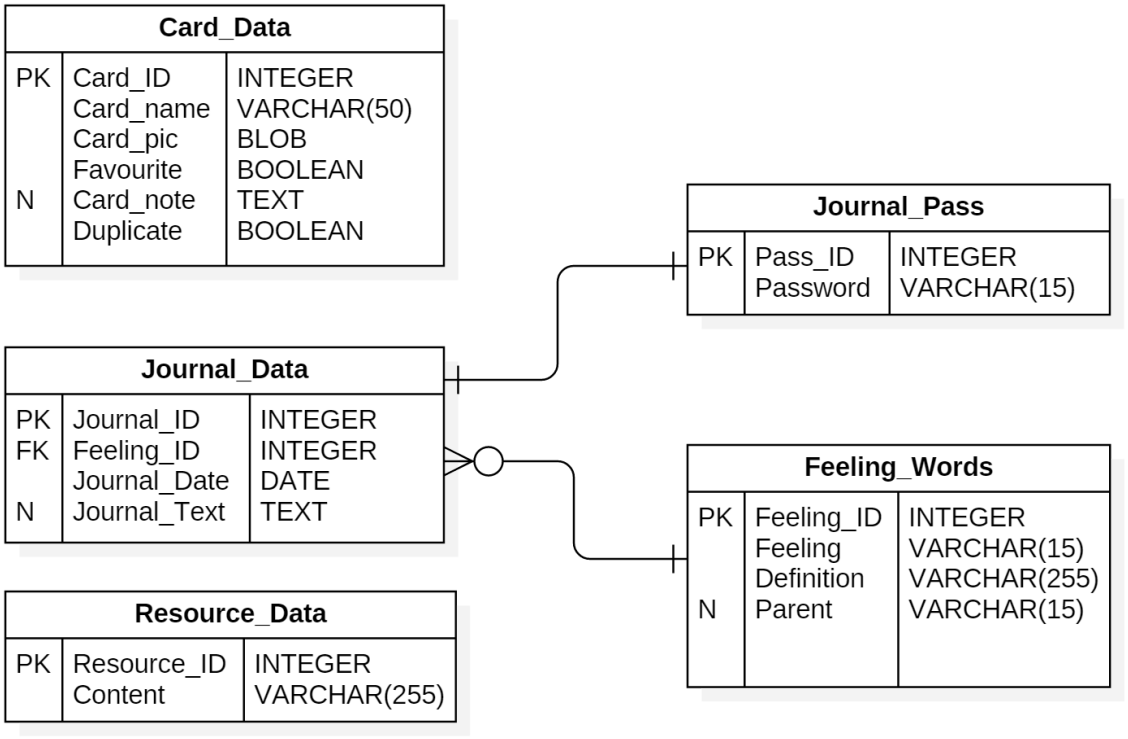
# 2. Data Storage & Management

The My Wellbeing Kit Android application will utilise SQLite for the storage and management of persistent user data. As a subset of SQL, SQLite inherits the query capability which enables fast and direct access to data within the applications database. This is required when the application would like to pull or push specific user entered data. The My Wellbeing Kit application features many instances where this is necessary.

The applications database will store both static and dynamic data. Static data is that which the user of the application will only be able to interact with but not change. Dynamic data is specific user entered information that is subject to change and can grow and shrink accordingly. SQLite facilitates this and does not require writing to external files for storage of data. Additionally the application must store encrypted data to enable the use of a password lock on the Journal system, this can be done through SQLite.

## 2.1. SQLite Database

### 2.1.1. Database ER Model




### 2.1.2. Detailed Database Structure

| Table | Attributes |
|---|---|
| Card_Data | Card_ID[PK], Card_name, Card_pic, Favourite, Card_note[N], Duplicate |
| Journal_Data | Journal_ID [PK], Feeling_ID [FK], Journal_Date, Journal_Text[N] |
| Journal_Pass | Pass_ID[PK], Password |
| Feeling_Words | Feeling_ID [PK], Feeling, Definition, Parent[N] |
| Resource_Data | Resource_ID, Content |

According to the needs of this project, all needed data will be classified according to the principle of high cohesion and low coupling in order to obtain higher efficiency while reducing the interference so this database is designed to contain five tables respectively:

- **Card_Data table:** Stores card data
- **Journal_Data table:** Stores journal entry data
- **Feeling_Words table:** Stores dictionary of emotions
- **Journal_Pass table:** Stores journal password (if enabled)
- **Resource_Data table:** Stores user entered helpful resources

See Appendix IV: Database Design for detailed information on specific tables.

# 3. XML Version, Schema and Namespace

The use of XML is mandatory when creating graphical user interfaces for Android applications, as such, the My Wellbeing Kit application will utilise XML to support the underlying structure of the user interface. Elements within the XML for each view allow for the communication and transfer of user entered data between view and controller classes to then be stored as persistent data within the SQLite database.

The My Wellbeing Kit application will use XML version 1.0 with UTF-8 encoding as this is the default for Android activities and suits the application's needs. This is to be declared at the top of all XML documents as follows:

```
<?xml version="1.0" encoding="utf-8"?>
```

A set of predefined Android XML schemas will be used by the application to facilitate the use of Android Studio's inbuilt GUI(see Glossary) design tool. In order to correctly use the predefined elements from the Android schema, each layout based XML document will include the following namespace declarations:

```
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
```

The following is an example of how each namespace can be used to access elements from the Android schema:

```
android:layout_width="match_parent"

tools:context="com.example.faheys.myapplication.MainActivity">

app:layout_constraintBottom_toBottomOf="parent"
```

Every application project must have an AndroidManifest.xml file (with precisely that name) at the root of the project source set. The manifest file describes essential information about the application to the Android build tools, the Android operating system, and Google Play.

The My Wellbeing Project Android application AndroidManifest.xml file will follow the example below:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">


        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>


    </application>
</manifest>
```

For each activity within the application there should be a new <activity></activity> section added between the <application></application> section parameters.

Example XML layout documents to be used in the application can be found in Appendix V: Example XML Files.

# 4. Interface Design

## 4.1. Stage 1

The process undertaken during the development of the My Wellbeing Kit application interface progressed through 4 stages. The first stage of development was designed in such a way that each individual member of the team provided their own ideas and vision for the project. This was structured so that each low/medium-fidelity prototype was developed independently from one another. Each member of the project team was asked to construct a prototype for what they envisioned the application would look like. All submissions were free to explore any artistic expression as long as they incorporated at the very least the 3 core screens for the application, the Wellbeing Cards, Journaling System and the Helpful Resources page.

Each submission was opened to review and critiqued by all other members of the team. In this way design elements that were liked were identified and encouraged. Conversely, those elements that would be difficult to incorporate, did not meet the criteria laid out in the Requirements Document, were not intuitive or simply were not aesthetically pleasing were rejected.

(See appendix 8.1. - 8.7. Interface Design and Iterations)

## 4.2. Stage 2

After the team reviewed the prototypes, the second stage of Interface Design began. This second stage of development initially began with extracting elements from the prototypes that were seen as the best in terms of being intuitive, fulfilling the criteria of the Requirements Document and having a pleasing design. The developers of these prototypes were then asked to perform a second stage of Interface Design. This involved incorporating the advice given throughout the review and readying their designs for another full team review. This process of team based review and critiquing was implemented before any showcasing of the prototypes to the client began. (See appendix 8.8. - 8.9. Interface Design and Iterations)

## 4.3. Stage 3

After the first two stages of Interface Design development the team was ready to get the client's ideas and feedback. Upon meeting with the client the team discussed the advantages of the current design and what they liked best about it from a coding and implementation standpoint. The client upon reviewing the two submitted prototypes provided the three developers and the team with their useful feedback that included: preference of fonts and colour scheme, a strict simplistic design choice, unique displays for the backing of each wellbeing card and more intuitive button design and layout. With these changes in mind stage 4 of the Interface Design development began.

## 4.4. Stage 4

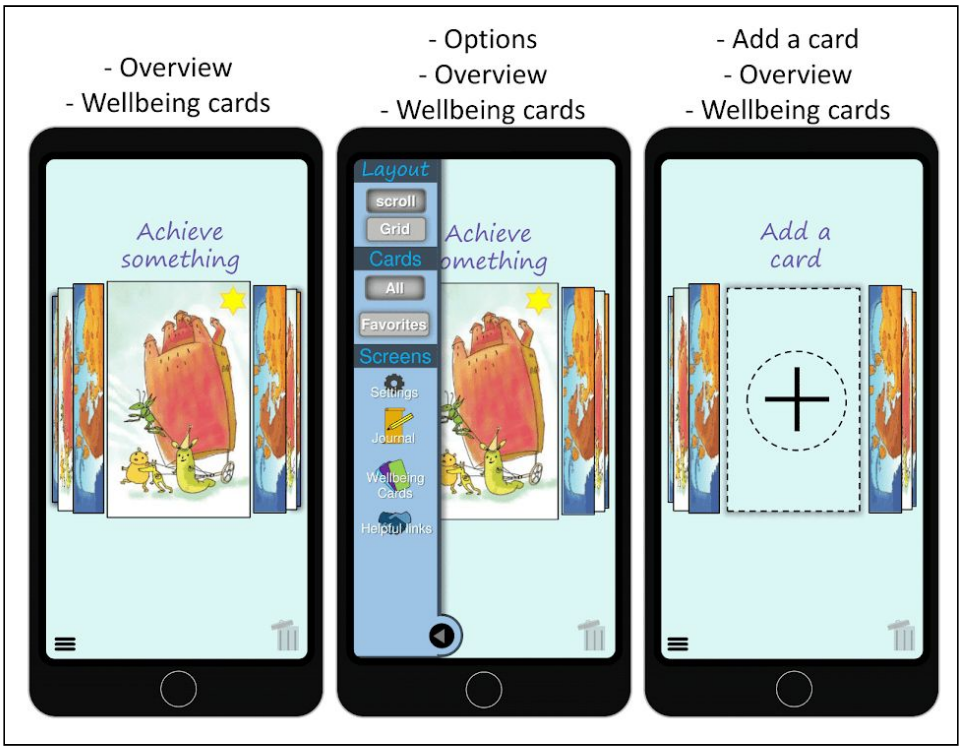
Stage 4 of development sought to unify the team and client's prototype vision. At this stage during the process a single member of the team was set upon the task of continual development. This meant proceeding through the first and second stages of development again, where the team would continually critique the prototype until it was deemed at a high enough standard of development to once more show to the client.

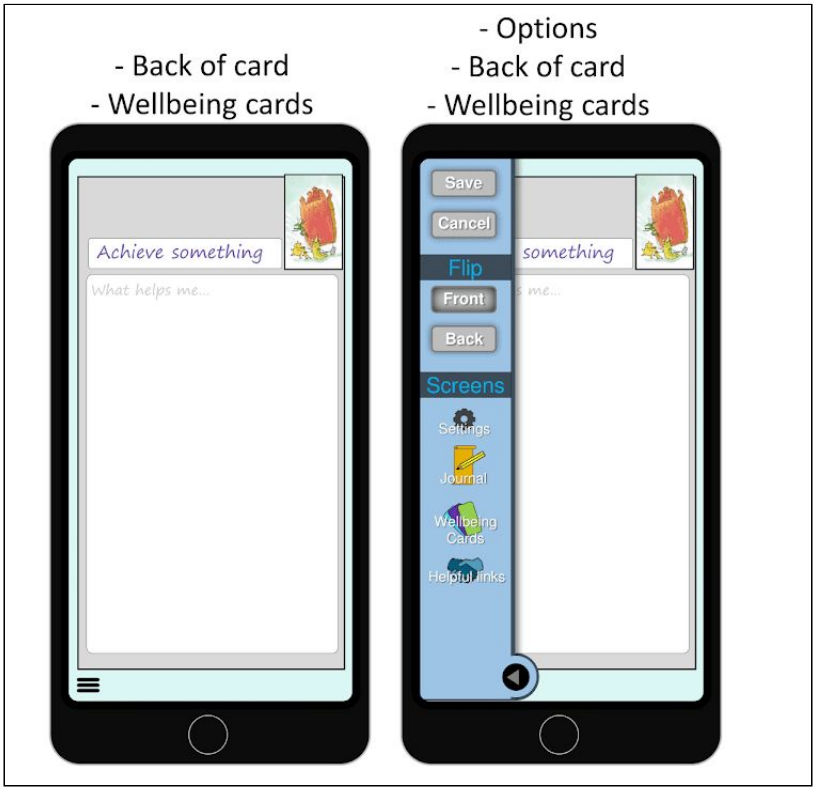
The client was then emailed with the newest iteration of interface design and asked for their thoughts and feedback on the design choices.

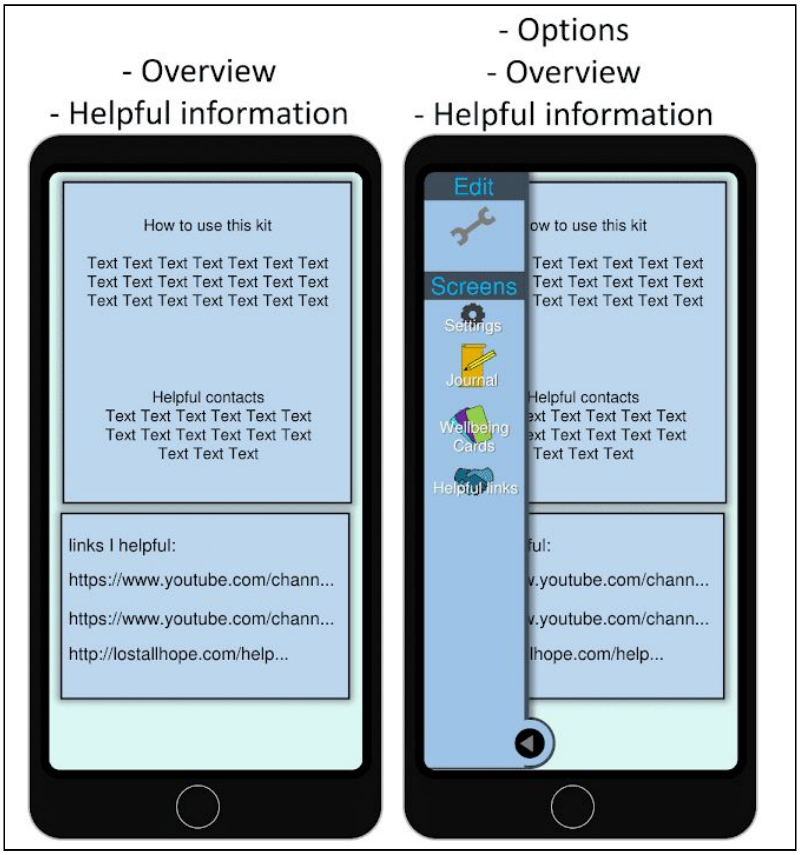
Once the client had made their ideas known the process would begin again.

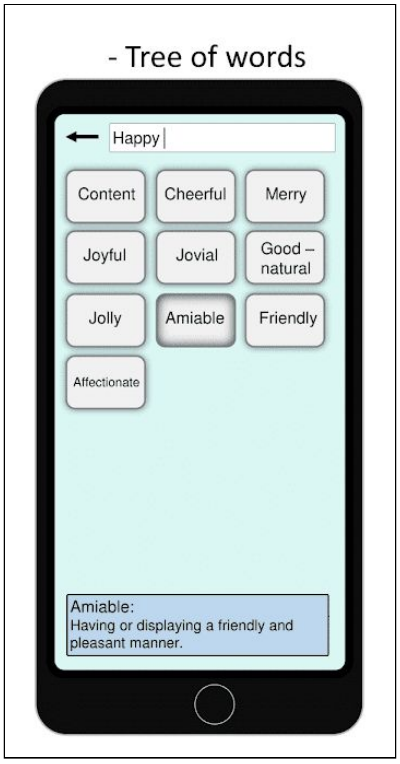
Over the life of the Design Report 11 different designs were mocked up and 4 of them showcased to the client.

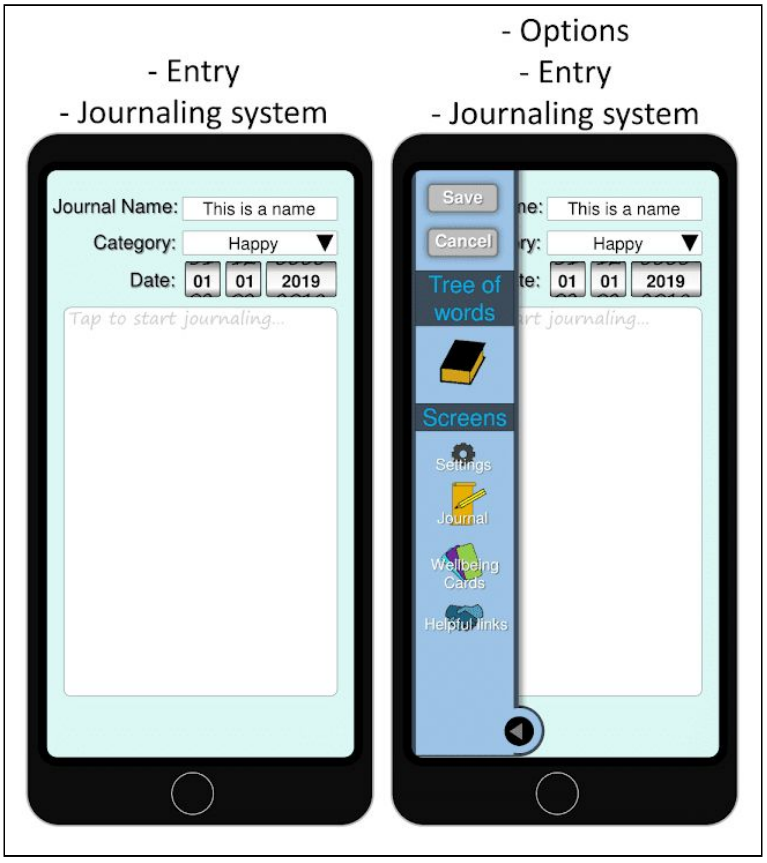
The following page includes several design prototypes from stage 4, subject to change throughout the development of the project.

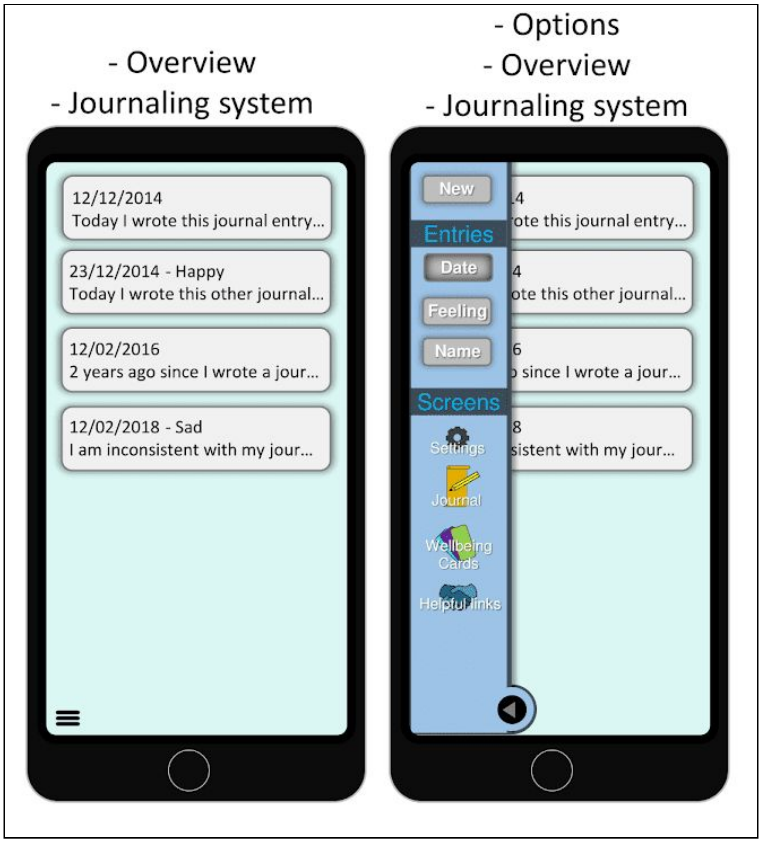

- Overview
- Wellbeing cards
- Options
- Overview
- Wellbeing cards
- Add a card
- Overview
- Wellbeing cards
Achieve something
Layout
scroll
Grid
Cards
All
Favorites
Screens
Settings
Journal
Wellbeing Cards
Helpful links
Add a card


- Back of card
- Wellbeing cards
- Options
- Back of card
- Wellbeing cards
Achieve something
What helps me...
Save
Cancel
Flip
Front
Back
Screens
Settings
Journal
Wellbeing Cards
Helpful links


- Overview
- Helpful information
How to use this kit
Text Text Text Text Text Text Text
Text Text Text Text Text Text Text
Text Text Text Text Text Text Text
Helpful contacts
Text Text Text Text Text Text
Text Text Text Text Text Text
Text Text Text
links I helpful:
https://www.youtube.com/chann...
https://www.youtube.com/chann...
http://lostallhope.com/help...
- Options
- Overview
- Helpful information
Edit
Screens
Settings
Journal
Wellbeing Cards
Helpful links


- Tree of words
Happy
Content
Cheerful
Merry
Joyful
Jovial
Good – natural
Jolly
Amiable
Friendly
Affectionate
Amiable:
Having or displaying a friendly and pleasant manner.


- Entry
- Journaling system
- Options
- Entry
- Journaling system
Journal Name:
This is a name
Category:
Happy
Date:
01
01
2019
Tap to start journaling...
Save
Cancel
Tree of words
Screens
Settings
Journal
Wellbeing Cards
Helpful links


- Overview
- Journaling system
- Options
- Overview
- Journaling system
12/12/2014
Today I wrote this journal entry...
23/12/2014 - Happy
Today I wrote this other journal...
12/02/2016
2 years ago since I wrote a jour...
12/02/2018 - Sad
I am inconsistent with my jour...
New
Entries
Date
Feeling
Name
Screens
Settings
Journal
Wellbeing Cards
Helpful links

# 5. Functional Prototypes

This project requires the creation of three functional prototypes. These were chosen with the purpose of allowing our team to investigate areas of the application which we believed we did not currently have sufficient knowledge to create without undertaking additional research. For each of the prototypes identified, the team members individually researched three different approaches to handling the issue and then further refined these to the to the list of three options presented below.

## 5.1. Card Flipping and Scrolling

This functional prototype is a solution to animating the movement of Wellbeing Cards in the application. It was chosen to address our team's lack of knowledge in the area of animation within Android applications. It is important for the design and feel of our application that users should be able to 'flick' through the cards in the same manner that they would when using the physical kit and as such we wanted to research a solution to closely replicate this in a digital format. Additionally, we wanted the users to be able to flip cards over so that they can view the back of them with the same intuitive ease.

### 5.1.1. Approach A

Android has a robust library for creating animations which can be included in any project through the android.animation dependency (Android Developers, 2018). With a combination of many of the provided functions within this library it would be possible to achieve the design goals for our application. This approach is beneficial in that it does not require using anyone else's pre-made solutions, meaning it would be tailored exactly to what we had in mind, rather than adapting to an existing solution. It would also be a valuable and challenging learning experience for the team as programmers. However, this also means that it could be very time consuming for little reward in terms of the overall project, when a premade solution (as discussed in 4.1.2, Approach B) would achieve a similar result with significantly less work.

### 5.1.2. Approach B

There is a very large number of free to use open source libraries for Android developers, covering all areas of application development. These packages includes animation libraries which build on the existing Android animation libraries (discussed in 4.4.1, Approach A) to create pre-made solutions for programmers to include in their applications. Android allows for these to be easily integrated into projects, simply by adding the URL of a package into the build.gradle file. After this has been done all the functions of the package are immediately available for use. Several existing packages found on GitHub very closely match the designs we had in mind for the application for both scrolling through views (or cards in our case) and also flipping between two views (viewing the back of the card). This approach is advantageous due to its ease of use and well tested animation code. The only downside is that the team has to make our design fit the mold defined by another programmer. This is alleviated by the open source nature of the code however, as the option exists to modify the libraries as necessary (depending on the terms of the package selected).

### 5.1.3. Approach C

Animations do not necessarily need to be built in code to be responsive to user input but rather can be pre-rendered out of 3D modelling software for instance and then simply activated by user touch. This approach could allow for a very similar looking application to what we have designed but would have the drawback of animations being static, meaning that they are activated by an input and would then play out to completion each time. It would be preferable to have dynamic animations which are guided by the user, meaning that views can be dragged around with a user's touch.

This approach provides the benefits of being straightforward to implement and the animations themselves can be refined to a level of great detail which will look exactly as intended every time. This may not be worth the distinctly static feel of the application that this would create however, as one of the primary design goals for the project is to make it feel as close as possible to using the physical kit.

### 5.1.4. Selected Approach

Approach B (4.1.2), using external open source packages, is the most suitable for our project. In particular, wajahatkarim3's EasyFlipView package was chosen to handle flipping the Wellbeing cards over (wajahatkarim3, 2017) and Devlight's InfiniteCycleViewPager package was chosen for scrolling through the list of Wellbeing Cards (Devlight, 2017). These two packages combined allow for us to recreate very closely the design we have already created for the Wellbeing Kit application. They can also be relied on to be thoroughly tested by the community of users and as such any bugs that do exist have been documented and can be worked around. This approach is significantly less time consuming than Approach A (4.1.1), using just the default Android animation libraries, which may have been more suitable to a team specialised in animation or for an application that needed very specific, non standard animations. Similarly, approach C, creating static animations, has its place in other projects but was not suitable to an application like this which needs to be as responsive as possible.
An extensive guide on implementing these features can be found in Appendix VI (10.1).

## 5.2. Data Structure for Dictionary

This functional prototype was needed to determine what type of data structure would be most appropriate for the implementation of the dictionary of emotional words. This dictionary needs to be able to be of a searchable hierarchical structure where at the top are the root emotional words (happy, sad, angry etc), and further down are more specific words (jealous, ecstatic, enraged). The specific data structure and how that would be implemented in Java were the focus of this functional prototype, as none of the development team have experience implementing data structures such as trees outside of the C language.

### 5.2.1. Approach A

A typical way to tackle a structure such as this would be with a node based non-binary tree system. In this there would be classes to create nodes and the tree structure which links all these nodes together and has functions to handle tree traversals and data fetching. Nodes are structures which contain a string for the title, pointers to any child nodes and a pointer to its

parent node. Having a pointer to the parent node makes tree traversal in all directions possible so that once a child node is found, other words from further up or across the tree can easily be navigated to. A downside to this is that tree searching algorithms are not especially quick, however this should not have a noticeable impact on a tree of this size.

### 5.2.2. Approach B

An alternative to node based tree system is a multidimensional array structure. This structure would use arrays nested with arrays to achieve a functionally similar result to the tree data structure. Operating in much the same way, at the top level would be an array of broad emotions (happy, sad, angry etc) which each contain their own arrays of more specific emotions and so on until the necessary number of layers have been reached. This approach is faster than the node based structure suggested in approach A, however due to the very small size of both, this benefit is not significant. The main downside to this approach is that none of the development team have as much experience with this as with the node based system which may present further unforeseen complications later in development.

### 5.2.3. Approach C

Instead of going with a Java based data structure, there is also the option of structuring hierarchical data through SQL tables. There are several approaches to doing this such as Nested Sets and Adjacency Lists. Any of these approaches require a high proficiency in SQL to manage the table, with queries to even perform basic queries or manipulation of data being potentially complicated (Worthy, 2014). This approach offers few benefits to our application and has a lengthy set of downsides including the complicated nature of retrieving data and the extra care needed when making changes to the way the structure is set out. The team all have a good understanding of SQL fundamentals but this approach is much more complicated than anything attempted by a team member before.

### 5.2.4. Selected Approach

Approach A (4.2.1) ended up being the most suitable for our team and project. The node based non-binary tree approach is a very straightforward way to handle this problem. There is extensive documentation available for creating trees of this type and although none of the team have done this in Java, it appears much simpler than C which all members have experience with. The other options did not provide significant enough benefits to make it worth straying away from the traditional method that the team is familiar with.

## 5.3. Data Storage

This functional prototype is a solution to storing data used by the application to generate entities such as Cards and Journal Entries. This prototype was chosen so that we could investigate and attempt to implement the most suitable form of data storage for the application. The heavily object oriented design of the application means that it is essential that the program should be able to read and write data to a persistent storage solution in a straightforward manner as it will be happening frequently.

### 5.3.1. Approach A

Android has a variety of options for persistent storage for applications between sessions. Using the internal storage on an Android device would provide sufficient space for the relatively small amount of data that this application will take up. Standard internal storage uses text files external to the application for holding data. The app can make as many of these files as needed and write whatever they need to them and read it back again later. This solution is very straightforward but may prove challenging when storing and reading data needed for the creation of entities like the ones in this application. An approach using CSV, could be effective but would be troublesome to manipulate. The simplicity of setting up this approach may not be worth the trouble of working with it.

### 5.3.2. Approach B

Another way for the application to locally store data is in XML files. XML is a markup language that performs a similar function to a text file with the additional benefit of marking up data make it much more practical for use as a form of database. XML tables can be manipulated and queried in a similar way to SQL tables. It is also very fast to use with small datasets due to the minimal overhead, although it does not scale as effectively as SQL alternatives. XML files are stored in plaintext in the Android filesystem, meaning it may prove to be a security issue for data such as journal entries which would need to be encrypted before being stored in XML.

### 5.3.3. Approach C

SQLite is a self contained, serverless implementation of SQL which has much of the power of full SQL implementations with very little overhead. It is the most widely used storage solution across all mobile devices, meaning there will be a lot of support and documentation accessible online. This approach provides an extensive amount of functionality which scales well with large datasets. A very useful part of the functionality is the extension SQLite Encryption Extension which can quickly encrypt whole databases, a feature which may be helpful when handing the encryption of journal entries in this application. Most members of the team have experience in using SQLite in a mobile application.

### 5.3.4. Selected Approach

Out of the options considered, Approach C, using SQLite for data storage has the best combination of functionality, performance and familiarity to team members. XML is also a viable option, with comparable performance and features for querying and manipulation. The benefits of SQLite such as database encryption and querying in a language that most group members are familiar with made it more suitable for us than an XML alternative.

# 6. Appendix I: Use Case Scenarios

## 6.1. Use Case 2: UC2_User_Selects_CardsMenuOption

**Use Case #:** UC2_User_Selects_CardsMenuOption

**Requirement**:

Users will gain access to their cards by selecting the 'My Wellbeing Kit' button located on the main menu.

**Overview**:

The main menu option to navigate to the Cards screen is the first option in the list of menu items. Selecting the 'My Cards' item will launch the viewCards activity whereby the user will be presented with their customised list of Wellbeing cards.The main menu can be accessed by sliding the tabular indicator on the left side of the screen.

**Preconditions**:

1. The application is running

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User slides Menu tab open. | 1. Display Main Menu over current screen.<br>2. Wait for onClickListener() to be called on a list item. |
| 2. User selects 'My Cards' menu option. | 1. Run onClickListener() for 'My Cards' list item.<br>2. Select all card data from Cards table in DB.<br>3. Display viewCards screen with appropriate data for each card. |

**Scenario Notes:**

- Action 1 must occur first, and can occur independently of action 2
- Action 2 must occur after action 1

**Post Conditions:**

- User is taken to viewCards screen, where they can see a list of their Wellbeing cards.

**Required Views (GUIs):**

- viewMenu
- viewCards

**Exceptions:**

1. The application cannot access local database. Application should quit with error code presented.

**Use Cases utilised:**

- None

## 6.2. Use Case 4: UC4_User_Flips_Card

**Use Case #:** UC4_User_Flips_Card

**Requirement**:

The user will also be offered the ability to flip their cards over rather than be limited to a simple swipe left or right viewing function.

**Overview**:

User can control the card not only swipe the card from both left and right side, but also can flip the card upon selection. Upon selection, the card will flip over with a small animation, revealing the back of the card where the user can enter a description.

**Preconditions**:

1. The user has selected 'My Cards' from the main menu, resulting in the viewCards screen being displayed.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User swipes the card from left of the screen to the right side | 1. Change list item being viewed to next cardID (currentCardID ++)<br>2. Load currentCardID image from DB<br>3. Refresh viewCards display with currentCardID data |
| 2. User swipe the card from right side of the screen to the left side | 1. Change list item being viewed to next cardID (currentCardID --)<br>2. Load currentCardID image from DB<br>3. Refresh viewCards display with currentCardID data |
| 3. User selects card | 1. Load data corresponding to currentCardID from Cards table in DB<br>2. Display viewCardBack screen |

**Scenario Notes:**

- All actions can occur independently of one another but not at the same time

**Post Conditions:**

- User is taken to editCard screen

**Required Views (GUIs):**

- viewCards
- viewCardBack

**Exceptions:**

1. Selected cards data is corrupt. Notify user and remove card from DB.

**Use Cases utilised:**

- None

## 6.3. Use Case 8: UC8_User_Writes_CardDescription

**Use Case #:** UC8_User_Writes_CardDescription

**Requirement**:

Upon flipping a card over the user will be presented with a space in which they have the opportunity to note down their ideas, emotions and thoughts in reference to that particular card.

**Overview**:

After selecting a card the user is can view the given cards back face where the card description is displayed. The description can be altered by selecting the editText field that is present.

**Preconditions**:

1. The user has selected a card from the viewCards screen (UC4_User_Flips_Card)

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects editText field | 1. Focus editText field<br>2. Load description data from Cards table in DB<br>3. Display on-screen keyboard |
| 2. User presses return(enter) key | 1. Store String value from editText field in Cards table in DB<br>2. Call notifyDataSetChanged()<br>3. Display viewCardBack screen |

**Scenario Notes:**

- Action 1 must occur first
- Action 2 must occur after action 1
- Action 1 can occur many times

**Post Conditions:**

- The user can change and view a cards description
- The user can view the back of a card

**Required Views (GUIs):**

- viewCardBack

**Exceptions:**

None

**Use Cases utilised:**

- UC4_User_Flips_Card

## 6.4. Use Case 9: UC9_User_Duplicates_Card

**Use Case #:** UC9_User_Duplicates_Card

**Requirement**:

The My Wellbeing Kit application will also offer the user the ability to duplicate cards.

**Overview**:

The option to duplicate cards will be available through the use of a button on the viewCards screen, the button will only appear after long pressing a card. Duplicated cards will be indicated by an icon located on the card itself.

**Preconditions:**

1. The user is on the viewCards screen

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User long presses a card | 1. Display dialog box asking to duplicate card<br>2. Wait onClickListener for positive and negative buttons of dialog box (yes/no) |
| 2. User selects yes dialog option. | 1. Pull all data corresponding to currentCardID from DB<br>2. Run createNewDuplicate() function<br>3. Display updated viewCards screen |
| 3. User selects no dialog option | 1. Close dialog box<br>2. Display viewCards screen |

**Scenario Notes:**

- Action 1 must occur first
- Actions 2 and 3 can only occur after action 1

**Post Conditions:**

- A duplicated card is created and can be viewed on the viewCards screen

**Required Views (GUIs):**

- viewCards

**Exceptions:**

None.

**Use Cases utilised:**

None.

## 6.5. Use Case 11: UC11_User_Favourites_Card

**Use Case #:** UC11_User_Favourites_Card

**Requirement**:

Users shall be able to flag cards as favourites, adding them to an additional list of 'favourite cards'.

**Overview**:

A toggleable button will be displayed when viewing cards which will allow for the flagging of the cards as a favourite and unflagging them when pressed again. Favourite cards will be presented first in the list of Wellbeing cards.

**Preconditions**:

1. The user is on the viewCards screen

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. The user selects Favourite toggle when toggle is set to 'false' | 1. getCurrentCardID()<br>2. Set isFavourite (bool) to true<br>3. Display updated viewCards |
| 4. The user selects Favourite toggle when toggle is set to 'true' | 1. getCurrentCardID()<br>2. Set isFavourite (bool) to false<br>3. Display updated viewCards |

**Scenario Notes:**

- Action 1 must occur first
- Action 2 can only occur after action 1

**Post Conditions:**

- User can indicate favourite cards and view them first in the list of cards on viewCards screen

**Required Views (GUIs):**

- viewCards

**Exceptions:**

- None

**Use Cases utilised:**

- None

## 6.6. Use Case 15: UC15_User_Creates_CustomCard

**Use Case #:** UC15_User_Creates_CustomCard

**Requirement**:

The user shall be able to add custom cards to their library.

**Overview**:

The user will select a menu option taking them to an edit text form used to create a new card in the application. This form will contain a 'Title' and 'Image' field. When selected from the phone's storage, the image can be adjusted with basic editing tools and will be scaled to the appropriate resolution. Once completed, the form will generate a new card based on the information provided and add it to the existing database of cards on the user's local storage.

**Preconditions**:

1. The user should have the application opened to the viewCards screen.

**Scenario**:

| Action | Software Reaction |
|---|---|
| 1. User selects createNew button | 1. Run onClickListener() for createNew button<br>2. Display createCard view<br>3. Create new row in Cards table<br>4. Set focus to lblTitle edit text field |
| 2. User adds title to new card | 1. IF lblTitle is not focussed, set focus to lblTitle<br>2. Display default on-screen keyboard<br>3. Store entered text into temp String value 'title' |
| 3. User selects addImage button | 1. Run onClickListener() for addImage button<br>2. Launch default image gallery application, wait for selection |
| 4. User crops image | 1. Load cropImage() with 128x128px parameters<br>2. Copy selected image into "Image" field of Cards table within DB.<br>3. Load selected image from DB into current createCard view context |
| 5. User selects 'Save' button | 1. Load data stored in temp String variable title to new card row in Cards table, within DB<br>2. Check 'Image' field is not null, if null load image into 'Image' field of |

| | |
|---|---|
| | Cards table in DB<br>3. Generate CardID and store in ID column of new card row<br>4. Display imageCreated dialog box<br>5. Call notifyDataSetChanged()<br>6. Display viewCards screen |
| 6. User selects ‘Cancel’ button | 1. Delete new card row from Cards table in DB<br>2. Display viewCards screen |

**Scenario Notes:**

- Actions 1 and 2 must occur in sequential order
- Actions 3, 4, 5 and 6 can only occur after action 1
- Action 4 can only occur after action 3
- Action 5 is only available after actions 2 and 3
- Action 6 can occur independently of actions 2, 3, 4, and 5

**Post Conditions:**

- User can view newly created custom card
- DB Table Cards has a new row with custom card information (if action 5 has occurred).

**Required Views (GUIs):**

- viewCards
- createCard
- cropImage

**Exceptions:**

1. Selected custom image is not of a valid file type. The application should inform the user that the card cannot be created and that another image must be selected. The application should then return to the custom card creation screen.
2. The user fails to enter a title and image. The application should not allow the user to select save until both fields have been filled in.

**Use Cases utilised:**

- None

## 6.7. Use Case 27: UC27_User_Edits_CustomCard

**Use Case #:** UC27_User_Edits_CustomCard

**Requirement**:

The user shall be able to edit cards that they have added to the application.

**Overview**:

Cards that have been added by the user can be edited. This will allow users to change the image and text associated with the card without needing to delete the old card and create a new one.

**Preconditions**:

1. The user must have previously created a custom card (UC15_User_Creates_CustomCard), otherwise there will be no editable cards.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects 'Edit' button with one card selected | 1. Load matching card data from Cards table in DB<br>2. Display editCustomCard view<br>3. Populate Title, Image, Description fields with data from DB |
| 2. User selects title field | 1. Focus title edit text field<br>2. Display on-screen keyboard<br>3. Upon 'return key' press store string from title field into temp variable |
| 3. User selects newImage button | 1. Load default image gallery application<br>2. Load selected image into cropImage() with 128x128px parameters<br>3. Copy selected image into "Image" field of Cards table within DB.<br>4. Load selected image from DB into current editCustomCard view context<br>5. Display editCustomCard view |
| 4. User selects 'Save' button | 1. Store temp title variable in card row of Cards table<br>2. Call notifyDataSetChanged()<br>3. Display viewCards screen |
| 5. User selects 'Discard' button | 1. Display discardChanges dialog box |
| 6. User selects 'Yes' dialog option | 1. Close discardChanges dialog box |

| | |
|---|---|
| | 2. Display viewCards screen |
| 7. User selects ‘No’ dialog option | 1. Close discardChanges dialog box<br>2. Display editCustomCard screen with previous context |

**Scenario Notes:**

- Action 1 must occur before all other actions
- Actions 2, 3, 4 and 5 can all occur independently of one another
- Actions 6 and 7 must occur after action 5

**Post Conditions:**

- User can view changes to custom card on viewCards screen
- DB stores new values to reflect changes to cards data fields

**Required Views (GUIs):**

- viewCards
- createCard
- editCustomCard

**Exceptions:**

1. Selected custom image is not of a valid file type. The application should inform the user that the card cannot be created and that another image must be selected. The application should then return to the editCustomCard view.
2. User does not have a default image gallery application available. Application should notify user.

**Use Cases utilised:**

- UC15_User_Creates_Card

## 6.8. Use Case 31: UC31_User_Creates_JournalEntry

**Use Case #:** UC31_User_Creates_JournalEntry

**Requirement:**

The user shall be able to create a new journal entry (see, Appendix 4.1) from the journal screen, accessible through a clearly visible button.

**Overview:**

This functionality allows the user to initiate a new journal entry from which they can detail their thoughts and feels, by selecting the 'create journal entry' button. Upon selection of the button the user will be prompted to fill out a journal entry in the blank form provided. The user will be able to save journal entries to the device for access at a later date.

**Preconditions:**

1. User has selected Journal from the main menu

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects the 'create journal entry' button | 1. Run onClickListener() for create journal entry button<br>2. Display editJournalEntry screen |
| 2. User selects the 'Keyword' drop down menu | 1. Load predetermined 'dictionary of emotions' into list<br>2. Await onClickListener() for list items |
| 3. User selects a 'Keyword' from dropdown menu | 1. UC43_User_Selects_Keyword |
| 4. User selects journalText editText field | 1. Focus journalText field<br>2. Display on-screen keyboard<br>3. Store journalText string data in temp String variable. |
| 5. User selects the 'Cancel' button | 1. Display dialog box 'are you sure?' |
| 6. User selects yes dialog option | 1. Close journal entry screen<br>2. Remove data changed in current context |
| 7. User selects no dialog option | 1. Close dialog box |

| | |
|---|---|
| 8. User selects the ‘Save’ button | 1. Run onClickListener() for save button<br>2. The session information entered is stored locally.<br>a. Get current date and store in date field of journalEntry table in DB<br>b. Store journalText in journalEntry table in DB<br>c. Store journal ‘Keyword’ in category field in journalEntry table in DB<br>3. Display pop-up “Journal #journal’s name# has been saved”.<br>4. Display updated viewJournal screen |

**Scenario Notes:**

- Action 1 must occur first
- Action 3 must occur after action 2
- Actions 4, 5 and 8 can occur independently of one another
- Actions 6 and 7 must occur after action 5

**Post Conditions:**

- User can create new journal entries and save them for later viewing

**Required Views (GUIs):**

- viewJournal
- editJournalEntry

**Exceptions:**

None.

**Use Cases utilised:**

- UC43_User_Selects_Keyword

## 6.9. Use Case 38: UC38_User_Edits_JournalEntry

**Use Case #:** UC38_User_Edits_JournalEntry

**Requirement:**

The user shall be able to make changes to already saved journal entries.

**Overview:**

This functionality allows the user to edit existing journal entries. This is made available to the user through the ‘edit journal entry’ button which appears when viewing a previous entry.

**Preconditions:**

1. User is viewing a journal entry

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects ‘edit journal’ button | 1. Load journal entry data from DB into local variables<br>2. Run editEntry() with variables passed as arguments<br>3. UC31_User_Creates_JournalEntry |

**Scenario Notes:**

- None

**Post Conditions:**

- User can edit a journal entry from the viewJournal screen

**Required Views (GUIs):**

- editJournalEntry
- viewJournal

**Exceptions:**

None

**Use Cases utilised:**

- UC31_User_Creates_JournalEntry
- UC43_User_Selects_Keyword
- UC43_App_Records_CreationDate

## 6.10. Use Case 43: UC43_User_Selects_Keyword

**Use Case #:** UC43_User_Selects_Keyword

**Requirement:**

Upon creating a journal entry the user shall be able to select a ‘keyword’(see, Appendix 4.1) from a predetermined ‘dictionary of emotions’(see, 3.1.5).

**Overview:**

The categorisation option will be available through a drop down menu located near the top of the new entry form. This option will prompt the user to select a ‘keyword’ that best describes how they are feeling.

**Preconditions:**

1. User is viewing the editJournalEntry screen

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. The user selects ‘keyword’ drop down menu | 1. Load predetermined ‘dictionary of emotions’ into list<br>2. Await onClickListener() for list items |
| 2. The user selects a ‘keyword’ from the drop down menu | 1. getCurrentListItem()<br>2. Store currentListItem in journalEntry field in DB under category |

**Scenario Notes:**

- Action 1 must occur first
- Action 2 must occur after action 1

**Post Conditions:**

- User can categorise their journal entries

**Required Views (GUIs):**

- editJournalEntry

**Exceptions:**

1. User does not select a category. Prompt user to select a category.

**Use Cases utilised:**

- UC31_User_Creates_JournalEntry

## 6.11. Use Case 46: UC46_User_Sorts_JournalEntries

**Use Case #:** UC46_User_Sorts_JournalEntries

**Requirement:**

The user shall be able sort saved journal entries by ‘keyword’, organised by date with most recent first.

**Overview:**

A button shall be available on the journal screen to select one of the predetermined keywords for filtering the entry list. Selecting the button will open a dialog box with a drop down menu to select a ‘keyword’ for filtering entries.

**Preconditions:**

1. User is viewing viewJournal screen

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects ‘filter’ button | 1. Load filter dialog box<br>2. Populate drop down list with base emotional ‘keywords’<br>3. Wait onClicklListener() for all list items |
| 2. User selects a keyword for filtering | 1. Run onClickListener() for currentItemID<br>2. Close dialog box<br>3. Display updated viewJournal screen |

**Scenario Notes:**

- Action 1 must occur fist
- Action 2 must occur after action 1

**Post Conditions:**

- User can sort journal entries on the viewJournal screen by their keyword

**Required Views (GUIs):**

- viewJournal

**Exceptions:**

1. There were no entries matching the keyword filter. Notify user that the list is empty.

**Use Cases utilised:**

- UC31_User_Creates_JournalEntry

## 6.12. Use Case 55: UC55_User_Selects_EmotionalBank

**Use Case #:** UC55_User_Selects_EmotionalBank

**Requirement**:

This is achieved by putting within the user's hand a powerful library of words with which they can easily navigate through and find descriptors for categorizing their feelings.

**Overview**:

The emotional bank can be accessed via a button on the editJournalEntry screen. The emotional bank will allow a user to select from a series of very general feelings, such as 'happy', 'sad', 'angry', etc. A word selected by the user will then suggest an assortment of synonyms that may be better suited to the user's emotional state.

**Preconditions**:

1. The user must be on the editJournalEntry screen, either creating a new entry (UC31_User_Creates_JournalEntry) or editing and existing one (UC38_User_Edits_JournalEntry).

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects Dictionary button | 1. Load Dictionary values from dictionary table on DB<br>2. Display viewDictionary screen |
| 2. User selects a base emotion from list | 1. Run onClickListener for currentListItem<br>2. Pull related words from DB<br>3. Display updated viewDictionary screen |
| 3. User selects a secondary emotion | 1. Run onClickListener for currentListItem<br>2. Pull dictiondaryDefinition of currentListItem<br>3. Display viewDictionary screen with definition added |
| 4. User selects 'Close' button | 1. Display editJournalEntry screen |

**Scenario Notes:**

- Action 1 must occur first
- Action 2 must occur after action 1
- Action 3 must occur after action 2
- Action 4 can occur independently after action 1

**Post Conditions:**

- User can access a dictionary of predetermined emotional words whilst creating or editing a journal entry

**Required Views (GUIs):**

- editJournalEntry
- viewDictionary

**Exceptions:**

None.

**Use Cases utilised:**

- UC31_User_Creates_JournalEntry
- UC38_User_Edits_Journal

## 6.13. Use Case 59: UC59_User_Enables_DescriptorPrompt

**Use Case #:** UC59_User_Enables_DescriptorPrompt

**Requirement:**

The second system, the 'Descriptor prompt' (see, Appendix 4.1), will simply be an expansion upon the idea of predictive text prompting used in modern day messaging and texting software will be available when users enable the 'Descriptor prompt' feature.

**Overview:**

This feature can be set up within the applications settings. Once enabled flagged words such as 'happy', 'sad', 'angry', etc. that have been entered into a journal entry will instead prompt the user with a suggested synonym giving them a greater vocabulary of words with which they can use to more accurately record thoughts and feelings.

**Preconditions:**

None.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects Descriptor prompt toggle in settings when toggled off | 1. Set bool isPromptOn = true;<br>2. Display updated viewSettings |
| 2. User selects Descriptor prompt toggle in settings when toggled on | 1. Set bool isPromptOn = false;<br>2. Display updated viewSettings |
| 3. User inputs a base keyword (ie. happy) into journal entry. | 1. Run strcmp() on journal entry description for keywords<br>2. Produce pop-up hint of suggested synonym for first word found. |

**Scenario Notes:**

- Action 1 can occur once before action 2 and only afterwards thereafter
- Action 2 must occur after action 1
- Action 3 can only occur after action 1

**Post Conditions:**

- The user is provided with suggested synonyms for common emotions

**Required Views (GUIs):**

- viewSettings
- editJournalEntry

**Exceptions:**

None.

**Use Cases utilised:**

- UC38_User_Edits_JournalEntry
- UC31_User_Creates_JournalEntry

## 6.14. Use Case 62: UC62_User_Views_HelpfulResources

**Use Case #:** UC62_User_Views_HelpfulResources

**Requirement:**

The users shall be able to view a list of helpful resources related to health and wellbeing.

**Overview:**

The user is provided with an expandable list of resources that will only be a button press away from the main menu for users who may not be sure what to search for to find such information online.

**Preconditions:**

None.

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User slides main menu open | 1. Display menu over current screen<br>2. Wait for onCLickListener() of a list item to be called |
| 1. User selects Helpful Resources from main menu | 1. Run onClickListener() for Helpful Resources list item<br>2. Display viewResources screen |

**Scenario Notes:**

- Action 1 must occur first
- Action 2 can only occur after action 1

**Post Conditions:**

- User can view a list of helpful resources

**Required Views (GUIs):**

- viewResources

**Exceptions:**

None.

**Use Cases utilised:**

None.

## 6.15. Use Case 65: UC65_User_Adds_HelpfulResource

**Use Case #:** UC65_User_Adds_HelpfulResource

**Requirement:**

Selecting the option will create a new blank entry at the bottom of the list, and prompt the user to enter text.

**Overview:**

The field will accept input from the on-screen keyboard. The new entry will automatically be saved after selecting the save button. The user will be able to see the new entry in the list after selecting the save button.Entries will be stored locally on the user's device for fast retrieval by the system when users access their helpful resources.

**Preconditions:**

1. The user is on the viewResources screen

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects 'add resource' button | 1. Open dialog box with editText field<br>2. Focus editText field<br>3. Display on-screen keyboard |
| 2. User selects Save button | 1. Create new row in Resources table in DB<br>2. Store editText String value in Resources table in DB<br>3. Close dialog box<br>4. Display updated viewResources screen |
| 3. User selects Cancel button | 1. Close Dialog box |

**Scenario Notes:**

- Action 1 must occur first
- Action 2 and 3 can only occur after action 1

**Post Conditions:**

- The user can add custom text to the viewResources screen

**Required Views (GUIs):**

- viewResources

**Exceptions:**

None.

**Use Cases utilised:**

- UC62_User_Views_HelpfulResources

## 6.16. Use Case 70: UC70_User_Edits_HelpfulResource

**Use Case #:** UC70_User_Edits_HelpfulResource

**Requirement:**

The user shall be able to edit entries in the list to keep them up to date, or remove old entries if they have become redundant or unnecessary.

**Overview:**

The application allows for entries in the Helpful Resources section to be edited using a button. Once pressed the screen will enter edit mode and allow the user to remove any custom entries on the list. By tapping on entries whilst in edit mode the user can edit custom entries which will be saved after exiting edit mode.

**Preconditions:**

1. User is on the viewResources screen

**Scenario:**

| Action | Software Reaction |
|---|---|
| 1. User selects edit button | 1. Display editResources screen<br>2. Await list item onClickListener() |
| 2. User ‘taps’(selects) custom entry | 1. Focus editText field of item<br>2. Display on-screen keyboard<br>3. Enable editing of selected string |
| 1. User presses return key | 1. Overwrite previous string in local variable<br>2. Display updated editResources screen |
| 3. User deletes custom resource entry | 1. Prompt user to confirm deletion with dialog box<br>a. Yes selected, delete entry from resources table in DB<br>b. No selected, close dialog box<br>2. Display updated editResources |
| 1. User selects Save button | 1. Display updated viewResources |

**Scenario Notes:**

- Action 1 must occur first
- Actions 2, 3 and 4 can occur independently of each other

**Post Conditions:**

- Users changes to helpful resources list are displayed on Helpful Resources screen.

**Required Views (GUIs):**

- viewResources
- editResources

**Exceptions:**

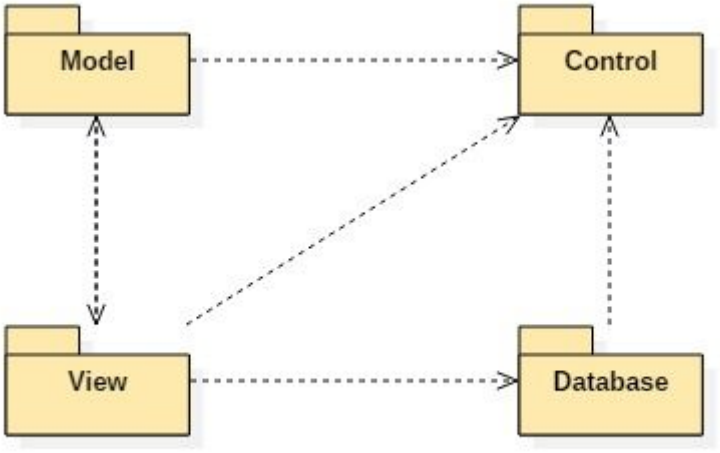
None.

**Use Cases utilised:**

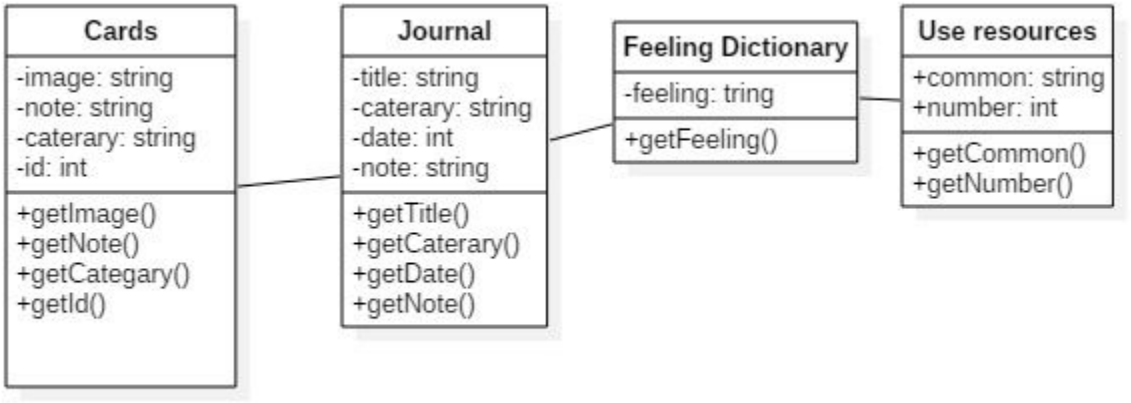
- UC62_User_Views_HelpfulResources

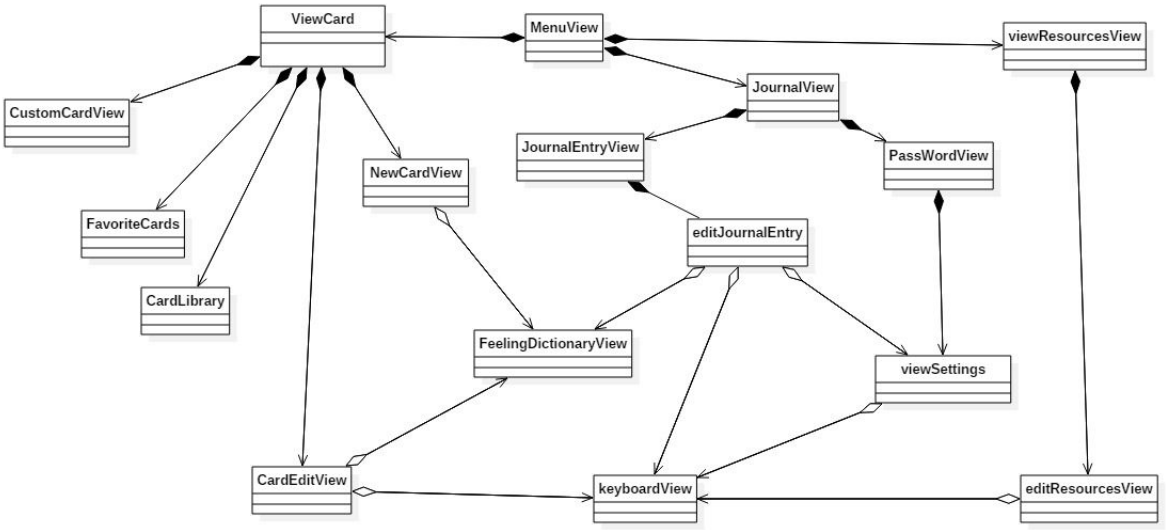
# 7. Appendix II: OO Modelling

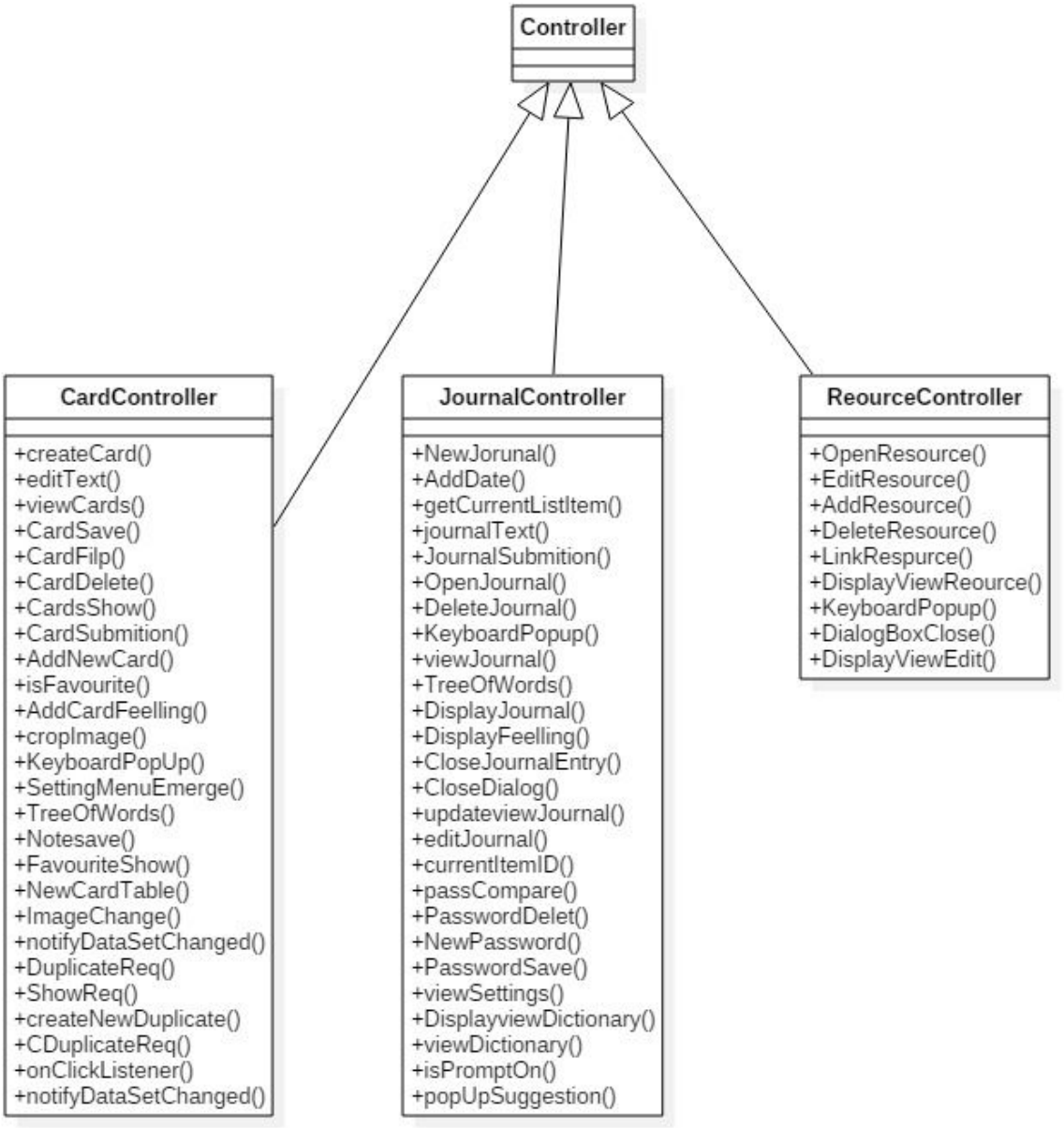
## 7.1. Package Diagram



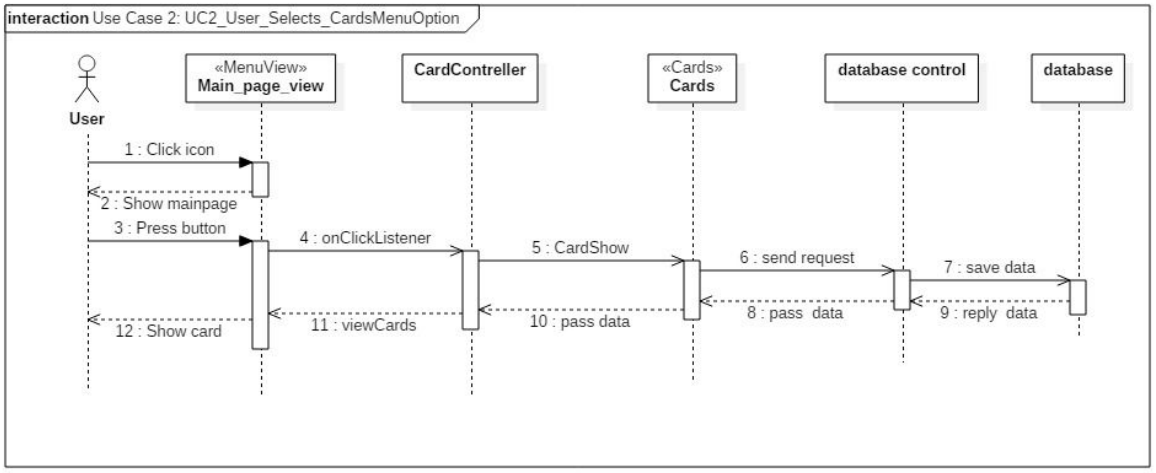
## 7.2. Class Diagrams

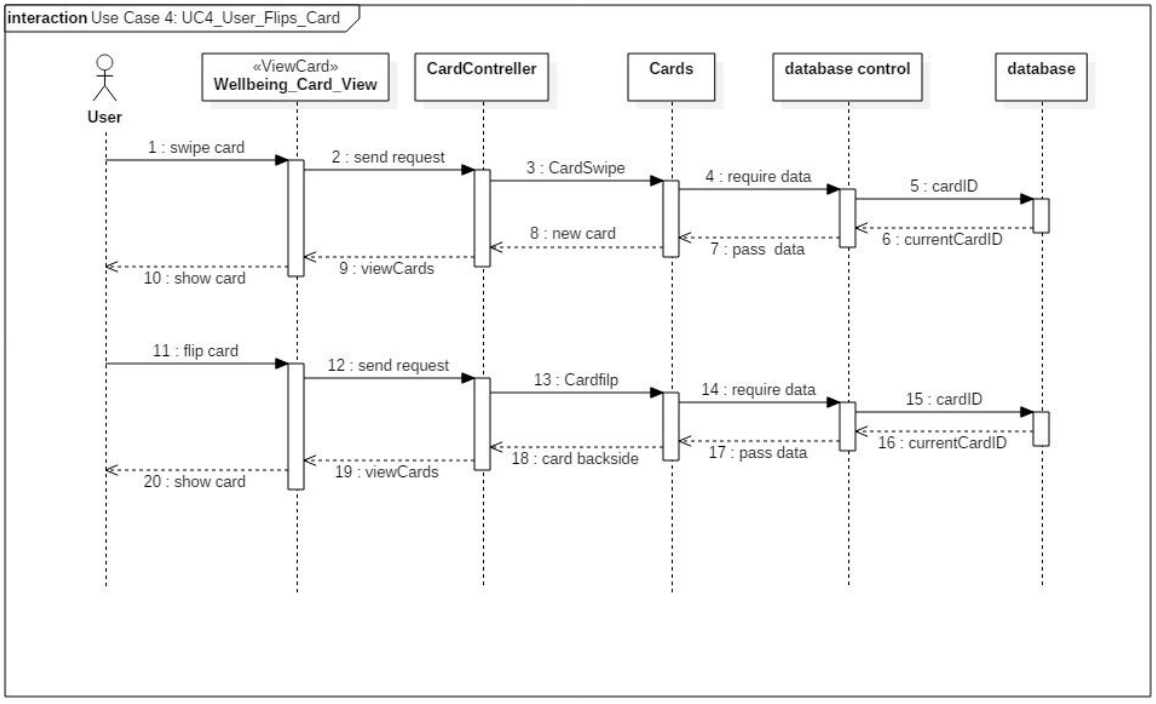


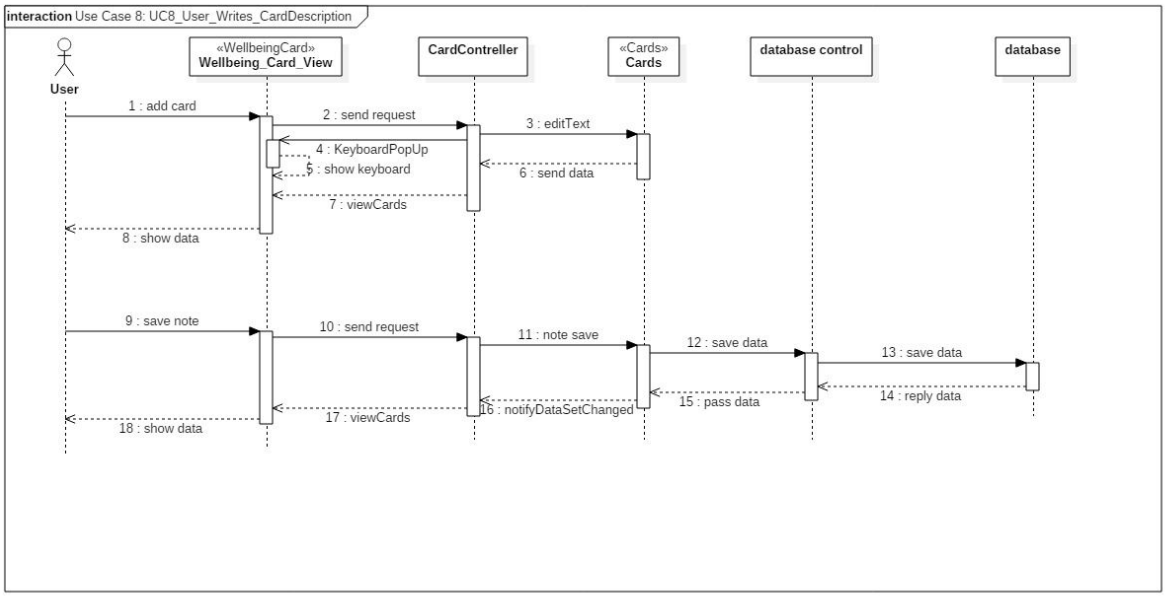



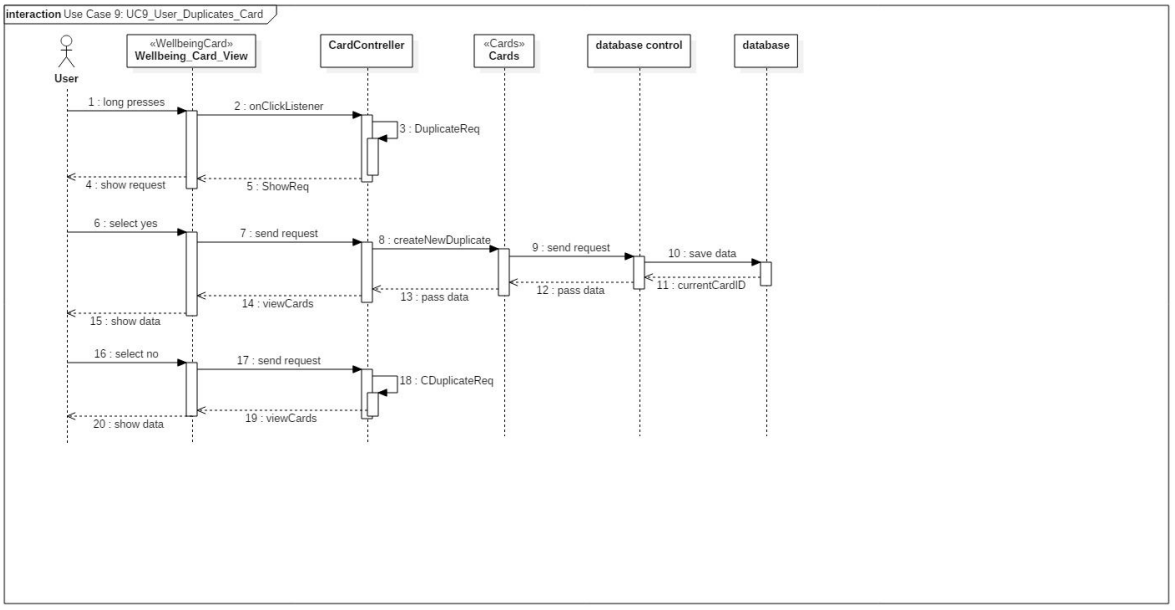

Controller
CardController
+createCard()
+editText()
+viewCards()
+CardSave()
+CardFilp()
+CardDelete()
+CardsShow()
+CardSubmition()
+AddNewCard()
+isFavourite()
+AddCardFeelling()
+cropImage()
+KeyboardPopUp()
+SettingMenuEmerge()
+TreeOfWords()
+Notesave()
+FavouriteShow()
+NewCardTable()
+ImageChange()
+notifyDataSetChanged()
+DuplicateReq()
+ShowReq()
+createNewDuplicate()
+CDuplicateReq()
+onClickListener()
+notifyDataSetChanged()
JournalController
+NewJorunal()
+AddDate()
+getCurrentListItem()
+journalText()
+JournalSubmition()
+OpenJournal()
+DeleteJournal()
+KeyboardPopup()
+viewJournal()
+TreeOfWords()
+DisplayJournal()
+DisplayFeelling()
+CloseJournalEntry()
+CloseDialog()
+updateviewJournal()
+editJournal()
+currentItemID()
+passCompare()
+PasswordDelet()
+NewPassword()
+PasswordSave()
+viewSettings()
+DisplayviewDictionary()
+viewDictionary()
+isPromptOn()
+popUpSuggestion()
ReourceController
+OpenResource()
+EditResource()
+AddResource()
+DeleteResource()
+LinkRespurce()
+DisplayViewReource()
+KeyboardPopup()
+DialogBoxClose()
+DisplayViewEdit()

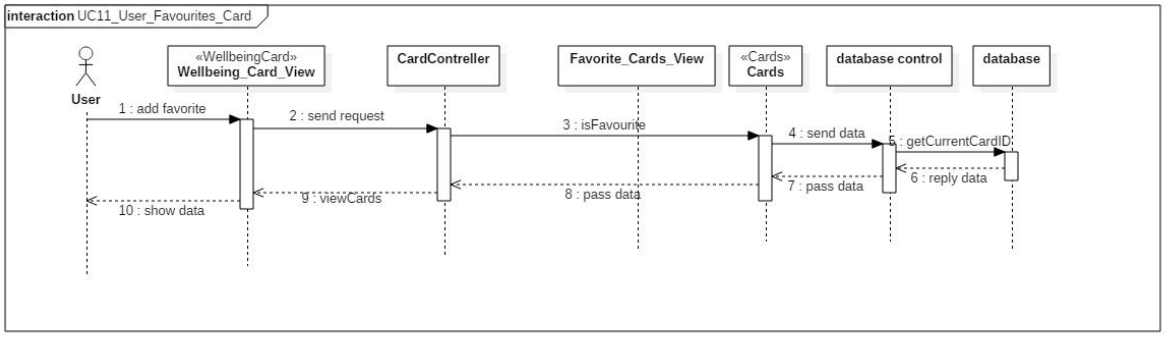
## 7.3. Sequence Diagrams

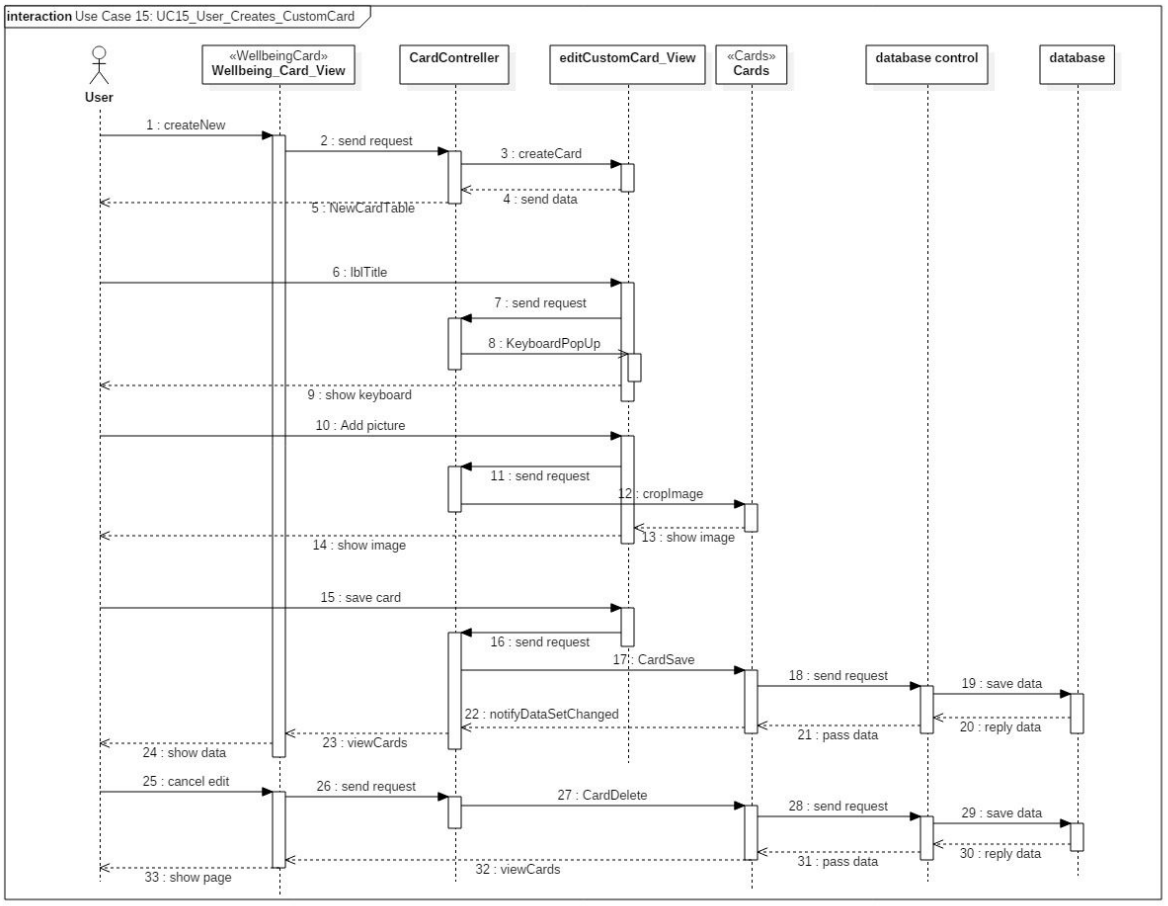


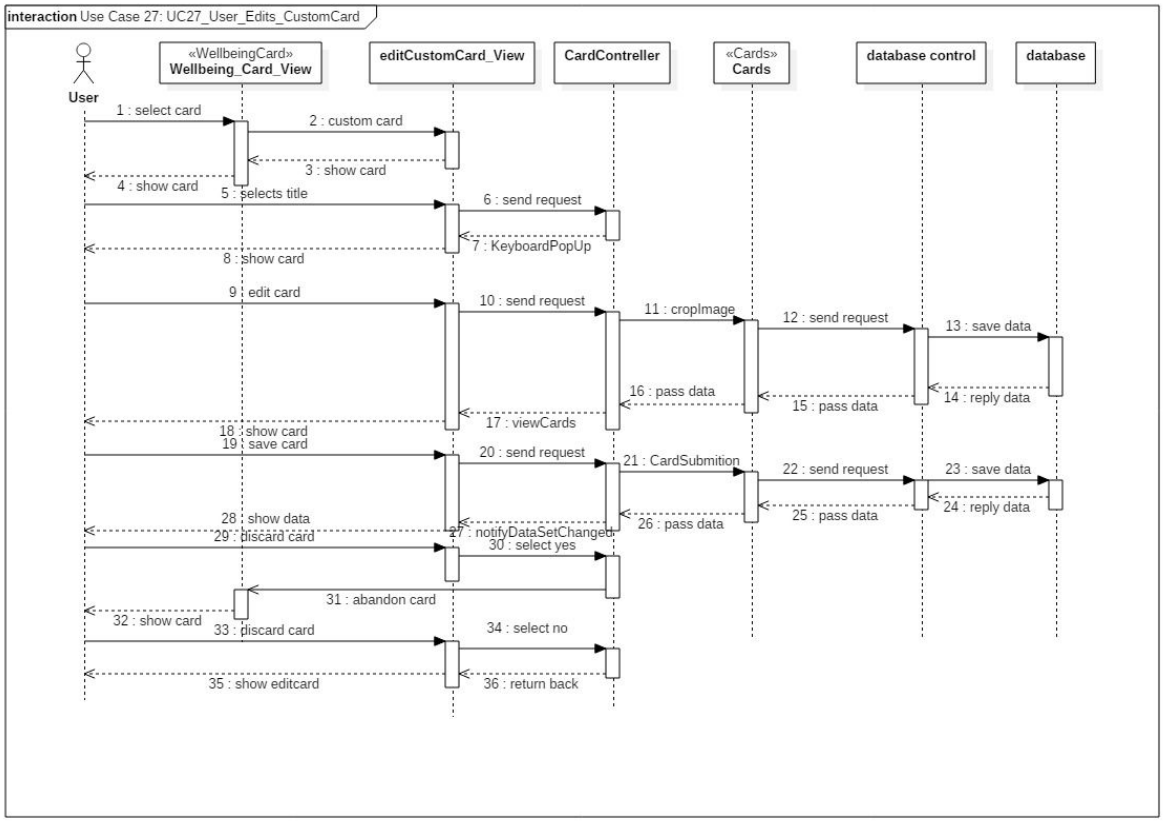



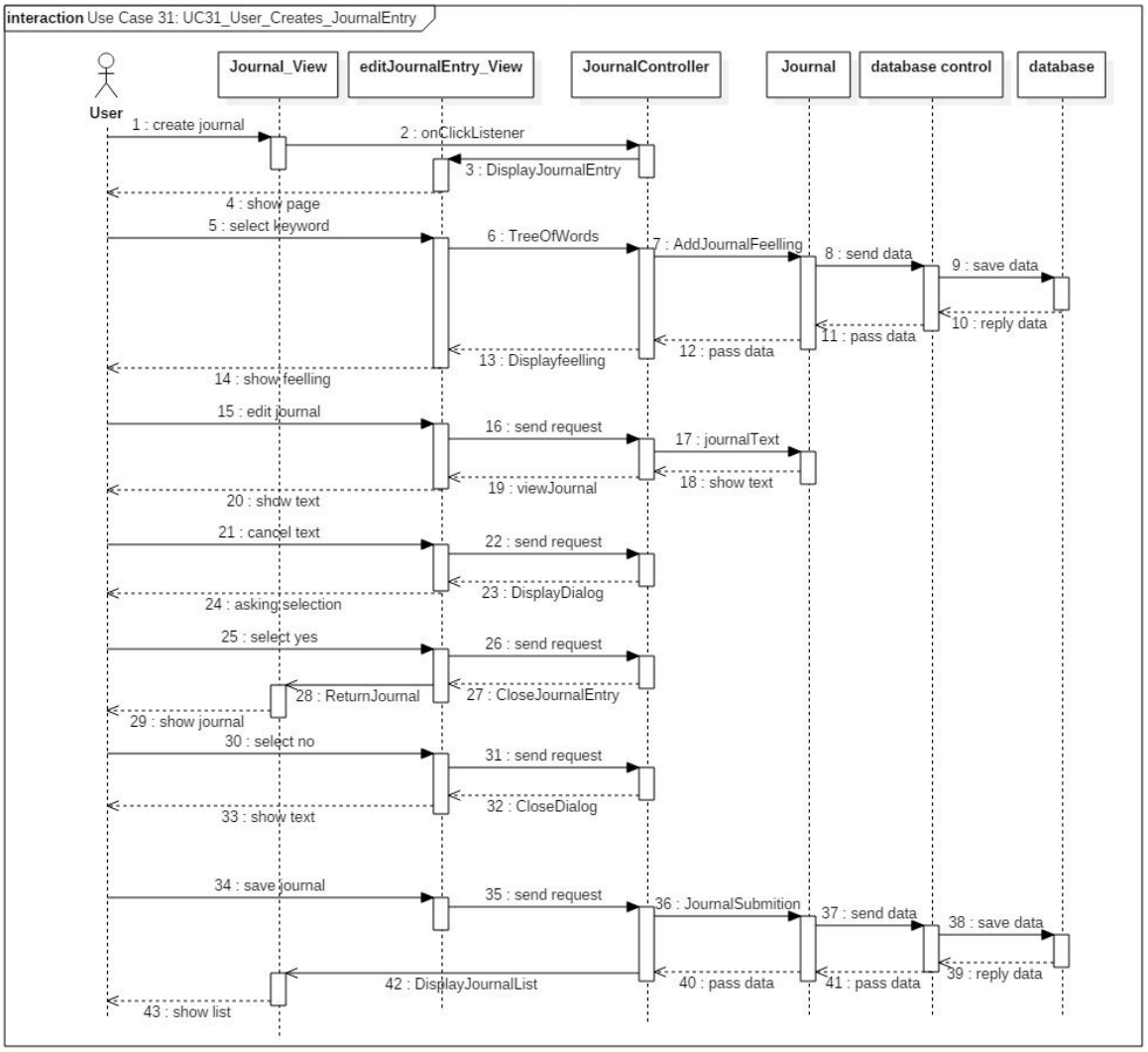

interaction Use Case 8: UC8_User_Writes_CardDescription
User
«WellbeingCard»
Wellbeing_Card_View
CardContreller
«Cards»
Cards
database control
database
1 : add card
2 : send request
3 : editText
4 : KeyboardPopUp
5 : show keyboard
6 : send data
7 : viewCards
8 : show data
9 : save note
10 : send request
11 : note save
12 : save data
13 : save data
14 : reply data
15 : pass data
16 : notifyDataSetChanged
17 : viewCards
18 : show data

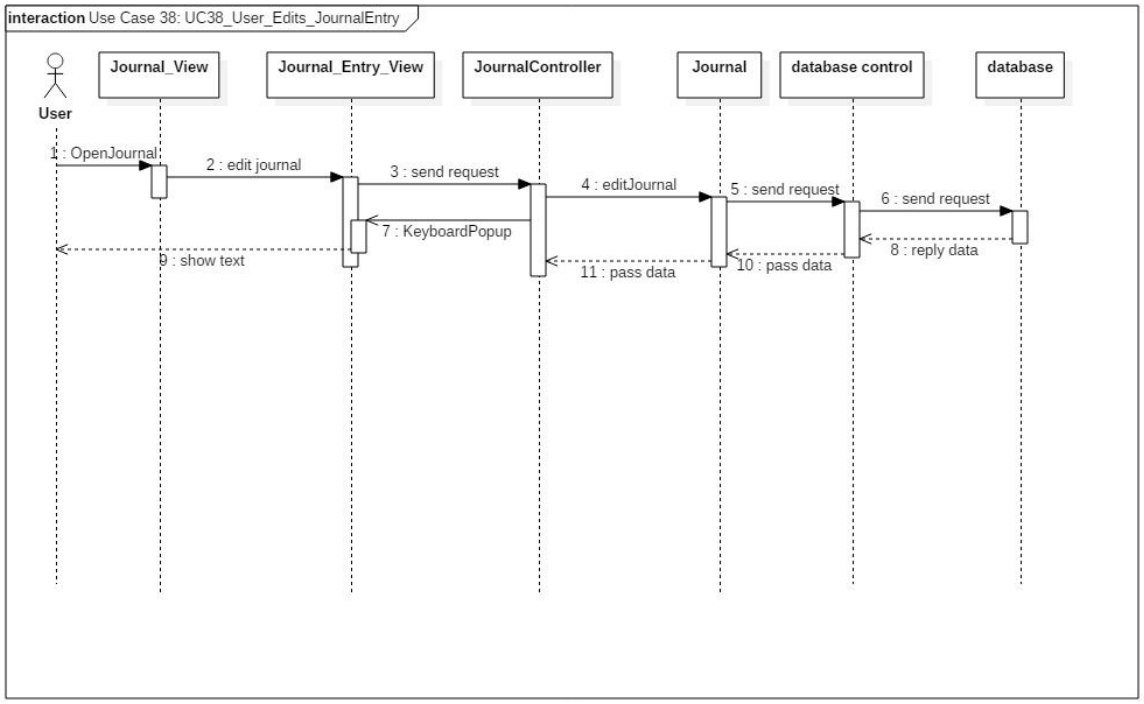

interaction Use Case 9: UC9_User_Duplicates_Card
User
«WellbeingCard»
Wellbeing_Card_View
CardContreller
«Cards»
Cards
database control
database
1 : long presses
2 : onClickListener
3 : DuplicateReq
4 : show request
5 : ShowReq
6 : select yes
7 : send request
8 : createNewDuplicate
9 : send request
10 : save data
11 : currentCardID
12 : pass data
13 : pass data
14 : viewCards
15 : show data
16 : select no
17 : send request
18 : CDuplicateReq
19 : viewCards
20 : show data

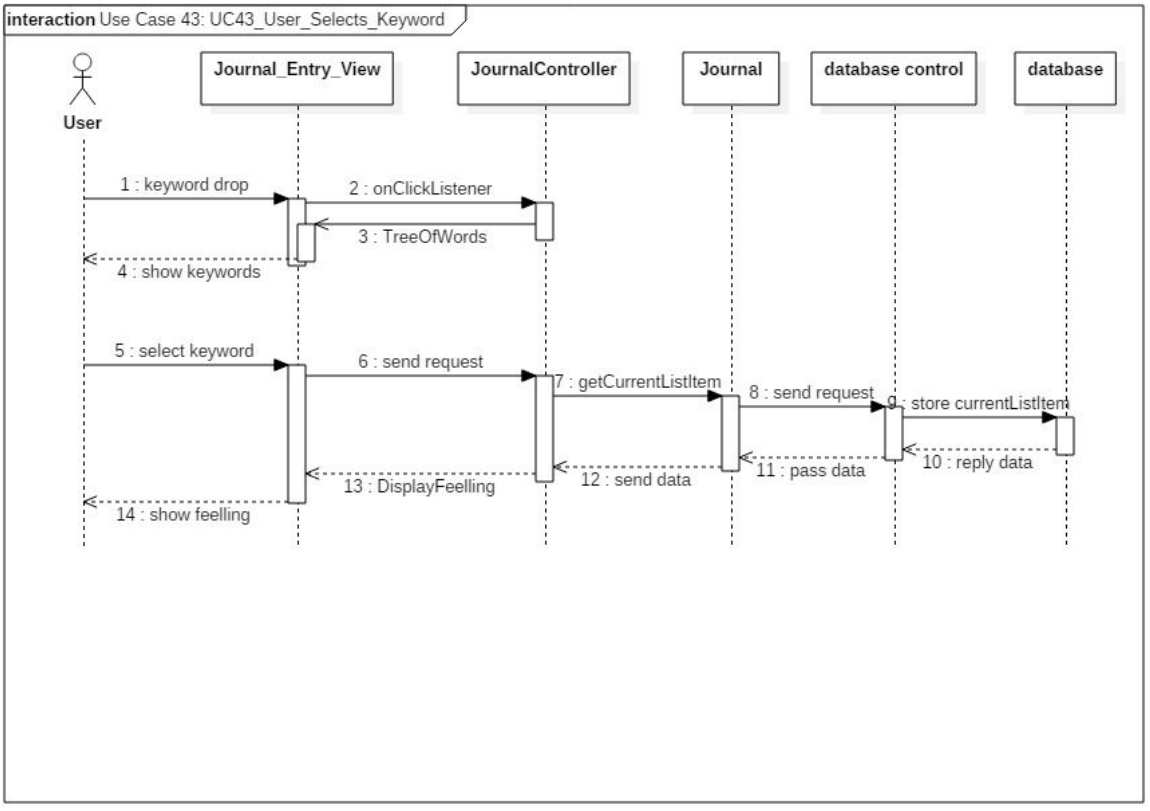

interaction UC11_User_Favourites_Card
User
«WellbeingCard»
Wellbeing_Card_View
CardContreller
Favorite_Cards_View
«Cards»
Cards
database control
database
1 : add favorite
2 : send request
3 : isFavourite
4 : send data
5 : getCurrentCardID
6 : reply data
7 : pass data
8 : pass data
9 : viewCards
10 : show data

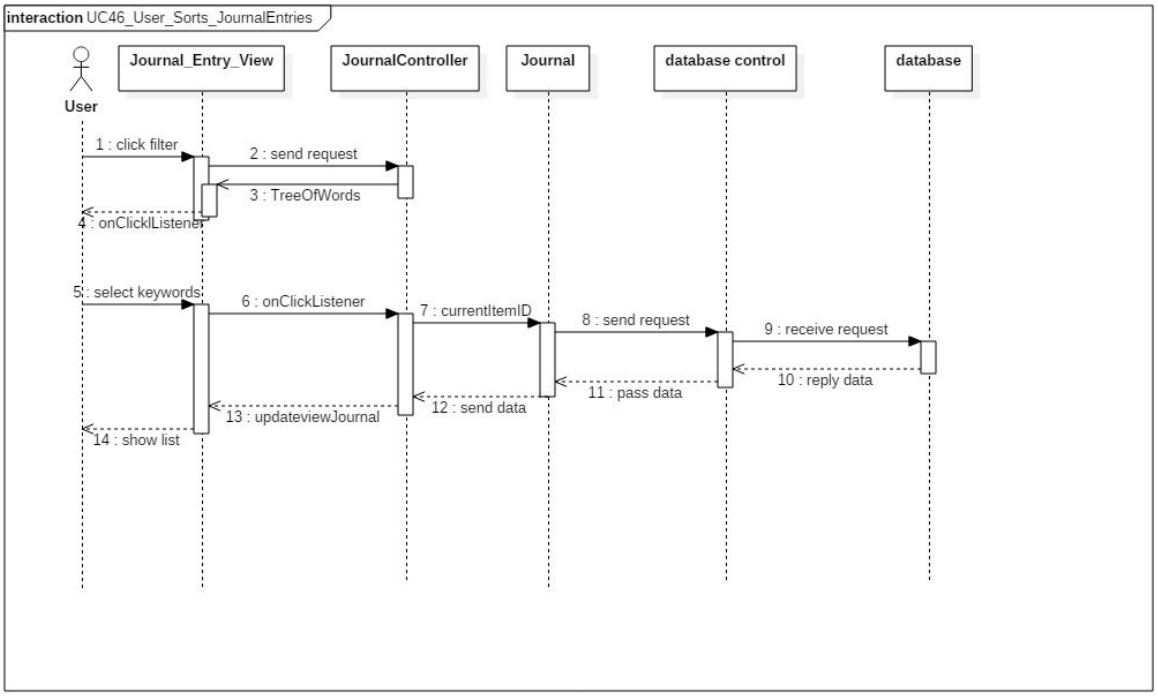

interaction Use Case 15: UC15_User_Creates_CustomCard
User
«WellbeingCard»
Wellbeing_Card_View
CardContreller
editCustomCard_View
«Cards»
Cards
database control
database
1 : createNew
2 : send request
3 : createCard
4 : send data
5 : NewCardTable
6 : lblTitle
7 : send request
8 : KeyboardPopUp
9 : show keyboard
10 : Add picture
11 : send request
12 : cropImage
13 : show image
14 : show image
15 : save card
16 : send request
17 : CardSave
18 : send request
19 : save data
20 : reply data
21 : pass data
22 : notifyDataSetChanged
23 : viewCards
24 : show data
25 : cancel edit
26 : send request
27 : CardDelete
28 : send request
29 : save data
30 : reply data
31 : pass data
32 : viewCards
33 : show page

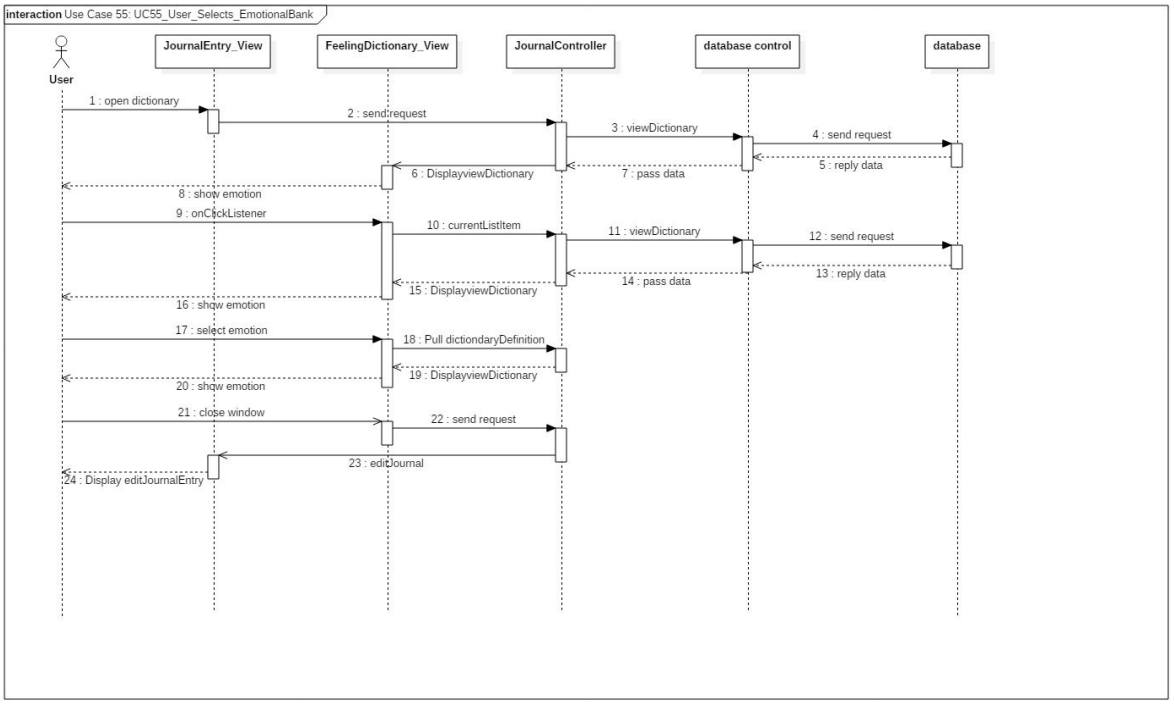

interaction Use Case 27: UC27_User_Edits_CustomCard
User
«WellbeingCard»
Wellbeing_Card_View
editCustomCard_View
CardContreller
«Cards»
Cards
database control
database
1 : select card
2 : custom card
3 : show card
4 : show card
5 : selects title
6 : send request
7 : KeyboardPopUp
8 : show card
9 : edit card
10 : send request
11 : cropImage
12 : send request
13 : save data
14 : reply data
15 : pass data
16 : pass data
17 : viewCards
18 : show card
19 : save card
20 : send request
21 : CardSubmition
22 : send request
23 : save data
24 : reply data
25 : pass data
26 : pass data
27 : notifyDataSetChanged
28 : show data
29 : discard card
30 : select yes
31 : abandon card
32 : show card
33 : discard card
34 : select no
35 : show editcard
36 : return back

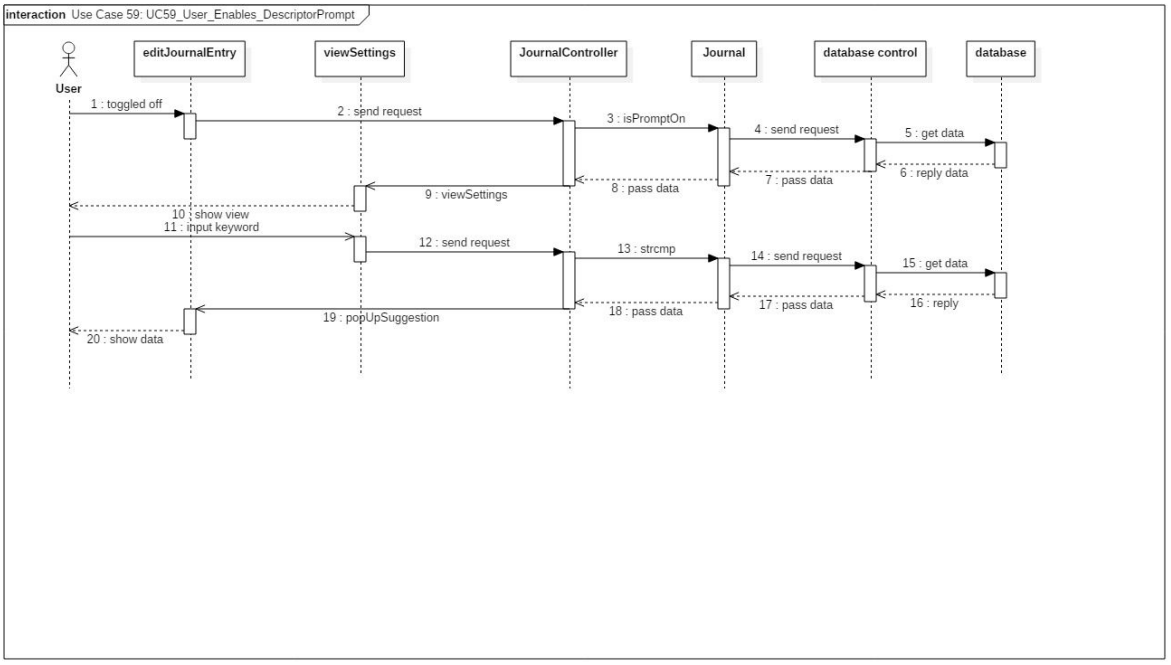

interaction Use Case 31: UC31_User_Creates_JournalEntry
User
Journal_View
editJournalEntry_View
JournalController
Journal
database control
database
1 : create journal
2 : onClickListener
3 : DisplayJournalEntry
4 : show page
5 : select keyword
6 : TreeOfWords
7 : AddJournalFeelling
8 : send data
9 : save data
10 : reply data
11 : pass data
12 : pass data
13 : Displayfeelling
14 : show feelling
15 : edit journal
16 : send request
17 : journalText
18 : show text
19 : viewJournal
20 : show text
21 : cancel text
22 : send request
23 : DisplayDialog
24 : asking selection
25 : select yes
26 : send request
27 : CloseJournalEntry
28 : ReturnJournal
29 : show journal
30 : select no
31 : send request
32 : CloseDialog
33 : show text
34 : save journal
35 : send request
36 : JournalSubmition
37 : send data
38 : save data
39 : reply data
40 : pass data
41 : pass data
42 : DisplayJournalList
43 : show list

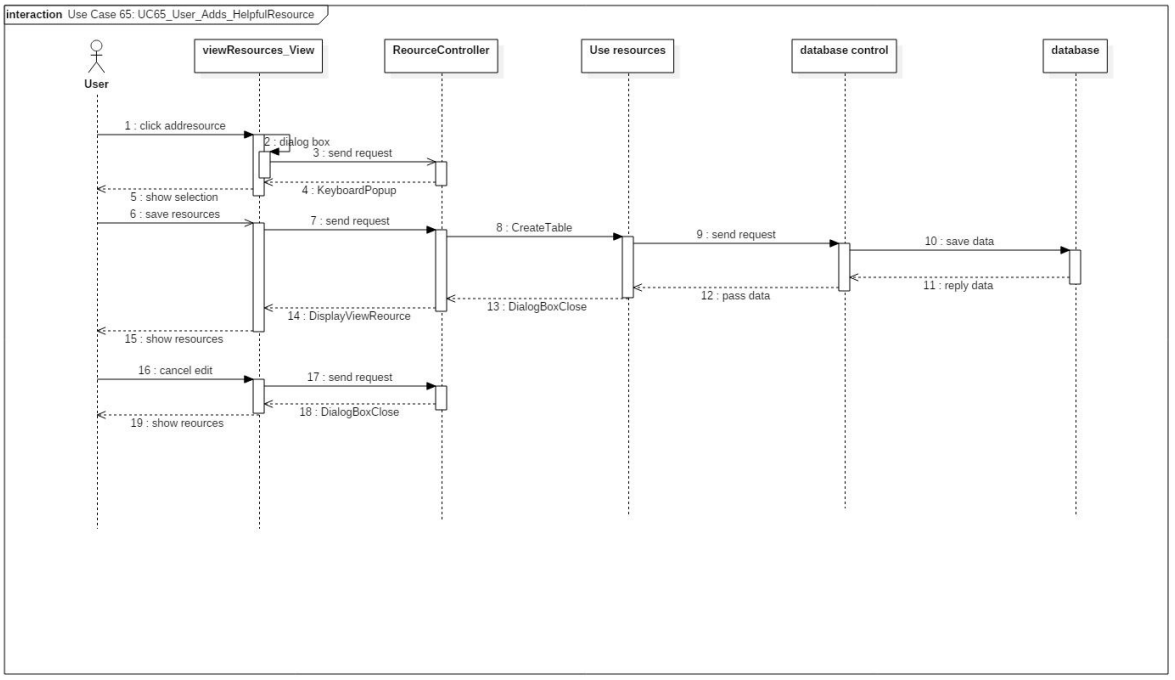

interaction Use Case 38: UC38_User_Edits_JournalEntry
User
Journal_View
Journal_Entry_View
JournalController
Journal
database control
database
1 : OpenJournal
2 : edit journal
3 : send request
4 : editJournal
5 : send request
6 : send request
7 : KeyboardPopup
8 : reply data
9 : show text
10 : pass data
11 : pass data

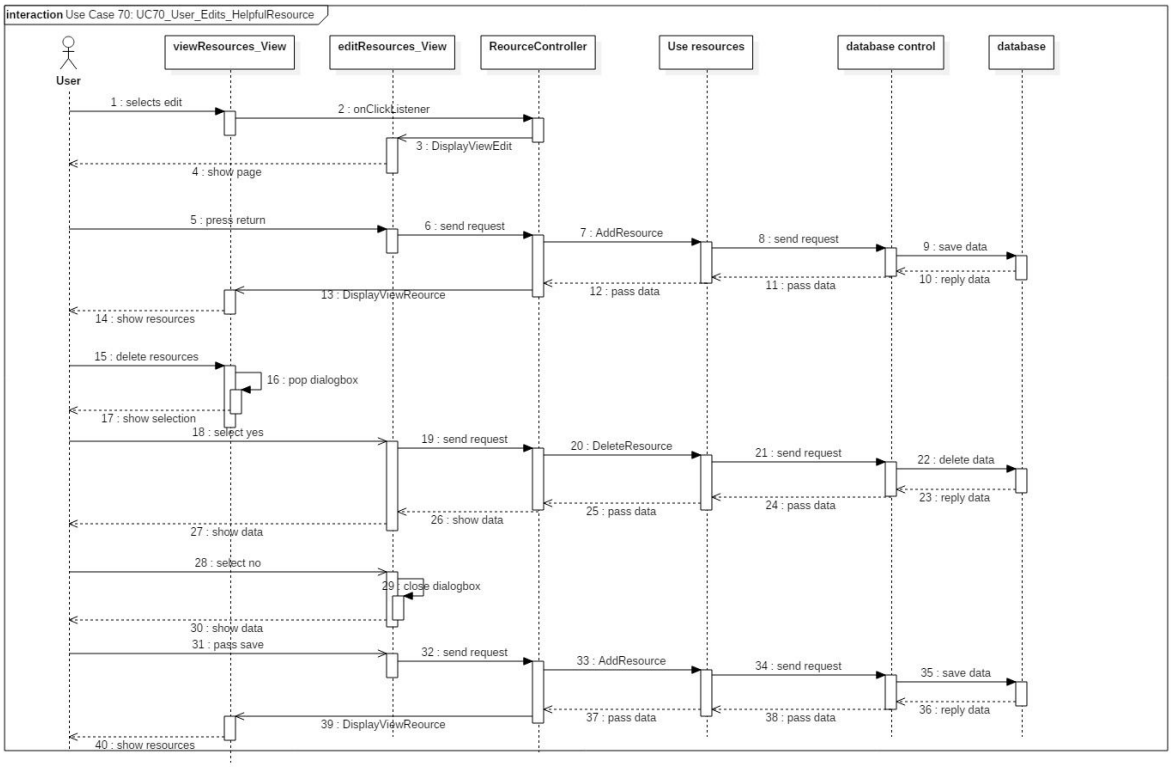

interaction Use Case 43: UC43_User_Selects_Keyword
User
Journal_Entry_View
JournalController
Journal
database control
database
1 : keyword drop
2 : onClickListener
3 : TreeOfWords
4 : show keywords
5 : select keyword
6 : send request
7 : getCurrentListItem
8 : send request
9 : store currentListItem
10 : reply data
11 : pass data
12 : send data
13 : DisplayFeelling
14 : show feelling


interaction UC46_User_Sorts_JournalEntries
User
Journal_Entry_View
JournalController
Journal
database control
database
1 : click filter
2 : send request
3 : TreeOfWords
4 : onClickListener
5 : select keywords
6 : onClickListener
7 : currentItemID
8 : send request
9 : receive request
10 : reply data
11 : pass data
12 : send data
13 : updateviewJournal
14 : show list


interaction Use Case 55: UC55_User_Selects_EmotionalBank
User
JournalEntry_View
FeelingDictionary_View
JournalController
database control
database
1 : open dictionary
2 : send request
3 : viewDictionary
4 : send request
5 : reply data
6 : DisplayviewDictionary
7 : pass data
8 : show emotion
9 : onClickListener
10 : currentListItem
11 : viewDictionary
12 : send request
13 : reply data
14 : pass data
15 : DisplayviewDictionary
16 : show emotion
17 : select emotion
18 : Pull dictiondaryDefinition
19 : DisplayviewDictionary
20 : show emotion
21 : close window
22 : send request
23 : editJournal
24 : Display editJournalEntry

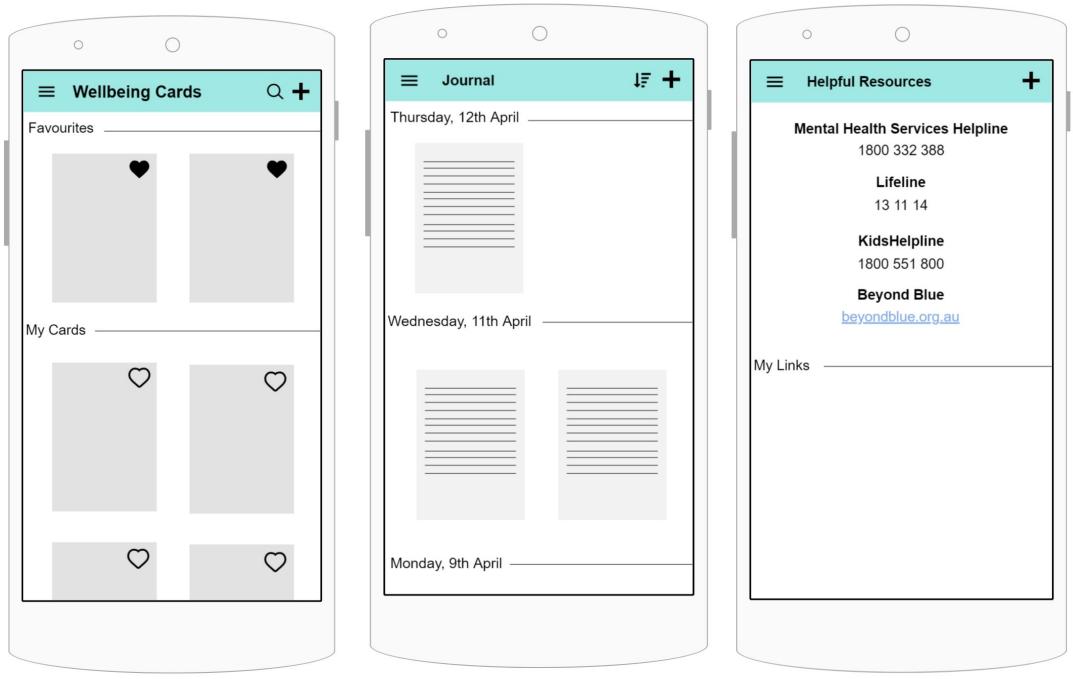

interaction Use Case 59: UC59_User_Enables_DescriptorPrompt
User
editJournalEntry
viewSettings
JournalController
Journal
database control
database
1 : toggled off
2 : send request
3 : isPromptOn
4 : send request
5 : get data
6 : reply data
7 : pass data
8 : pass data
9 : viewSettings
10 : show view
11 : input keyword
12 : send request
13 : strcmp
14 : send request
15 : get data
16 : reply
17 : pass data
18 : pass data
19 : popUpSuggestion
20 : show data


interaction Use Case 65: UC65_User_Adds_HelpfulResource
User
viewResources_View
ReourceController
Use resources
database control
database
1 : click addresource
2 : dialog box
3 : send request
4 : KeyboardPopup
5 : show selection
6 : save resources
7 : send request
8 : CreateTable
9 : send request
10 : save data
11 : reply data
12 : pass data
13 : DialogBoxClose
14 : DisplayViewReource
15 : show resources
16 : cancel edit
17 : send request
18 : DialogBoxClose
19 : show reources


interaction Use Case 70: UC70_User_Edits_HelpfulResource
User
viewResources_View
editResources_View
ReourceController
Use resources
database control
database
1 : selects edit
2 : onClickListener
3 : DisplayViewEdit
4 : show page
5 : press return
6 : send request
7 : AddResource
8 : send request
9 : save data
10 : reply data
11 : pass data
12 : pass data
13 : DisplayViewReource
14 : show resources
15 : delete resources
16 : pop dialogbox
17 : show selection
18 : select yes
19 : send request
20 : DeleteResource
21 : send request
22 : delete data
23 : reply data
24 : pass data
25 : pass data
26 : show data
27 : show data
28 : select no
29 : close dialogbox
30 : show data
31 : pass save
32 : send request
33 : AddResource
34 : send request
35 : save data
36 : reply data
37 : pass data
38 : pass data
39 : DisplayViewReource
40 : show resources

# 8. Appendix III: Interface Design and Iterations

## 8.1. Prototype 1 - Version 1




This prototype focuses heavily on a materialistic theme that is found commonly throughout Android applications. The Wellbeing cards screen would act as the main point of interaction and launch screen for the application. Favourite cards are segmented from the main collection of Wellbeing cards and placed at the top as they are likely to be interacted with most often. Simply toggling the heart icon available on each card will add the selected card to the favourites section at the top of the screen, which will then expand downwards to accommodate the addition. Basic interactions with cards involve long pressing to access card options such as delete and edit, while a single press will expand that card to reveal its description.

The plus icon located top left enables the user to add a new card. The user would then be given the option to create a duplicate of an existing card or start from scratch, launching the card editor.

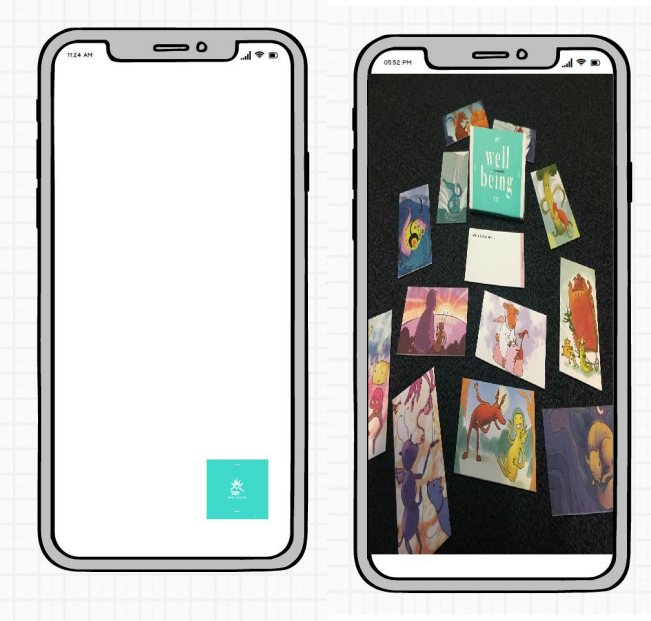
The journal screen follows a similar theme with the journal entries utilising the same basic layout for displaying items, the entries are divided by date (most recent first) with the ability to sort them by keyword or date available through the sort button located top left of the screen.

The Helpful Resources screen is very simple in its functionality as it does not large amounts of detail. It is intended to be used as a reference for users should they need to contact a specific organisation, therefore this interface design strives to keep the page as clean and clutter free as possible. The addition of new entries can be accessed through the plus button as on previous screens.

A core design aspect of this prototype is to keep the user experience consistent. Adding new items remains accessible from the top left of the screen across all views, the ‘burger menu’ system allows for a hidden main menu that can be accessed at any time, while all ‘main’ screens are structurally similar, utilising the categorised list view type.

Navigation on Android devices will utilise the hardware/software buttons on the device being used. The IOS variation of this prototype would implement navigation buttons on screen.

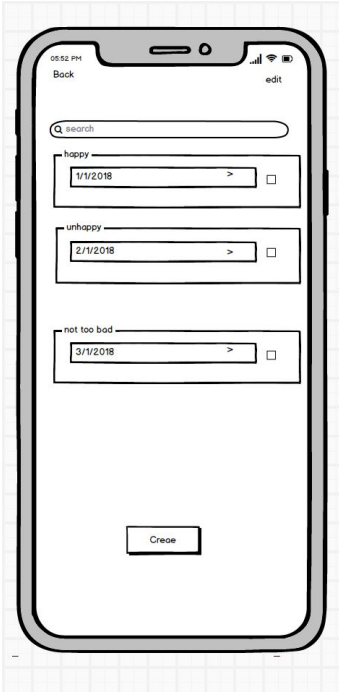
## 8.2. Prototype 2 - Version 1



The first two images in this prototype feature mockups of logos and loading screens respectively.


In this view, the user can swipe the screen to view different cards. The user can also select their favourite cards or copy the existing card to create another one using the current card's title or image.
If it is a custom card, the edit button will be presented for the user, taking them to the edit view.
The Journal Entry button will take users to the journal view.
The back button at the top left of the screen will take the user back to the home screen.

The back of the card can be viewed by double tapping its front. Selecting the edit button will allow users to add text to the back of the card.



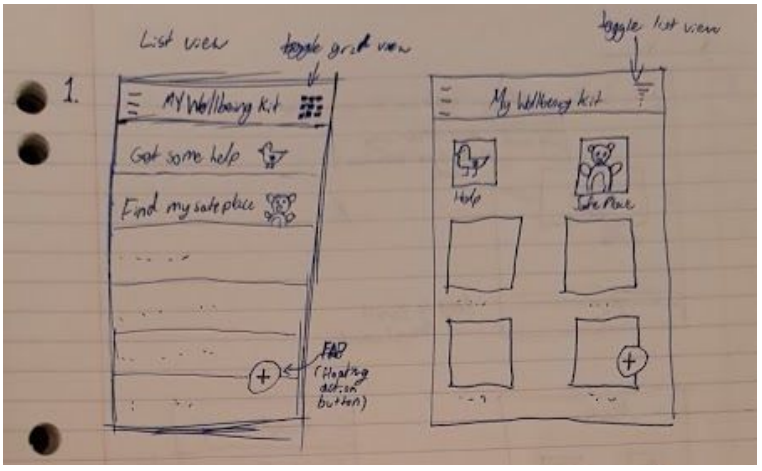

On viewing the list of journal entries, the user will be able to sort through them by looking at their date of entry, or alternatively, their emotional category.


The user will be able to input the title, date and content of the journal entry on this screen. A ‘choose’ button can be selected in the top right of the view to select a related keyword for the entry.

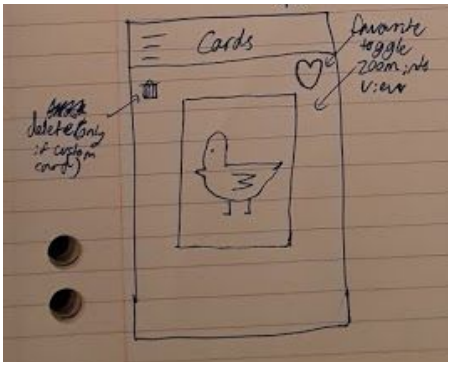
## 8.3. Prototype 3 - Version 1

### Design A

This prototype attempts to utilise a lot of the Android UI libraries to create a straightforward design that will be easily accessible to most users. Features such as the ‘burger menu’ and ‘FAB’ (floating action button) are standard in many Android applications and as such users know exactly what to expect when they use them.



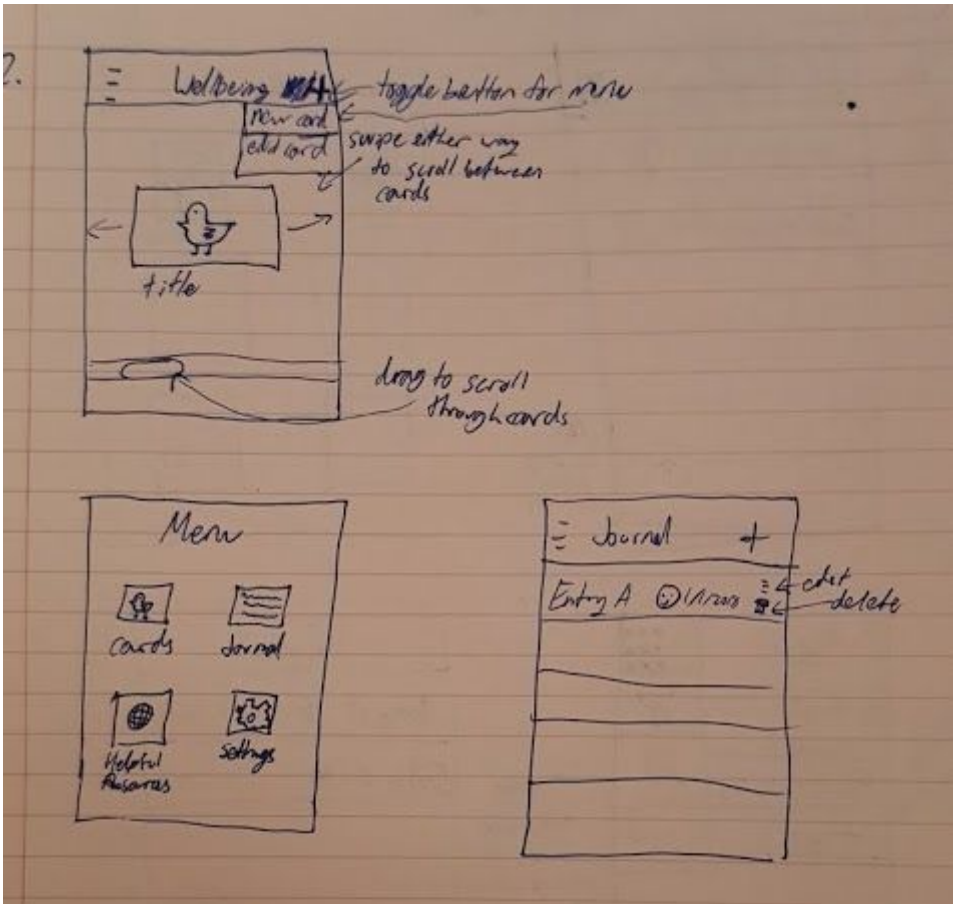

The cards can be toggled between a list and grid view in this design (the button to do this is visible in the top right of the views).




The burger menu is visible of the left. It can be closed by swiping the screen back towards the side or tapping elsewhere on the screen.
The journal entries are presented in a list view with the emotion and date used to distinguish between entries.

## Design B

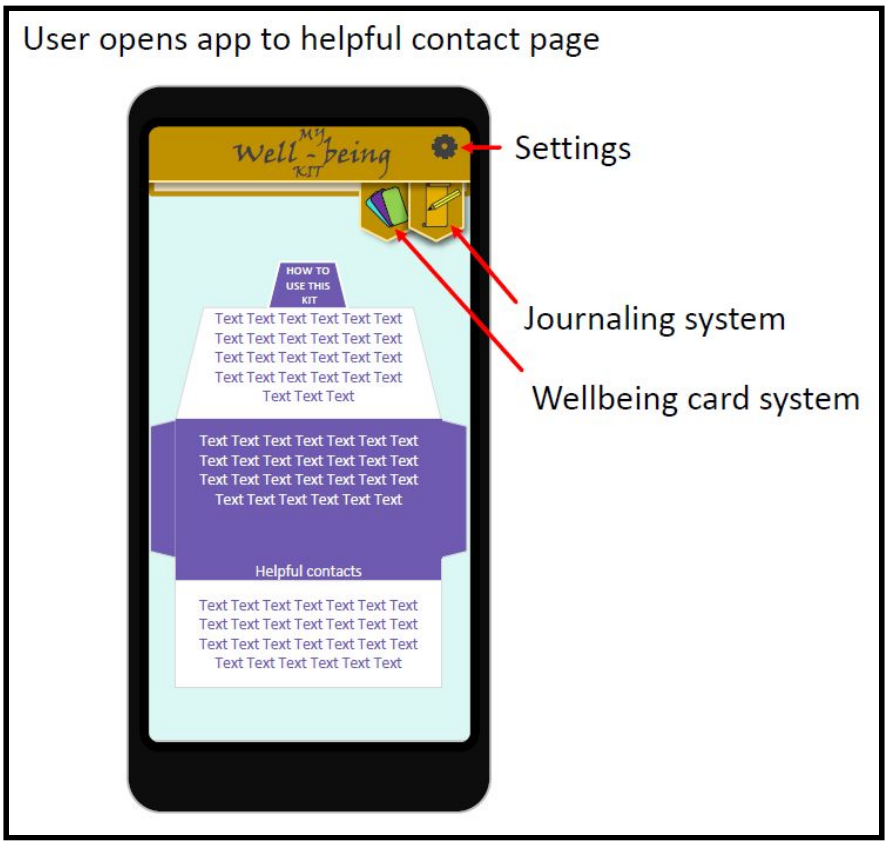
This prototype uses a less Android specific UI design. This design has a grid based menu system rather than the list and also handles the cards differently. Rather than a list or grid, the cards take up most of the screen and can be swiped between. This simulates the way users use the physical cards by flipping through a deck.



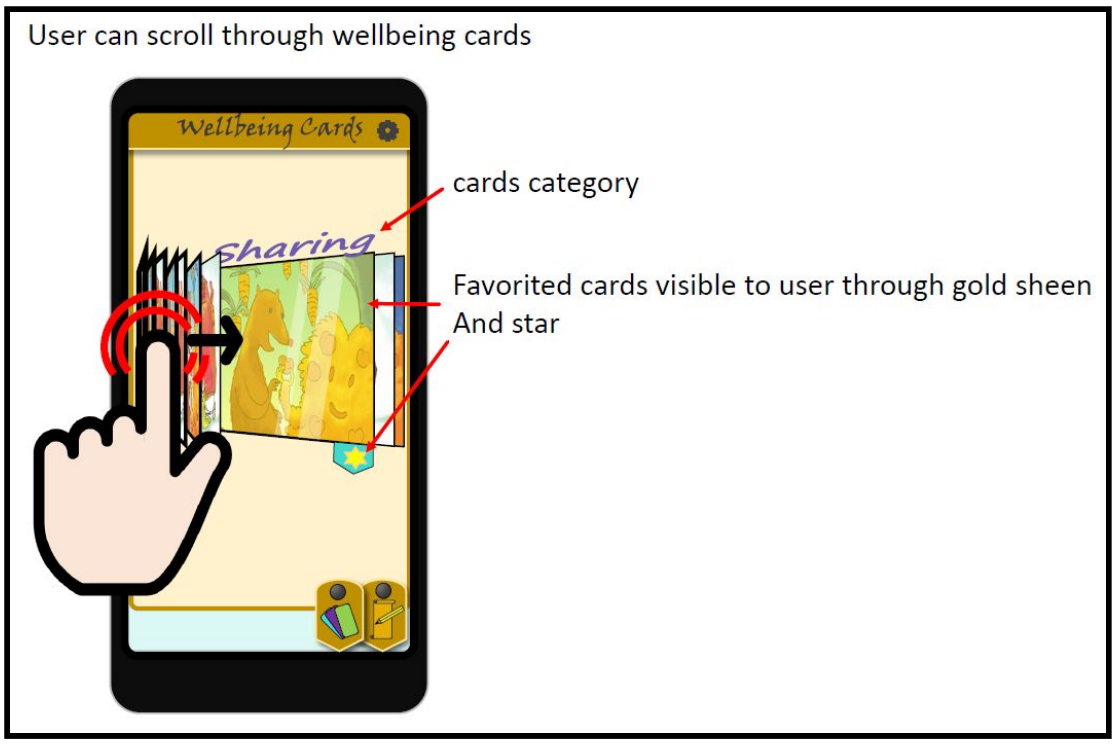
## 8.4. Prototype 4 - Version 1 (also see appendix 8.8)

### Helpful contacts

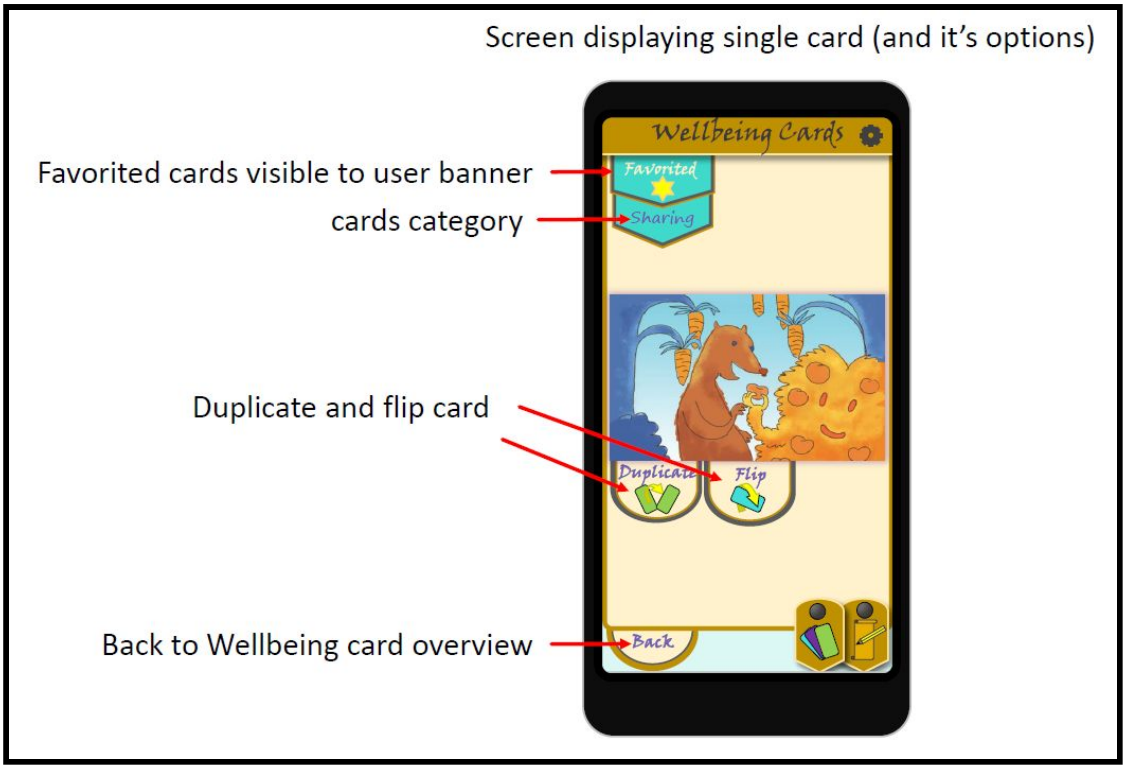
The user will open the application up to the ‘Helpful resources’ page. The other menus will be available to the user through the use of drop down windows. The tags that allow you to do swap between screens will persist throughout all widows of the application, giving the ability to navigate quickly between each window.
The settings page will also be made available to the through the cog button on the top right hand corner of each screen.



## Wellbeing Cards - Overview



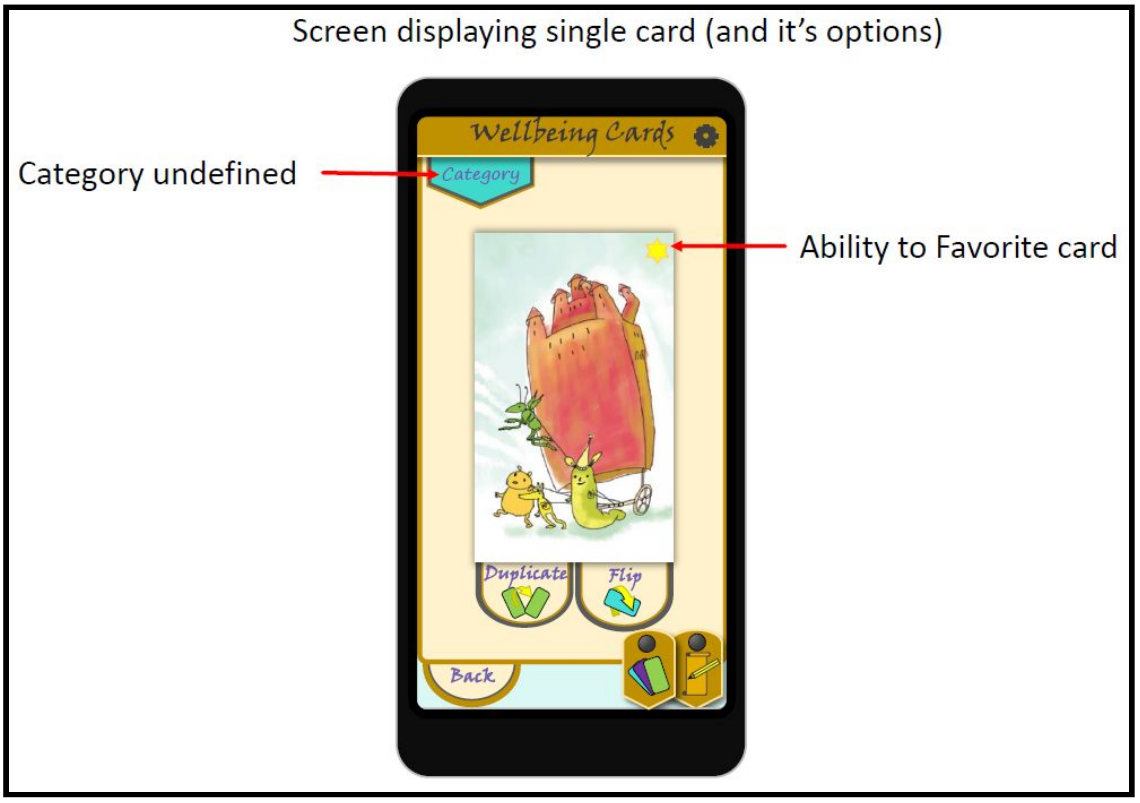

The Wellbeing card system will allow the user to scroll through their cards with the use of a dynamically animated window. This screen will provide the user with only the most import information about their cards i.e. Whether the card is favorited and the name of the card.



## Wellbeing Cards - Single card

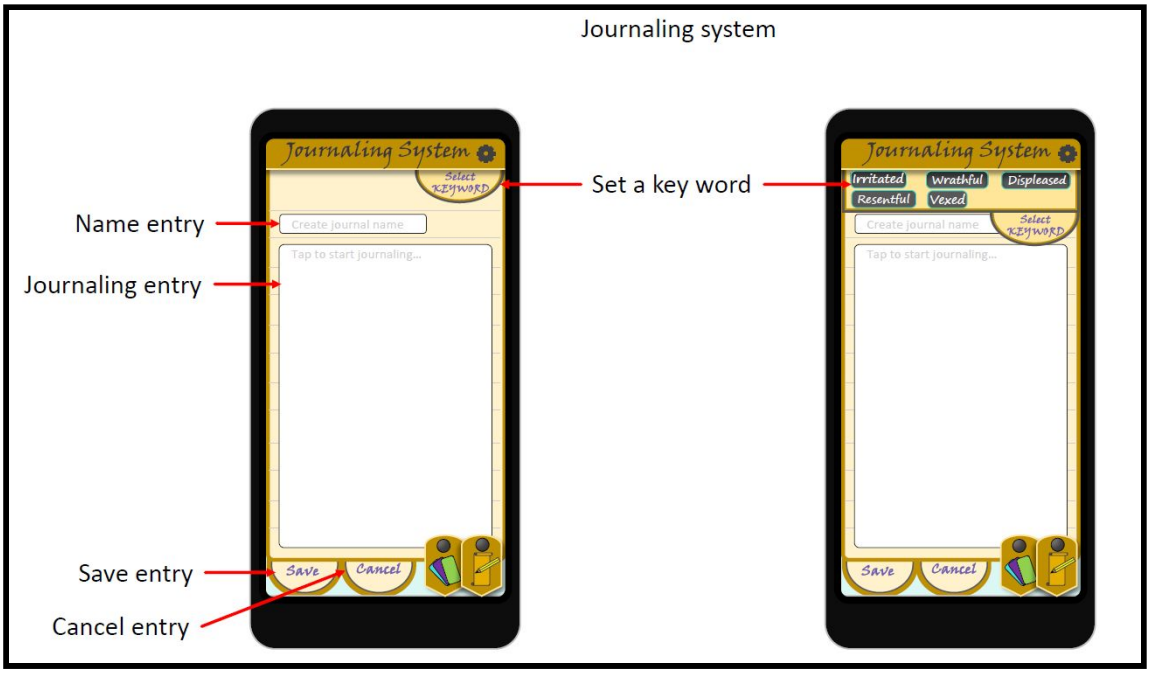
Upon clicking a card the user will be presented a screen that allows them to interact with the cards features and more finally tune their options. This screen delivers the options of Duplicating the chosen card, flipping the card over to access it's back so as to write a description, naming the card and favoriting the card.
In the prototype provided the card has been both favroited and named.

In the prototype provided the card has not been favorited or named.

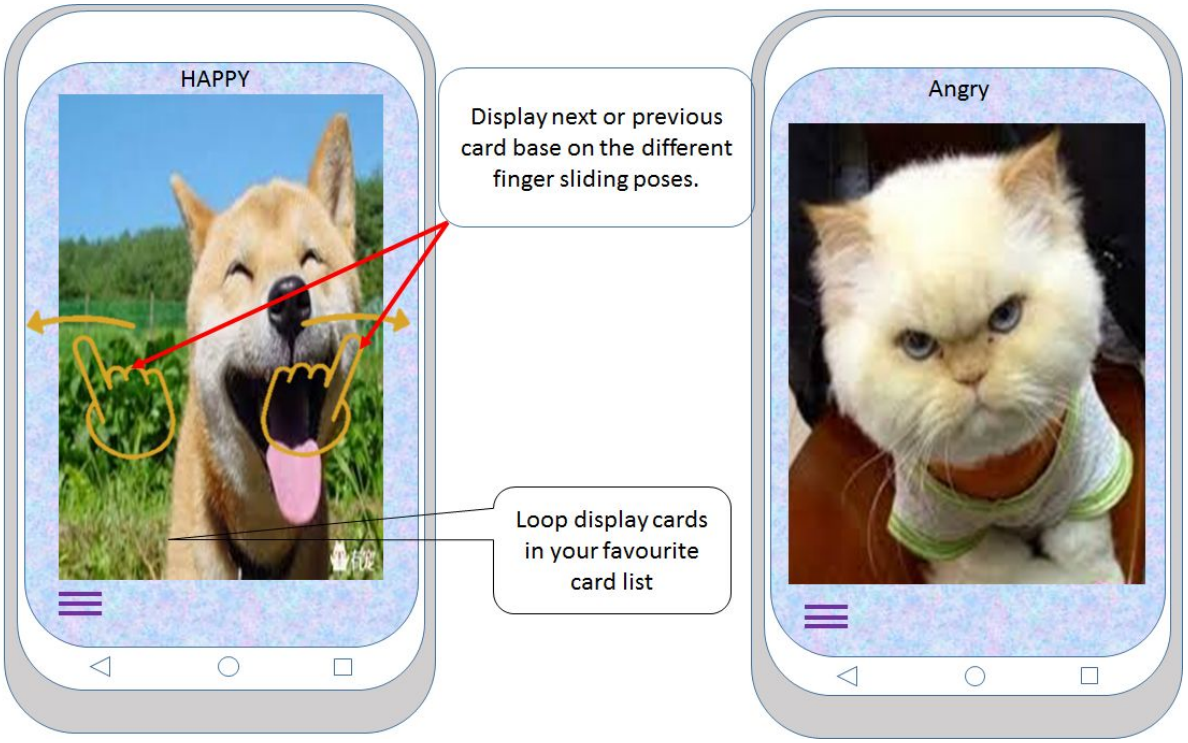


## Journaling System

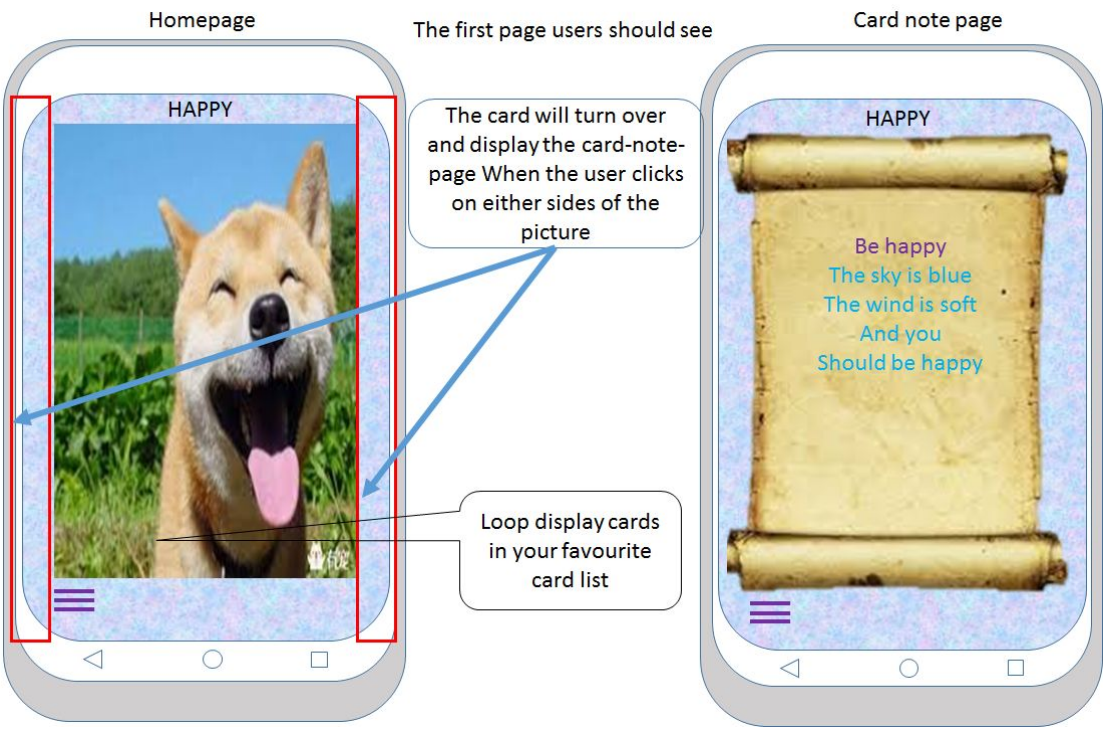
The Journaling system allows the user to create a new entry. The options provided on this screen allow the user to name their entry, input their journaling entry through plain text, use the tree of words to find better descriptors and save or cancel their entry.



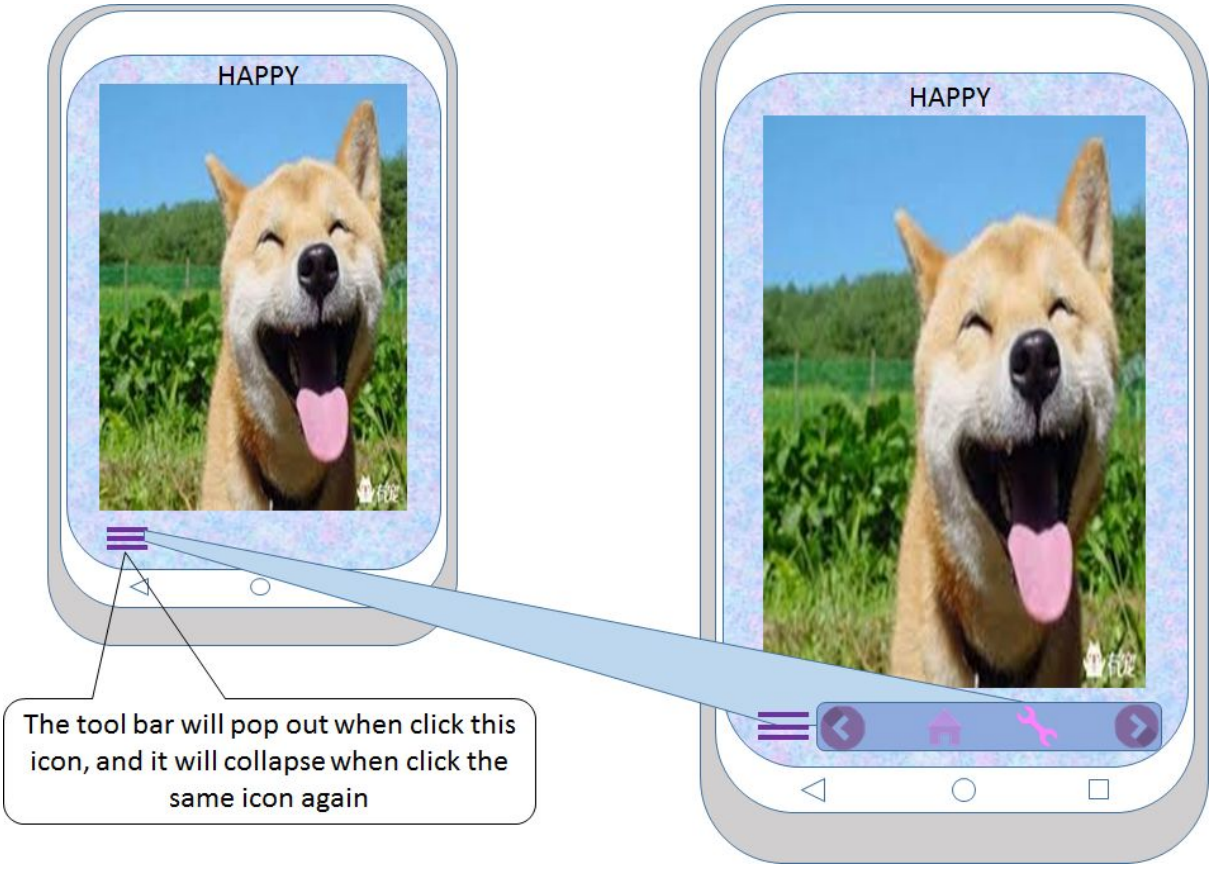
## 8.5. Prototype 5 - Version 1 (also see appendix 8.9)



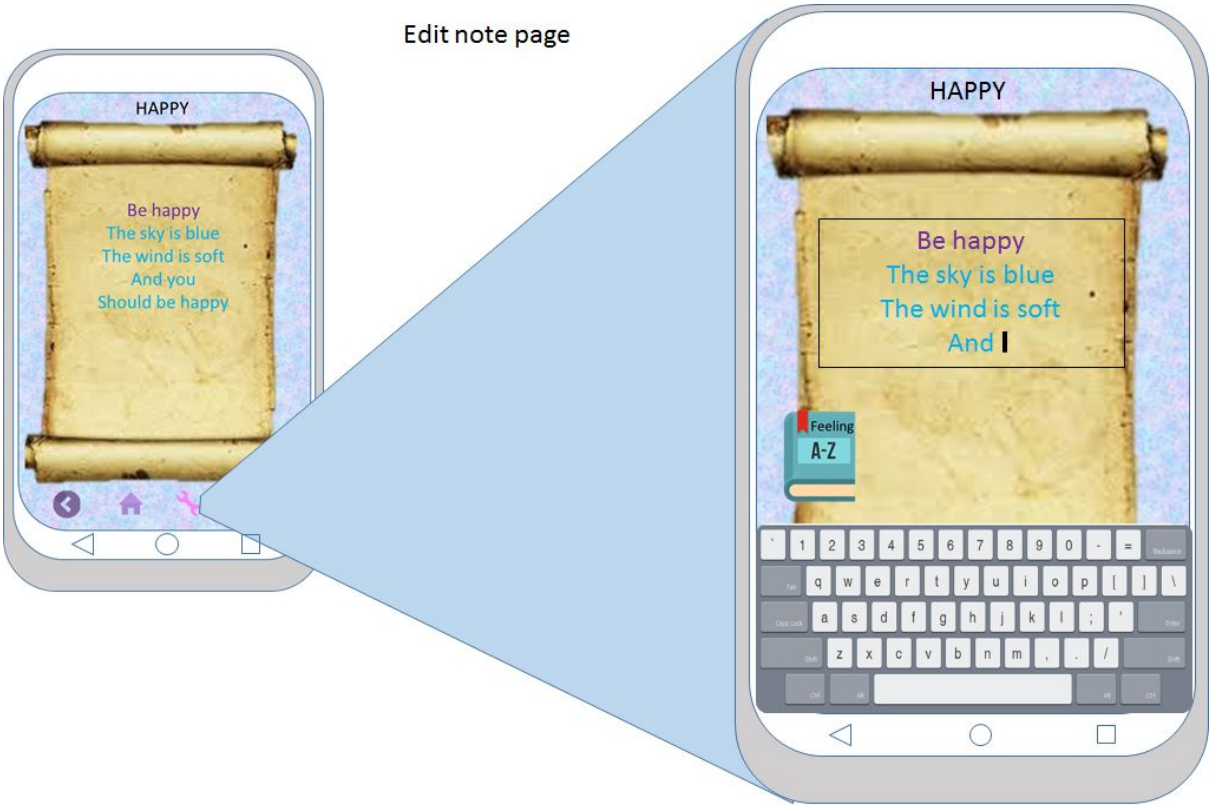

This is the first page presented to the user upon starting the application, this page will display cards which in the favourite card library.



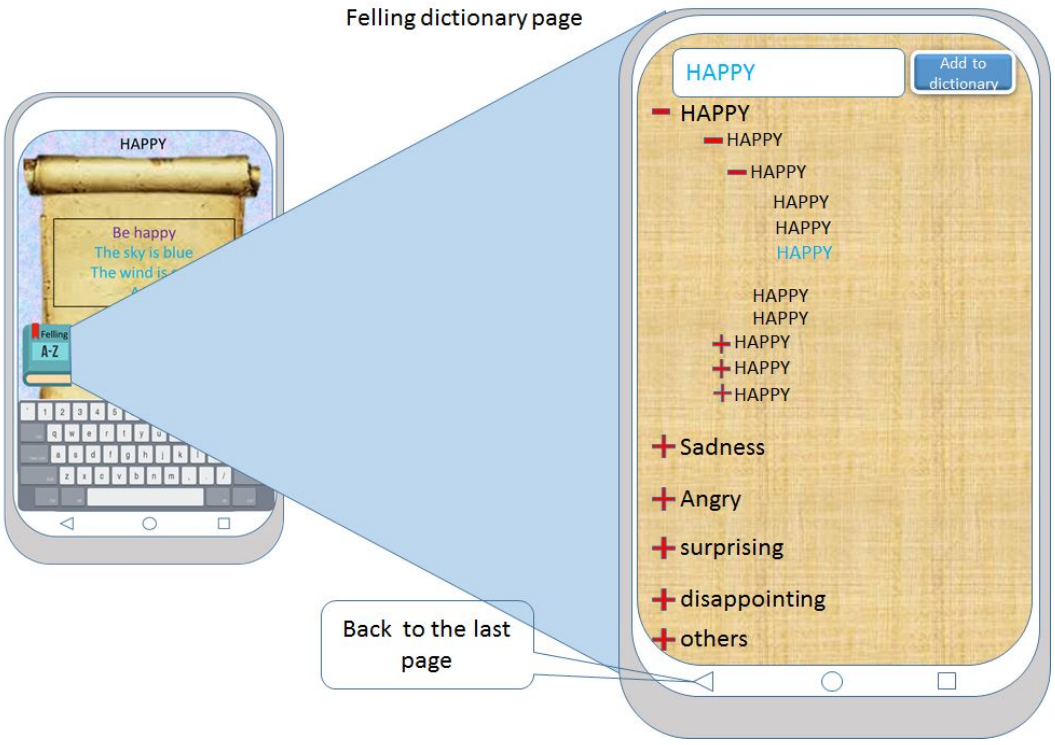

When the user touch either sides of the card, the card will turn over and display the card notes.



In order to keep the pages as simple as possible, the toolbar will be hidden until the user press the main functions icon.

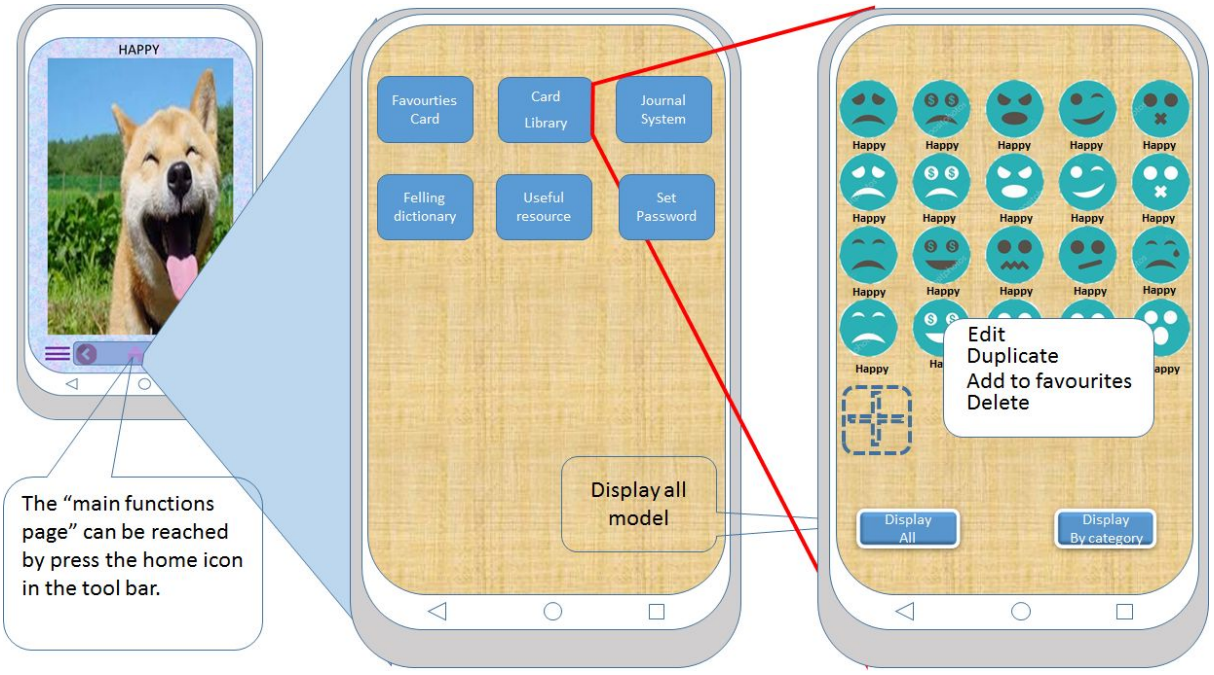



This is the edit note page, users can reach this page by pressing the edit icon on any card notes page. An outline will be displayed around the text area to illustrate to the user that this area can now be edited. And there is an icon (designed to look like a dictionary) which opens the emotional words library. Users can select it to enter the emotional words library page.

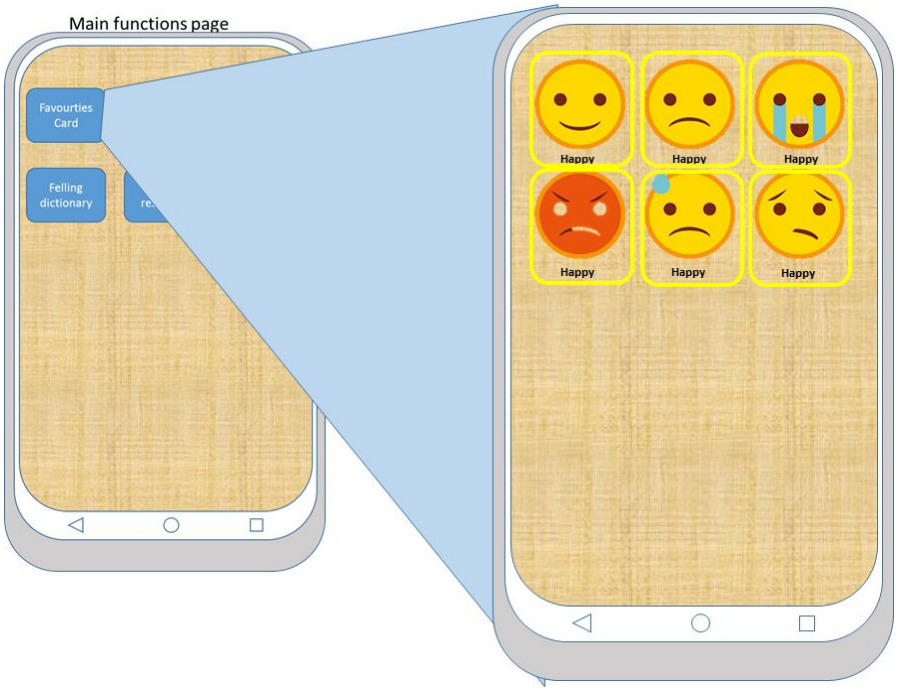



This is the feeling dictionary page, this page emotional words as a tree structure. Users can easily find needed word by going through the tree structure, and when the user finds a needed word, they can tap on the word and the word will be displayed in the textbox which the user can copy the word from.

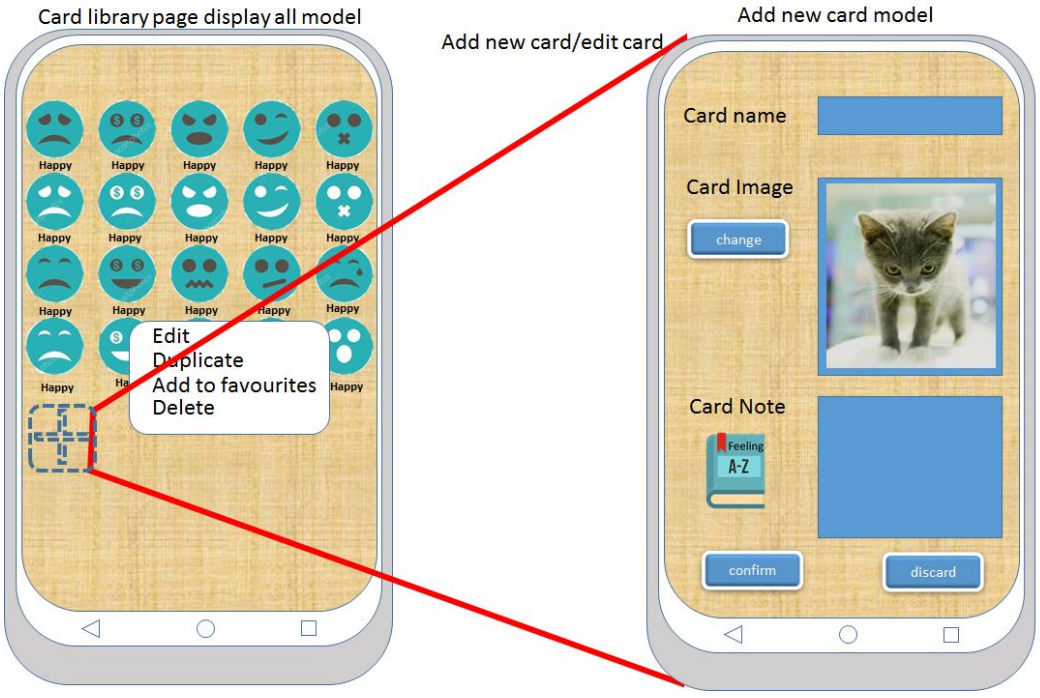
Similarly, when the user wants to add a word in the library, the user should input the word in the textbox, then the user should select appropriate category and then press the add to dictionary button.



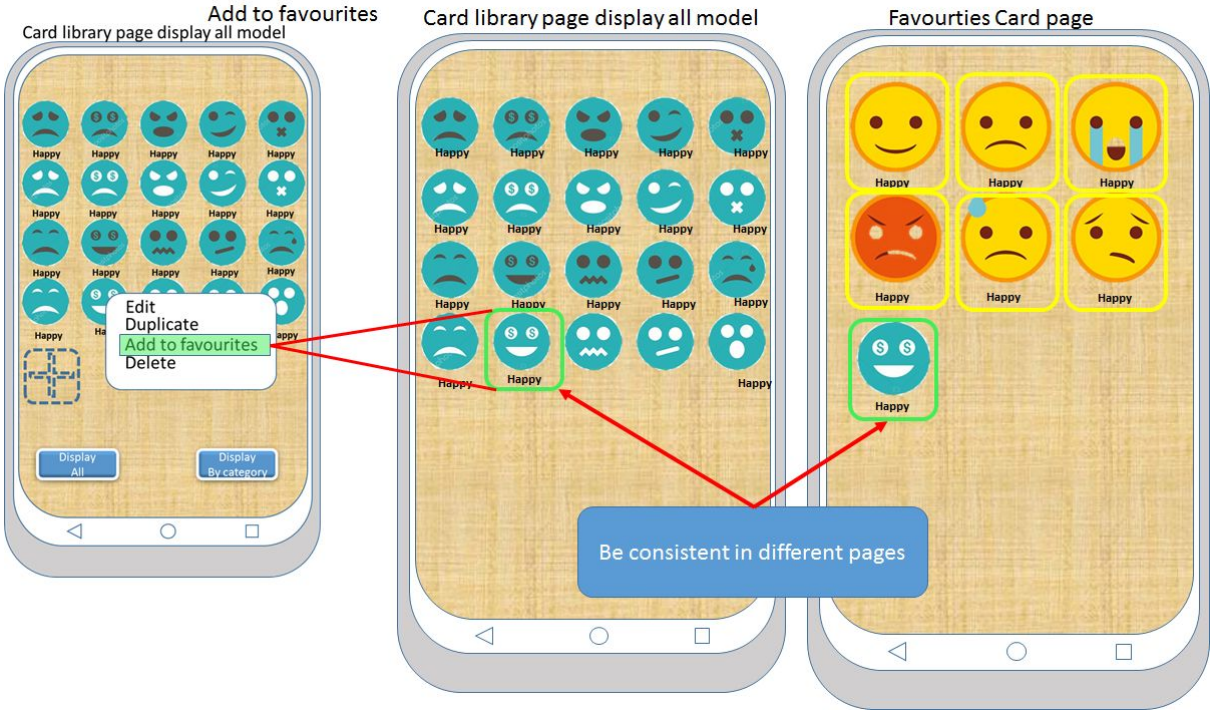
The main functions page can be reached by pressing the icon in the toolbar. This page contains a series of entries of different functions which include Favorite Cards, Card Library, Journal System, Feeling Dictionary, Useful Resources and Set Password. In the Card Library page, a secondary window will pop up when user holds down on a card thumbnail which displays edit, duplicate, add to favourites and delete functions.



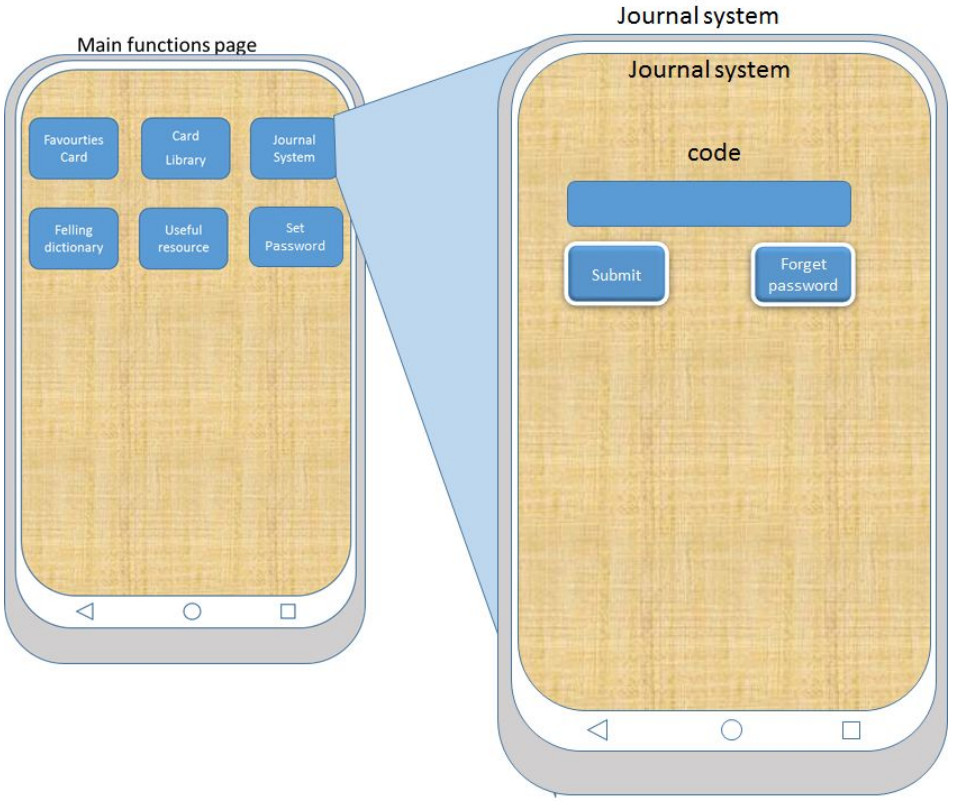

This is the favourites card  page which shows the list of favorite cards.



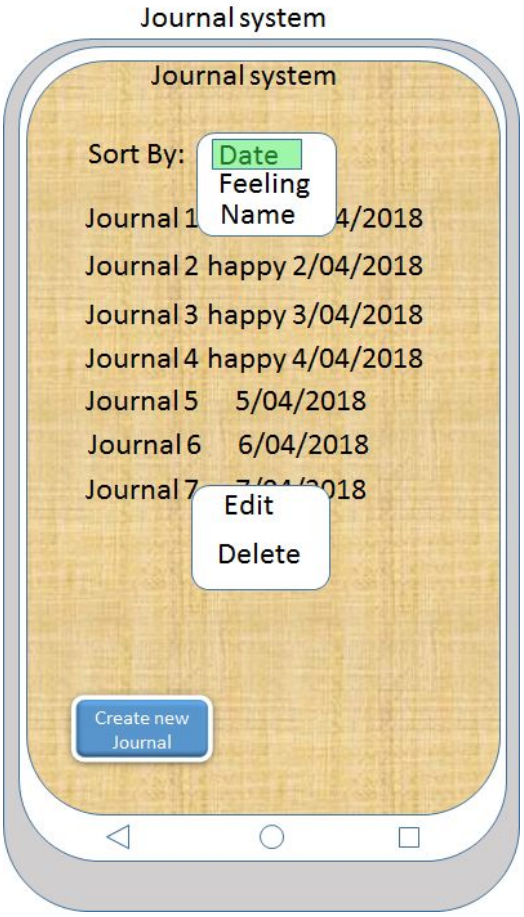
This is the add new card function , the user could enter this function by click the add icon.The user could select images and input notes for new cards.



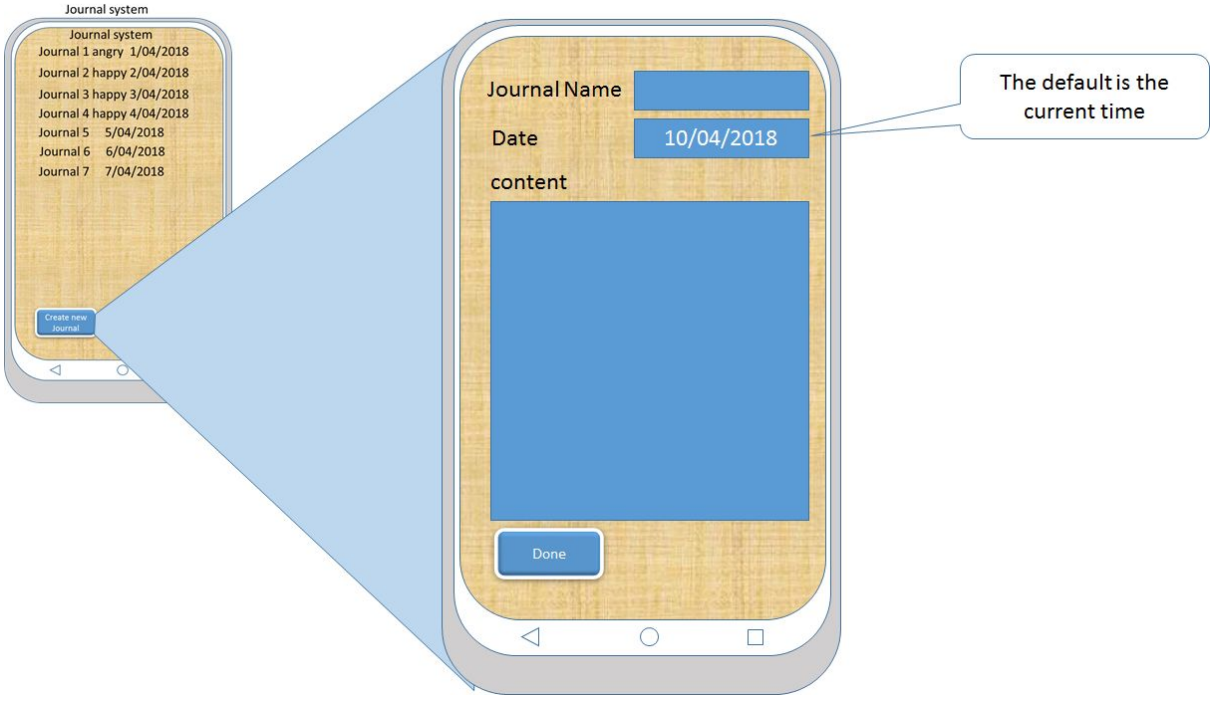

This view demonstrates how users can add any existing cards to the favourite cards library.



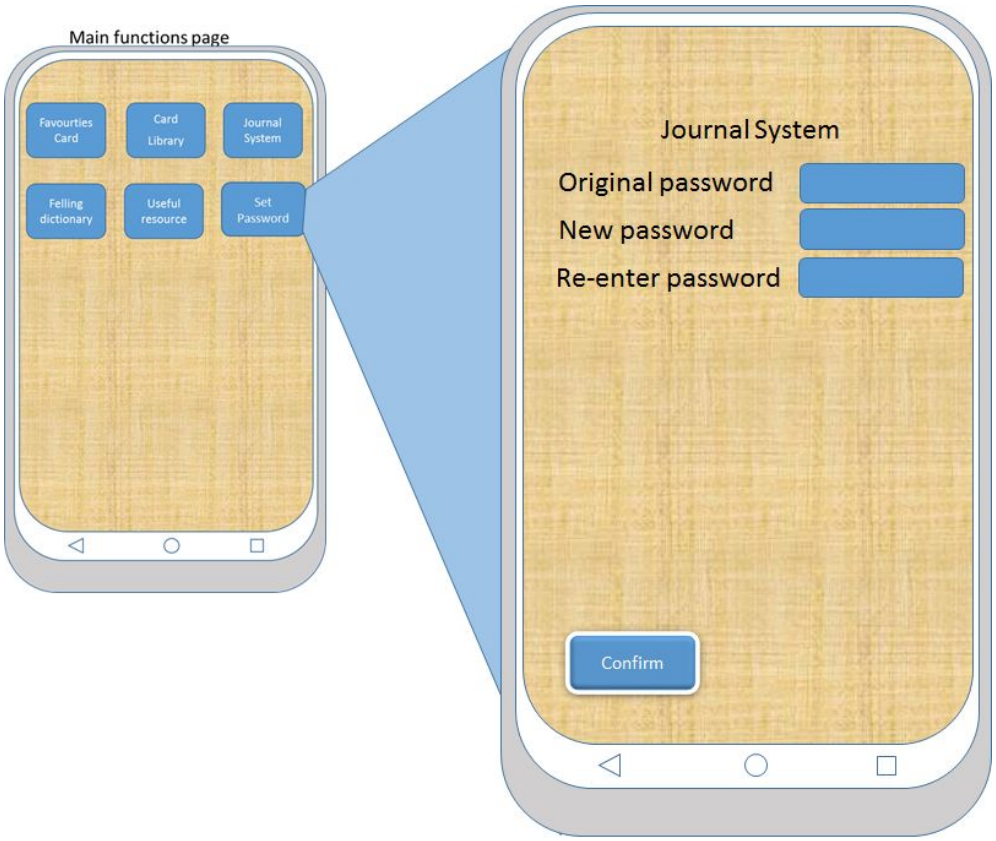

The input password page.

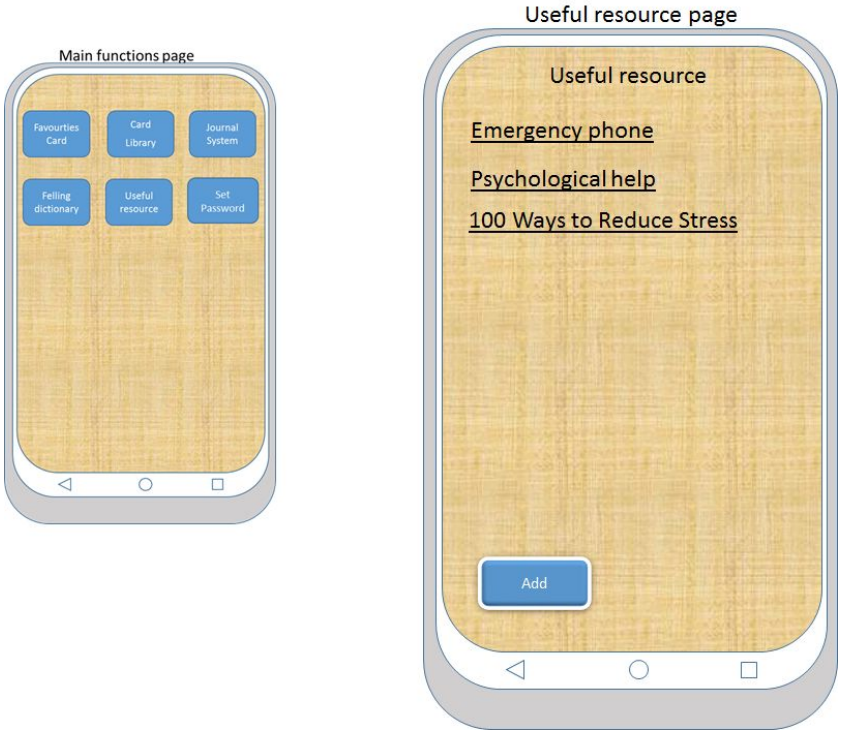



The journal system view page, the existing journal will be display as a list and user could create new journal.

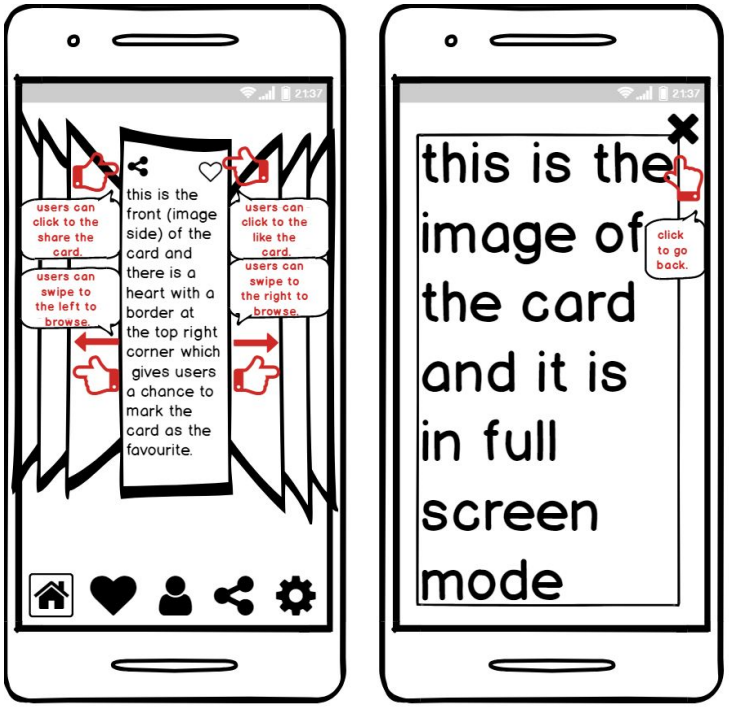



View for the user to input the name and the content of a journal entry.

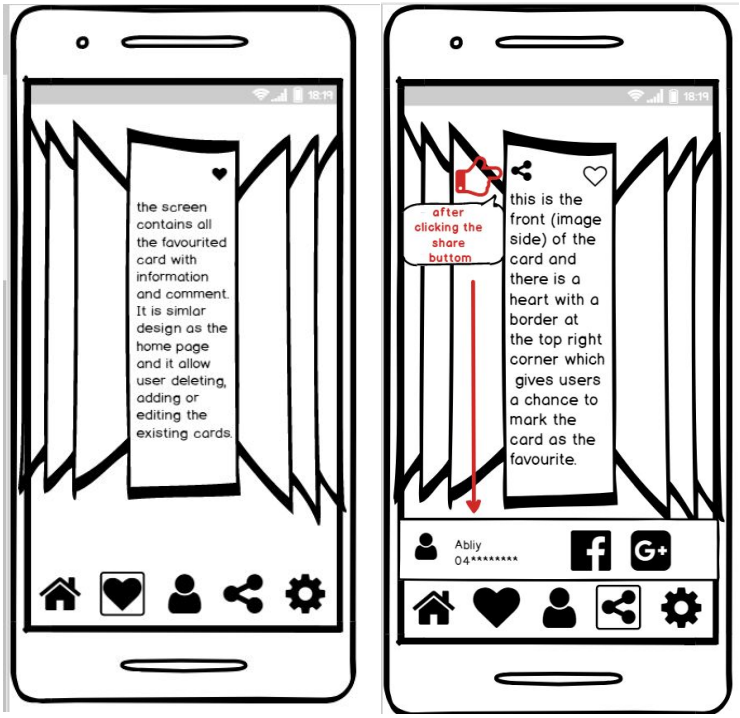

Main functions page
Favourties Card
Card Library
Journal System
Felling dictionary
Useful resource
Set Password
Journal System
Original password
New password
Re-enter password
Confirm


Main functions page
Favourties Card
Card Library
Journal System
Felling dictionary
Useful resource
Set Password
Useful resource page
Useful resource
Emergency phone
Psychological help
100 Ways to Reduce Stress
Add

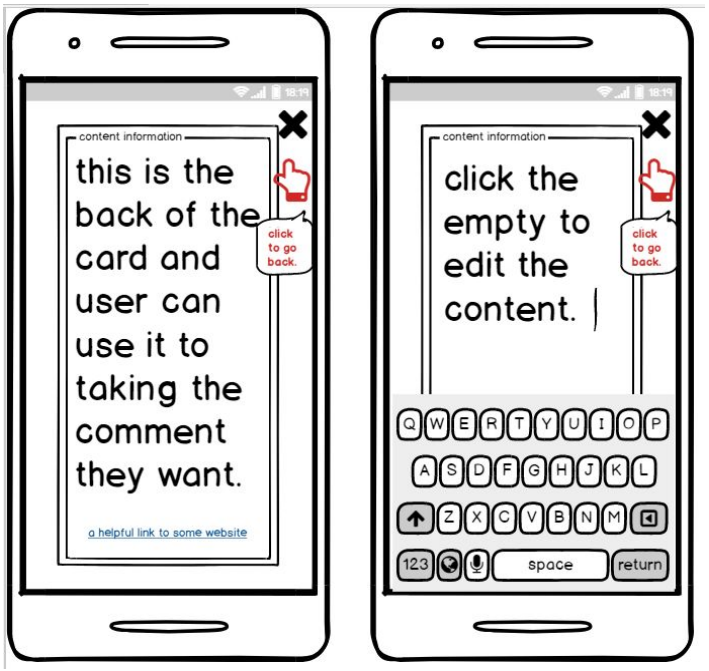
## 8.6. Prototype 6 - Version 1




This design is mainly focused on using gestures to achieve the basic functionality and navigation inside the application. Users can view different cards by swiping left or right on the screen. At the same time go to different page by simply clicking the buttons from the navigation dock at the bottom. All the functionality has a icon instead of text to keep the interface free from clutter and because icons are easily recognisable even across different screen sizes. The application also provides the ability to have a full-screen view of every card.



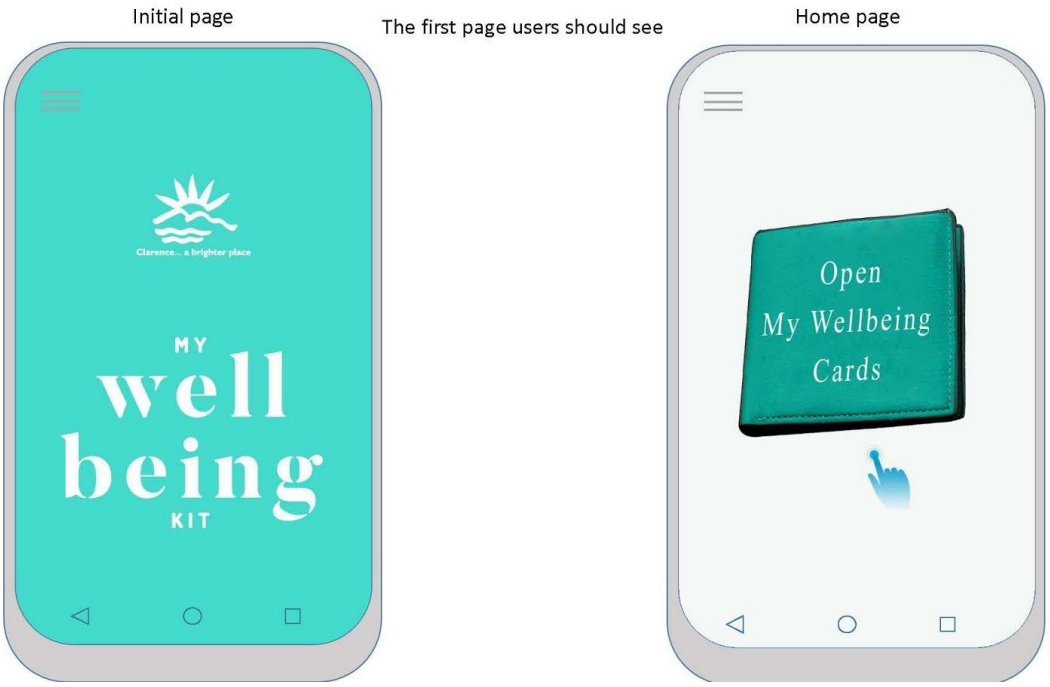
Once the ‘share’ button is selected, the current screen will display a pop-up window which allow the user to share via contact list or social media (this functionality is a possibility for Stage 3). Once they finished, the pop-up window will be hidden automatically to keep the interface clear.



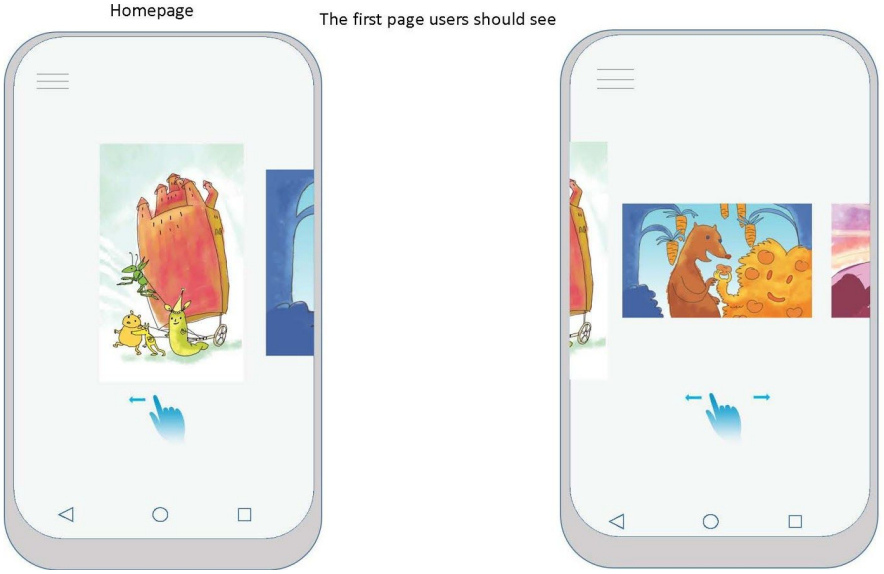

This is the back of the card, which users can view by tapping the front of the card causing the software to swap the display to the back and enter edit mode.
The cross icon in the top right corner can bring the user back to the home screen, saving changes and leaving edit mode.


This is the journal screen where users can view their journal entries. Here the user can scroll through existing entries and select any of them to bring them up and read or edit them. There is a button at the bottom of the screen which allows the user to create a new journal entry.

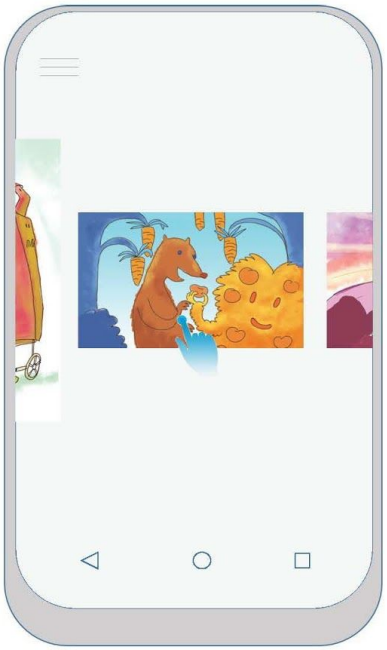
## 8.7. Prototype 7 - Version 1

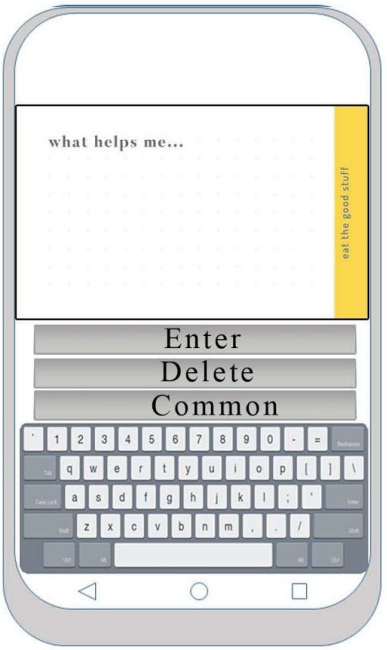



This view highlights the loading page users will be presented when opening the application and then the landing screen which will automatically be displayed when loading is completed. The user can tap on the 'kit' on screen to open the cards view.




This shows how users will use swiping gestures to scroll through the available wellbeing cards.

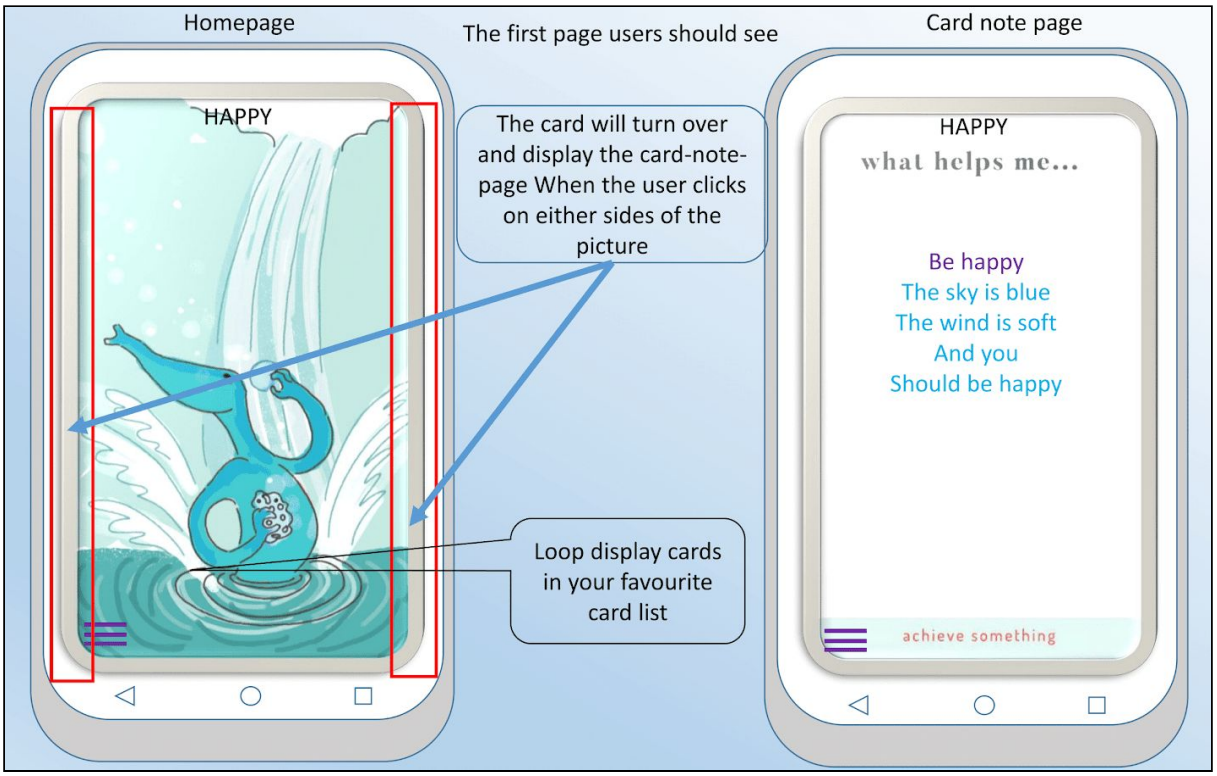




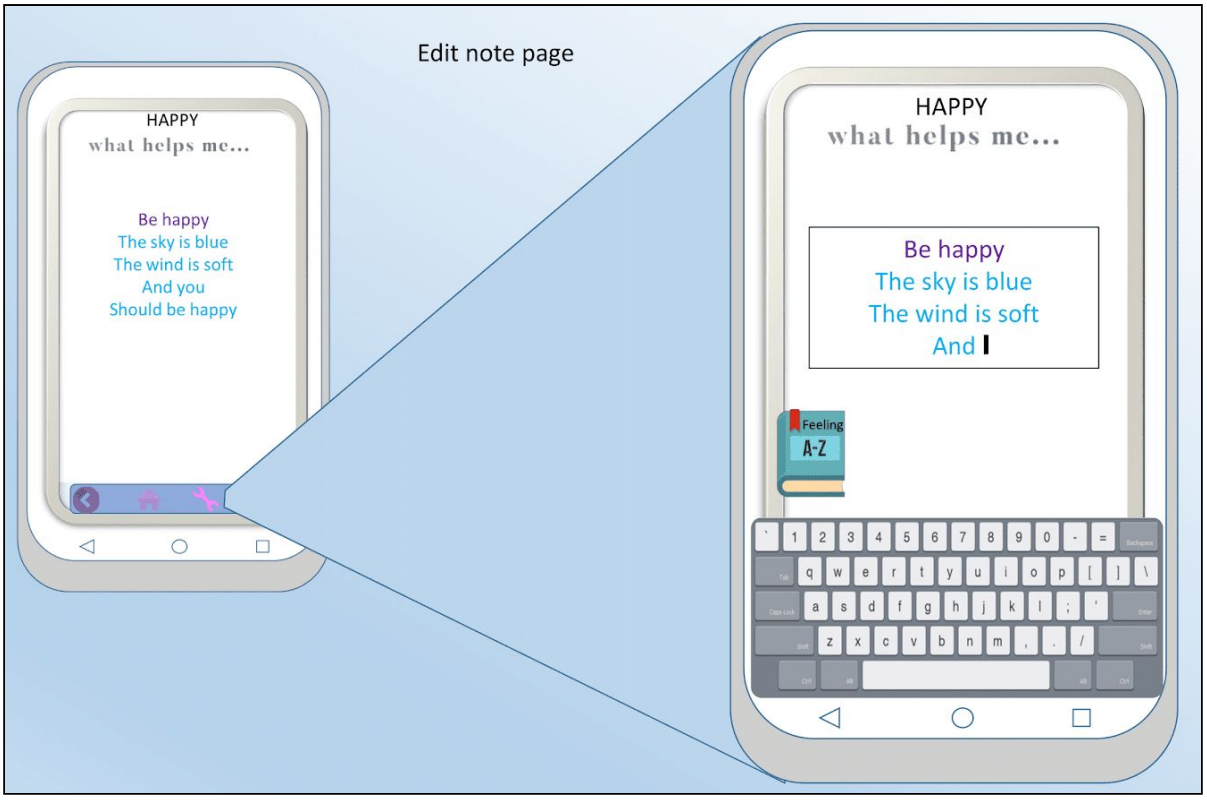

Once a card has been tapped, the user will be presented with the back of a card and the options to enter new text, delete the card or duplicate the card.

## 8.8. Prototype 5 - Version 2 (also see appendix 8.5)

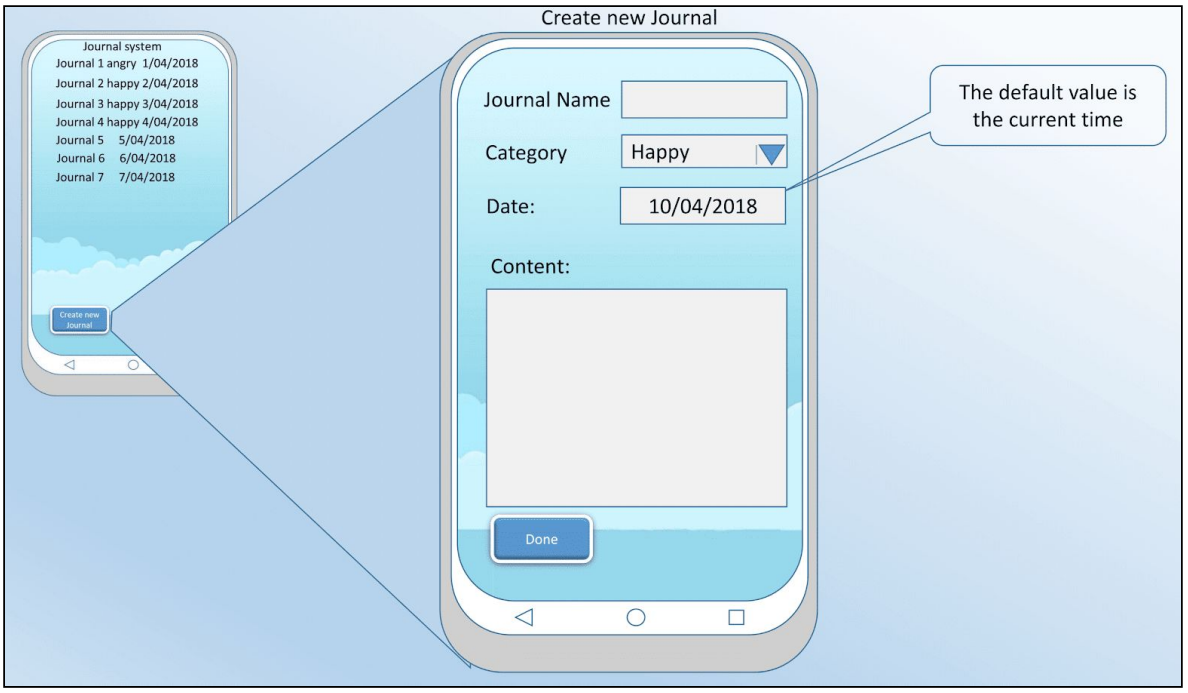
(Home page) - Wellbeing cards:



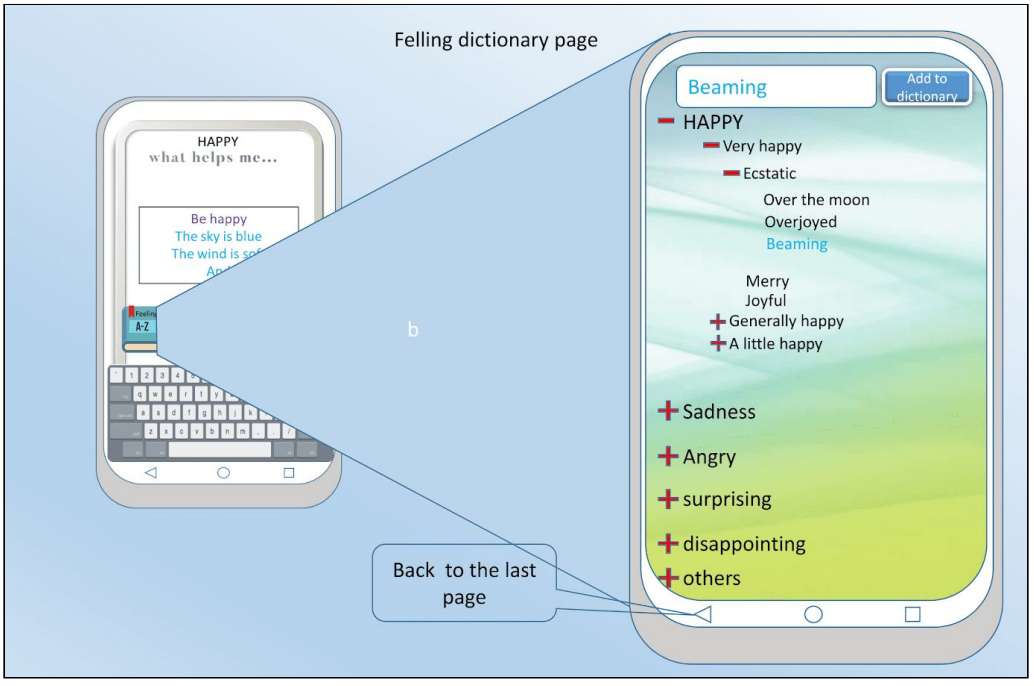

Wellbeing cards (editing):



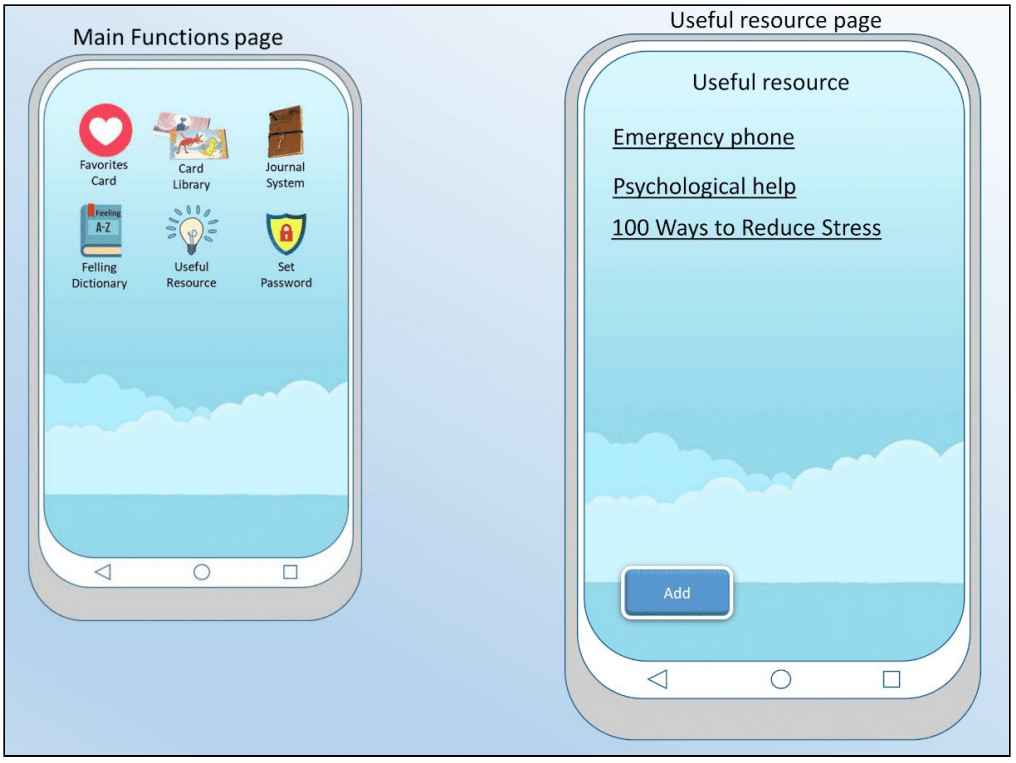
## Journaling system:




## Tree of words:



## Helpful resources:



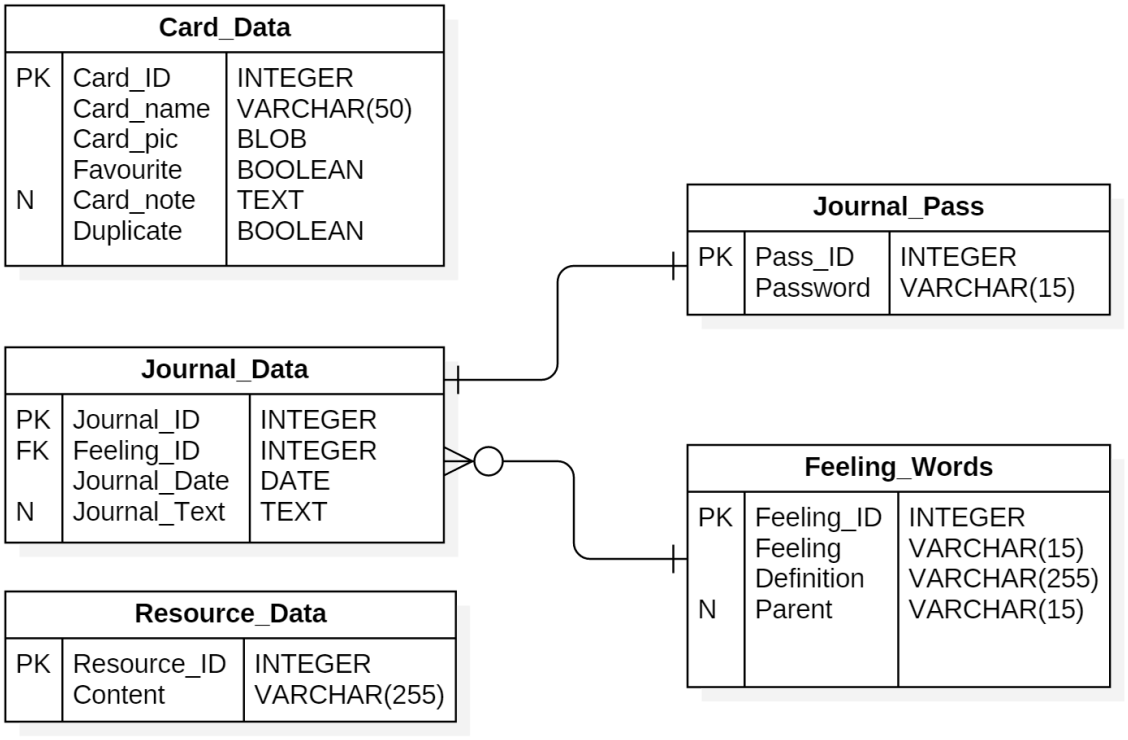
# 9. Appendix IV: Database Design

## 9.1. Attribute description Table

| Flag | Description |
|---|---|
| PK | Primary Key: States that a column uniquely identifies a row in a table |
| FK | Foreign Key: states that a row in a column is a reference to another unique row in a column of another table. |
| NN | Non-Null: states that a column must contain data (cannot be empty) in a table |
| N | Null: states that a column can be empty in a table |

## 9.2. Database Tables - Local SQLite Database

### 9.2.1. Database Overview



### 9.2.2. Card_Data table:

| Attribute Name | Data Type | Flags | Description |
|---|---|---|---|
| Card_ID | INTEGER | PK,NN | Used to assign a unique identifier to each row. |
| Card_name | VARCHAR(50) | NN | Stores the name of each card. The purpose is to distinguish different cards. |
| Card_pic | BLOB | NN | Stores the card image. |
| Favorite | BOOLEAN | NN | Stores the status of each card. This item is used to distinguish whether the card is a favourite card or not. |
| Card_note | TEXT | N | Stores the notes on the back of the card. The purpose is to make it easier for the app to show and modify notes. |
| Duplicate | BOOLEAN | NN | Stores the status which whether the card is duplicated from existing card. |

**Purpose:** The purpose of this table is to store the card's information and provide this information to the applications card controller class so that it can distributed to the necessary parts of the application. Data stored in this table will be modified through general use of the application; creating new cards or modifying existing ones will require interaction with this table through the database adapter class.

**Justification**: The card entity will be used frequently by various parts of the program, and the card element has a low degree of coupling with other elements of the app. This means the table for the card element follows the principle of high cohesion and low coupling database design.

### 9.2.3. Journal_Data table:

| Attribute Name | Data Type | Flags | Description |
|---|---|---|---|
| Journal_ID | INTEGER | PK, NN | This column is used to assign a unique identifier to each row. |
| Feeling_ID | INTEGER | FK, NN | A foreign key to the Feeling_id column of the Journal_list table. Used to link the session to the Feeling_words. |
| Journal_Date | DATE | NN | Store the date which the journal is created. The purpose is to provide basis for sorting by edit time. |
| Journal_Text | TEXT | N | Stores the content of the journal. The purpose is to record the content of each journal entry. |

**Purpose:** The purpose of this table is to store the journal entries written by the user.

**Justification:** The journaling system needs a table in the database to store all data that will be used to create each journal entry entity.

### 9.2.4. Feeling_Words table:

| Attribute Name | Data Type | Flags | Description |
|---|---|---|---|
| Feeling_ID | INTEGER | PK, NN | This column is used to assign a unique identifier to each row. |
| Feeling | VARCHAR(255) | NN | Stores the word (feeling/emotion) |
| Definition | VARCHAR(255) | NN | Stores statements that explain the feeling. The goal is to help users to choose the right word. |
| Parent | VARCHAR(15) | N | Stores the parent word if there is one. |

**Purpose:** The purpose of this table is to store emotions and their definitions for the dictionary function of the application.

**Justification:** The feeling words will be retrieved from here by the data structure that will hold them. They could be hardcoded into this structure but that would make the expansion of the dictionary at a later date much harder.

### 9.2.5. Journal_Pass table:

| Attribute Name | Data Type | Flags | Description |
|---|---|---|---|
| Pass_ID | INTEGER | PK, NN | Used to assign a unique identifier to each row. |
| Password | VARCHAR(15) | | Stores hashed password. |

**Purpose:** The purpose of this table is to store user-related data.

**Justification:** User information needs to be used in multiple places so a separate list is more suitable.

### 9.2.6. Resource_Data table:

| Attribute Name | Data Type | Flags | Description |
|---|---|---|---|
| Resource_ID | INTEGER | PK, NN | Used to assign a unique identifier to each row. |
| Content | VARCHAR(255) | NN | Stores the content of the resource |

**Purpose:** The purpose of this table is to store useful resources.

**Justification:** This table is needed to generate helpful resource entities which can be expanded on by the user's own input.

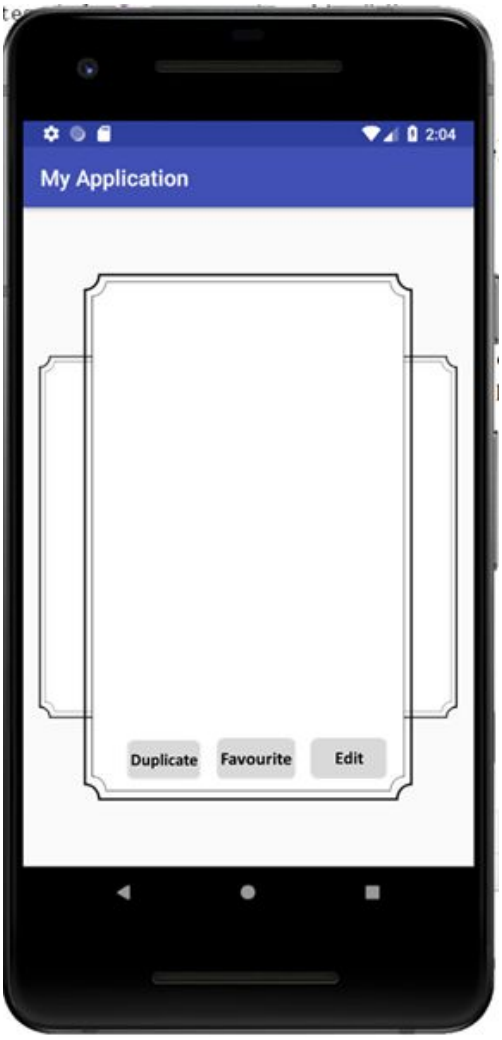
# 10. Appendix V: Example XML Files

## 10.1. Screen: viewCards

Basic cards view will provide the functionality that generate the entire set to cards as well as scrolling through the cards and view the detail of the cards.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <com.gigamole.infinitecycleviewpager.HorizontalInfiniteCycleViewPager
        android:id="@+id/horizontal_cycle"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:icvp_interpolator="@android:anim/accelerate_decelerate_interpolator"
        app:icvp_scroll_duration="250"
        app:icvp_center_page_scale_offset="30dp"
        app:icvp_min_page_scale_offset="5dp"
        app:icvp_max_page_scale="0.8"
        app:icvp_min_page_scale="0.55"
        app:icvp_medium_scaled="false"
    />

</RelativeLayout>
```

```
<Button
    android:id="@+id/duplicate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginStart="50dp"
    android:layout_marginTop="52dp"
    android:text="Duplicate" />

<Button
    android:id="@+id/favourite"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_alignStart="@+id/duplicate"
    android:layout_marginTop="114dp"
    android:text="Favourite" />
<Button
    android:id="@+id/edit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginStart="50dp"
    android:layout_marginTop="52dp"
    android:text="Edit" />
```



## 10.2. Screen: viewJournal

The journal entry view is the window that allows users to view their journals as a list and the xml schema implement this by using the simple arraylist data structure.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/myList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:layout_editor_absoluteX="0dp"
        tools:layout_editor_absoluteY="516dp" />

</android.support.constraint.ConstraintLayout>
```

```
    <Button
        android:id="@+id/edit1"
        android:layout_width="79dp"
        android:layout_height="36dp"
        android:layout_alignParentTop="true"
        android:layout_alignStart="@+id/duplicate"
        android:layout_marginTop="131dp"
        android:text="Edit" />

    <Button
        android:id="@+id/edit2"
        android:layout_width="79dp"
        android:layout_height="35dp"
        android:layout_alignParentEnd="true"
        android:layout_alignParentTop="true"
        android:layout_marginEnd="19dp"
        android:layout_marginTop="27dp"
        android:text="Edit" />

    <Button
        android:id="@+id/edit3"
        android:layout_width="79dp"
        android:layout_height="35dp"
        android:layout_alignEnd="@+id/duplicate"
        android:layout_below="@+id/favourite"
        android:text="Edit" />

    <item
        android:id="@+id/action_settings1"
        android:orderInCategory="100"
        android:title="sort by date"
        app:showAsAction="never" />

    <item
        android:id="@+id/action_settings2"
        android:orderInCategory="100"
        android:title="sort by favourite"
        app:showAsAction="never" />
```

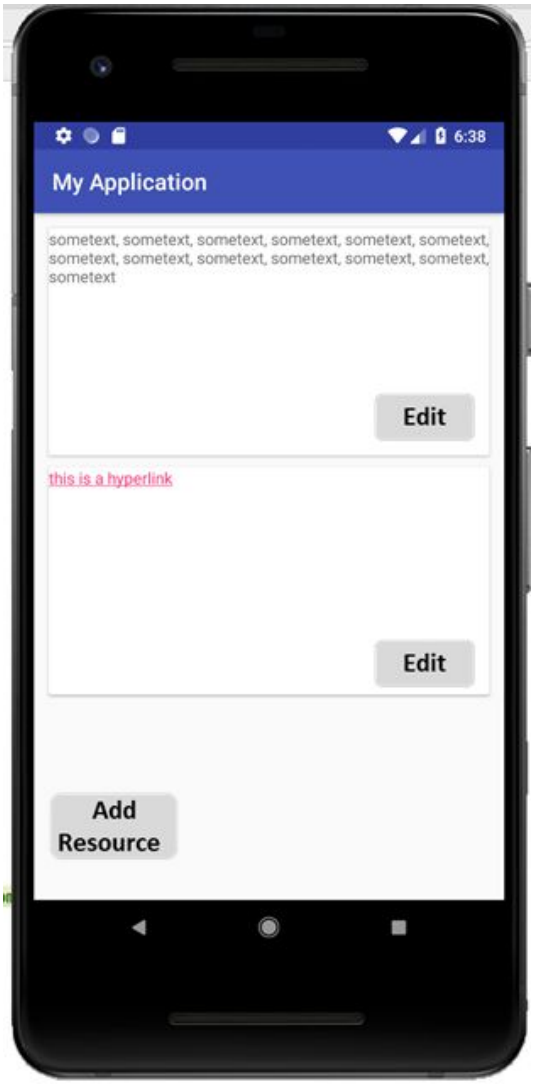

1:33
sources
entry1
entry2
Sort by date
Sort by Key word
entry3
Edit
entry4
Edit
entry5
Edit
Create New

## 10.3. Screen: viewResources (edited)

The resources view is the window that allows users to browse the entire resource information and be able to access them through the hyperlink.

```
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="utas.edu.au.myapplication.MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0">

        <android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:card_view="https://schemas.android.com/apk/res-auto"
            android:id="@+id/content1"
            android:layout_width="match_parent"
            android:layout_height="200dp"
            android:layout_margin="5dp"
            card_view:cardCornerRadius="4dp">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="fill_parent"
                android:orientation="vertical" />

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext" />


        </android.support.v7.widget.CardView>

        <android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:card_view="https://schemas.android.com/apk/res-auto"
            android:id="@+id/content2"
            android:layout_width="match_parent"
            android:layout_height="200dp"
            android:layout_margin="5dp"
            card_view:cardCornerRadius="4dp">>
            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/linktosite"/>

        </android.support.v7.widget.CardView>
```

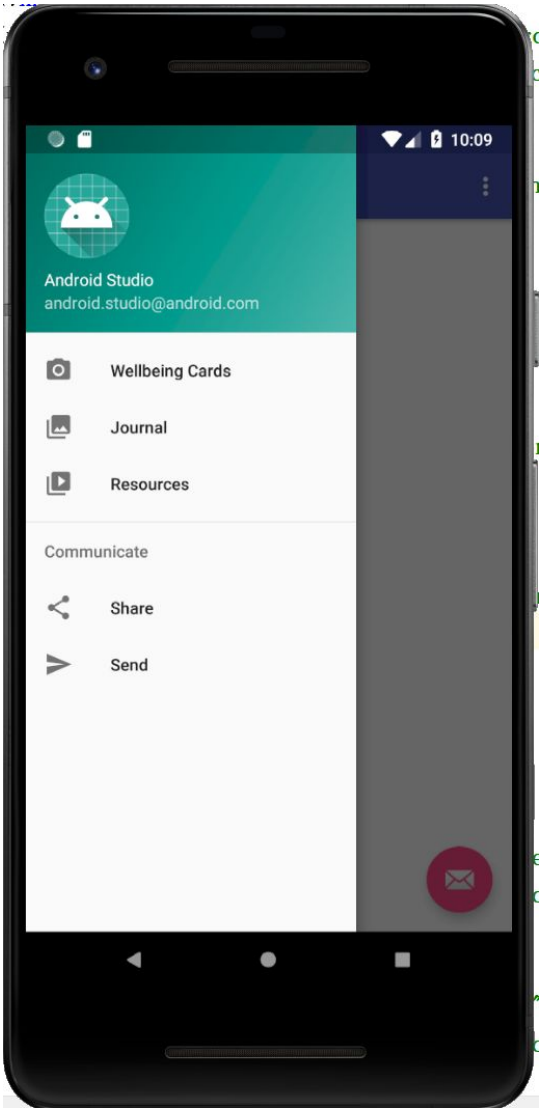
```
            android:layout_width="98dp"
            android:layout_height="wrap_content"
            android:text="Add Resource"
            tools:layout_editor_absoluteX="16dp"
            tools:layout_editor_absoluteY="447dp" />


    </android.support.v7.widget.CardView>

    <android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:card_view="https://schemas.android.com/apk/res-auto"
        android:id="@+id/content2"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_margin="5dp"
        card_view:cardCornerRadius="4dp">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="this is a hyperlink"/>

        <Button
            android:id="@+id/add_resource1"
            android:layout_width="98dp"
            android:layout_height="wrap_content"
            android:text="Edit"
            tools:layout_editor_absoluteX="16dp"
            tools:layout_editor_absoluteY="447dp" />
```


6:38
My Application
sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext, sometext
Edit
this is a hyperlink
Edit
Add Resource

## 10.4. Screen: Menu Sidebar

The main screen sidebar have the navigation system, which allow users to switch view between different views.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/andro
    xmlns:tools="http://schemas.android.com/tools"
    tools:showIn="navigation_view">

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_layout1"
            android:icon="@drawable/ic_menu_camera"
            android:title="Wellbeing Cards" />
        <item
            android:id="@+id/nav_layout2"
            android:icon="@drawable/ic_menu_gallery"
            android:title="Journal" />
        <item
            android:id="@+id/nav_layout3"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="Resources" />
    </group>

    <item android:title="Communicate">
        <menu>
            <item
                android:id="@+id/nav_share"
                android:icon="@drawable/ic_menu_share"
                android:title="Share" />
            <item
                android:id="@+id/nav_send"
                android:icon="@drawable/ic_menu_send"
                android:title="Send" />
```


10:09
Android Studio
android.studio@android.com
Wellbeing Cards
Journal
Resources
Communicate
Share
Send

# 11. Appendix VI: Functional Prototypes

## 11.1. Card Animations (Flipping, Scrolling)

The goal of this functional prototype is to allow users to flip the card between the back and the front as well as scrolling through those cards. This approach uses two external Github libraries.

### 11.1.1. Flipping

*File: build.gradle*

Compile the external library which allows access to the functions needed to 'flip' a card to the view on the other side of it.

```
//Add library
compile 'com.wajahatkarim3.EasyFlipView:EasyFlipView
```

*File: activity_main.xml*

The following shows how to build the layout and set up the constraint for it, which later will add the picture of the card inside the frame. The action function will trigger after performing a touch action inside the frame when it is applied.

```
<com.wajahatkarim3.easyflipview.EasyFlipView
    android:id="@+id/easyFlipView"
    android:layout_centerInParent="true"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:filpOnTouch="true"
    app:flipDuration="500"
    app:flipEnabled="true">
</com.wajahatkarim3.easyflipview.EasyFlipView>
```

Making the image layout be the size of the image while the inputting whatever the size of the image is (both front and the back).

```
<ImageView
    android:gravity="center"
    android:src="@drawable/card_back"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

<ImageView
    android:gravity="center"
    android:src="@drawable/card_front"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

### 11.1.2. Scrolling

*File: MainActivity.java*

Compile the external library which allows access to the functions for creating a Cycle Viewpager.

```
//Add Library
compile 'com.github.devlight:infinitecycleviewpager:1.0.2'
```

Input the source path for the image adapters when the adapter is created, as well as initialize the data.

```
public class MainActivity extends AppCompatActivity {

    List<Integer> lstImages=new ArrayList<>();


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initData();

        HorizontalInfiniteCycleViewPager pager= findViewById(R.id.horizontal_cycle);
        MyAdapter adapter=new MyAdapter(lstImages,getBaseContext());
        pager.setAdapter(adapter);
    }

    private void initData() {
        lstImages.add(R.drawable.pic1);
        lstImages.add(R.drawable.pic2);
        lstImages.add(R.drawable.pic3);
    }
}
```

*File: MyAdapter.java*

Create the adapters for the images and generate each functions to grab an image view.

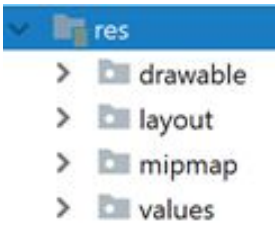
```
public class MyAdapter extends PagerAdapter {

    List<Integer> lstImages;
    Context context;
    LayoutInflater layoutInflater;

    public MyAdapter(List<Integer> lstImages, Context context) {
        this.lstImages = lstImages;
        this.context = context;
        layoutInflater=LayoutInflater.from(context);

    }

    @Override
    public int getCount() {
        return lstImages.size();
    }

    @Override
    public boolean isViewFromObject(@NonNull View view, @NonNull Object object) {
        return view.equals(object);
    }

    @Override
    public void destroyItem(@NonNull ViewGroup container, int position, @NonNull Object object) {
        container.removeView((View)object);
    }

    @Override
    public Object instantiateItem(@NonNull ViewGroup container, int position) {
        View view=layoutInflater.inflate(R.layout.card_item, container, attachToRoot: false);
```

## 11.2. Data structure for Dictionary

Within the dictionary function, they will have the flagged word dictionary.

*Folder:*

- Ø *res*
  - Ø *drawable*
  - Ø *layout*
  - Ø *values*




This is the folder which will store all the styling and formatting of the dictionary. Those folders can be accessed to customise the appearance of the dictionary.

*File: SimpleViewHolder.java*

This file contains a function that generates an empty node which stores the text data.

```
public class SimpleViewHolder extends TreeNode.BaseNodeViewHolder<Object> {

    public SimpleViewHolder(Context context) {
        super(context);
    }

    @Override
    public View createNodeView(TreeNode node, Object value) {
        final TextView tv = new TextView(context);
        tv.setText(String.valueOf(value));
        return tv;
    }

    @Override
    public void toggle(boolean active) {

    }
}
```

*File: TreeNode.java*

This is the function which will generate a space for the text from the dictionary tree. It acts as the children (all the words) of the main parent (categories).

```
public TreeNode addChild(TreeNode childNode) {
    childNode.mParent = this;
    childNode.mId = generateId();
    children.add(childNode);
    return this;
}

public TreeNode addChildren(TreeNode... nodes) {
    for (TreeNode n : nodes) {
        addChild(n);
    }
    return this;
}

public TreeNode addChildren(Collection<TreeNode> nodes) {
    for (TreeNode n : nodes) {
        addChild(n);
    }
    return this;
```

This allows the application to get the level of the dictionary entry so that it can display the node at the correct level in the tree.

```
public int getLevel() {
    int level = 0;
    TreeNode root = this;
    while (root.mParent != null) {
        root = root.mParent;
        level++;
    }
    return level;
}
```

*File: MyHolder.java*

This file creates the frame of each text holder and modifies the size and dimension for each one (both parent and children).

```
public View createNodeView(TreeNode node, IconTreeItem value) {
    final LayoutInflater inflater = LayoutInflater.from(context);
    final View view;
    if (child == DEFAULT) {
        view = inflater.inflate(R.layout.parent, root: null, attachToRoot: false);
    } else {
        view = inflater.inflate(child, root: null, attachToRoot: false);
    }

    if (leftMargin == DEFAULT) {
        leftMargin = getDimens(20dp);
    }
    view.setPadding(leftMargin, getDimens(20dp), getDimens(20dp), getDimens(20dp));

    img = (ImageView) view.findViewById(R.id.image);
    tvValue = (TextView) view.findViewById(R.id.text);
    img.setImageResource(value.icon);
    tvValue.setText(value.text);
    return view;
}
```

This calls the variables from the res folder so that the function can display the buttons icon as well as performing the action after the user taps a button.

```
public void toggle(boolean active) {
    if (toggle)
        img.setImageResource(active ? R.drawable.ic_arrow_drop_up : R.drawable.ic_arrow_drop_down);
}

public static class IconTreeItem {
    public int icon;
    public String text;

    public IconTreeItem(int icon, String text) {
        this.icon = icon;
        this.text = text;
    }
}

private int getDimens(int resId) {
    return (int) (context.getResources().getDimension(resId) / context.getResources().getDisplayMetrics().density);
}
```

## 11.3. Data storage (user data)

The application will use SQLite for storage of data necessary to create entities. This variety of SQL creates a local database without any need for a server connection while still providing the functionality of a regular MySQL database.

*File: PropertyDatabase.java*
This function is provided to create the database as well as the ability and allow it to be updated. It also opens the connection to the SQLite database in memory. Additionally, it generates a list of arrays for setting up the columns of the table. The data type ArrayList gives the ability to select the data in the database.

```
PropertyDatabase databaseConnection = new PropertyDatabase(this);
final SQLiteDatabase db = databaseConnection.open();
```

```
private static final String TAG = "PropertyDatabase";
private static final String DATABASE_NAME = " PropertyDatabase";
private static final int DATABASE_VERSION = 1;
private SQLiteDatabase mDb;
private DatabaseHelper mDbHelper;
private final Context mCtx;
public PropertyDatabase(Context ctx) { this.mCtx = ctx; }

private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        Log.d(TAG, "DatabaseHelper onCreate");
        db.execSQL(PropertyTable.CREATE_STATEMENT);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.d(TAG, "DatabaseHelper onUpgrade");
        db.execSQL("DROP TABLE IF EXISTS " + PropertyTable.TABLE_NAME);
        onCreate(db);
    }
}

public static ArrayList<Property> selectAll(SQLiteDatabase db) {
    ArrayList<Property> results = new ArrayList<Property>();
    Cursor c = db.query(TABLE_NAME, null, null, null, null, null, null);
    return results;
}

mDbHelper = new DatabaseHelper(mCtx);
mDb = mDbHelper.getWritableDatabase();
return mDb;
```

*File: PropertyTable.java*

This class allows the program to create entities from the database such as a Card in this example, which uses name, date and comment fields to generate a card. It also handles retrieving data that needs to be stored in the database.

```
public static final String TABLE_NAME = "property";
public static final String KEY_CARD_ID = "CARD_ID";
public static final String KEY_NAME = "NAME";
public static final String KEY_DATE = "DATE";
public static final String KEY_INFO = "INFOMATION";

public static Property createFromCursor(Cursor c) {
    if (c == null || c.isAfterLast() || c.isBeforeFirst()) {
        return null;
    }
    else {
        Property p = new Property();
        p.setCARD_ID(c.getInt(c.getColumnIndex(KEY_PROPERTY_ID)));
        p.setNAME(c.getString(c.getColumnIndex(KEY_NAME)));
        p.setDATE(c.getInt(c.getColumnIndex(KEY_DATE)));
        p.setINFO(c.getString(c.getColumnIndex(KEY_INFO)));

        return p;
    }
}

public static final String CREATE_STATEMENT = "CREATE TABLE "
    + TABLE_NAME
    + " (" + KEY_PROPERTY_ID + " integer primary key autoincrement, "
    + KEY_NAME + " string not null, "
    + KEY_DATE + " int not null, "
    + KEY_INFO + " string not null " +");";

public static void insert(SQLiteDatabase db, Property p) {
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, p.getNAME());
    values.put(KEY_DATE, p.getDATA());
    values.put(KEY_INFO, p.getINFO());
    db.insert(TABLE_NAME, null, values);
}
```

*File: ListActivity.java*

This used the resulting database handle to create a table, which allow inserting data into the table.

```
public void onCreate(SQLiteDatabase db) {
    Log.d(TAG, "Create");
    db.execSQL(PropertyTable.CREATE_STATEMENT);
    Property card1 = new Property();
    card1.setNAME("user1");
    card1.setDATE(01022018);
    card1.setINFO("some information");
    Property card2 = new Property();
    card2.setNAME("user2");
    card2.setDATE(01022018);
    card2.setINFO("some information");
    PropertyTable.insert(db, card1);
    PropertyTable.insert(db, card2);
    ArrayList<Property> properties = PropertyTable.selectAll(db);
}
```

The result of adding the adapter type, which acts as the Controller between the data (Model), and the ListView (View), allows the listing of the data from the database and display data on screen.

```
public class PropertyAdapter extends ArrayAdapter<Property>
private int mLayoutResourceID;

public PropertyAdapter(Context context, int resource, List<Property> objects) {
    super(context, resource, objects);
    this.mLayoutResourceID = resource;
}

@NonNull
@Override
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
    return super.getView(position, convertView, parent);

    LayoutInflater layoutInflater = (LayoutInflater)getContext().getSystemService(Service.LAYOUT_INFLATER_SERVICE);
    View row = layoutInflater.inflate(mLayoutResourceID, parent, false);
    TextView textView = row.findViewById(android.R.id.text1);
    Property p = this.getItem(position);
    textView.setText(p.getNAME()+": "+p.getNAME());

    return row;
}
```

# 12. Appendix VII: Glossary

| Term | Description |
|---|---|
| **XML** | eXtensible Markup Language |
| **Schema** | XML Schema is a language for expressing constraints about XML documents |
| **SQLite** | SQLite is an embedded SQL database engine |
| **Dependency** | The way Android studio includes external binaries in a project. Can be a local reference or a remote server. |
| **GUI** | Graphical User Interface |
| **CSV** | Comma Separated Values. A common format for storing data in a text file. |
| **Java** | An object-oriented programming language commonly used for Android development. |

# 13. Appendix VIII: References

*Android Studio Tutorial - Easy Flip View*, *published by EDMT Dev, 16/01/2017, viewed 23/04/2018,* <https://www.youtube.com/watch?v=xacptJxva4s>

*Android Studio Tutorial - Infinite Cycle View Pager*, *published by EDMT Dev, 29/04/2017, viewed 23/04/2018,* <https://www.youtube.com/watch?v=4ct0oPf_u2o&t=142s>

*Android Studio Tutorial - Draggable Tree View*, *published by EDMT Dev, 01/07/2017, viewed 23/04/2018,* <https://www.youtube.com/watch?v=v78ZCKZihyA&t=39s>

*App Manifest Overview, Android Developers, Last updated April 23, 2018, viewed 29/04/2018,* <https://developer.android.com/guide/topics/manifest/manifest-intro>

*Animations and Transitions, Android Developers, Last updated April 17, 2018, viewed 29/04/2018,* <https://developer.android.com/training/animation/>

*Infinite Cycle View Pager, Devlight, Last updated August 23, 2017, viewed 29/04/2018,* <https://github.com/Devlight/InfiniteCycleViewPager>

*Easy Flip View, wajahatkarim3, Last updated December 21, 2017, viewed 29/04/2018,* <https://github.com/wajahatkarim3/EasyFlipView>

*Hierarchical Data in SQL, Worthy, D., published August 4, 2014, viewed 29/04/2018,* <https://blog.duncanworthy.me/sql/hierarchical-data-pt1-adjacency-list/>