Mandatory assignment 02

Dennis Thinh Tan Nguyen

April 25, 2016

Contents

1 Important Backend design choices

During the design and development of the final version of Tingle, many design choices have been made to improve but also extend the backend of the application. Some of the most important choices are described in the following subsections.

1.1 MVC architecture

Tingle is utilizing the MVC architectural design pattern to separate the system into three interconnected parts. This was previously introduced in Tingle V2 with a simple implementation of the MVC design pattern (see figure 1 for class diagram).

As for the final version Tingle with the introduction of various of new features such as networking, SQLite database storage and search/sorting the MVC pattern enabled one to extend the model, controller or view in a clear way and keep complexity within each of their respected set of responsibility (see figure 3 for class diagram of the final version of Tingle). The appendix will also contain class diagrams of previous versions of Tingle that can be used for comparison.

1.2 Repository pattern and storage

To enable future extensions of the repository, a generic repository pattern has been adopted. The previous hardcoded thingsDB (TingleV2-3) in TingleActivity made it impracticable to extend or even replace a given storage solution. Consequently, in Tingle V4, the following components have been introduced a:

- ThingsDatabase
- IRepository
- InMemoryRepository
- Entity

As for the final version of Tingle, the InMemoryRepository, which stored items in memory, has been replaced by an SQLite repository. The SQLite repository enables the application to store user-entered "things" persistently. The current database consists of a single table that contains all currently stored things. They are uniquely identified by a random generated UUID when they are created and added to the database.

One may argue that a single table is an inferior solution which it is in many cases, but the scale of the current system is so small that I have preferred to keep it simple and focus on other aspects. A solution of this would be creating relationship schemas, E/R diagrams and then define a normalized database for the items based on that.

An alternative solution for persistent data storage is to store the data on the cloud and thus, the data is not coupled to a single device but can be accessed from other devices as well. For the sake of scope and simplicity, this has been omitted.

1.2.1 IDs for Things/entities

Another abstract class called Entity serves as a superclass for the Thing Class. The Entity class contains an ID which is used to uniquely identify a Thing, reduce future code duplication but also separate application logic from storage logic. A repository can take classes that are derived from Entity and store them.

1.2.2 TempThingToStore

It is worth to mention a simple but important design choice, the implementation of TempThingToStore. This class is used when a user has made some changes for a given existing thing. Instead of overwriting settings instantly, this class provides a way to store changed settings before they are committed to the database. The user can cancel her changes in case of undesired made changes. If the user wishes to accept the changes, information that is stored within an instance of TempThingToStore is then committed to the database. TempThingToStore is also used to retain changes when the user switches between landscape and portrait mode.

1.3 Dependency injection

Since the introduction of Dagger2 Framework[1] back in Tingle V4, the extension of an SQLite database for persistent storage was simple to implement and extend. One was just required to create another IRepository and implement all the code that was required to run SQlite. Finally, one can just feed the new repository to Dagger2 and thus the new repository will be injected to ThingsDatabase.

Dagger2 was previously only used to inject repositories, but this has been extended and as such Dagger2 will also provide and inject the NetworkHandler and SearchHandler as well. The most interesting of these two classes is the new SearchHandler class which will be introduced later in section 1.4, but as for dependency injection, one can change what type of sort algorithm to use as SearchHandler takes an interface of "ISort" and therefore sorting is also modular in the same sense as ThingsDatabase with its repositories.

[1] Dagger2 website: http://google.github.io/dagger/

1.3.1 TingleApplication class

The Application class has been extended in a new class called TingleApplication. This design choice has been made considering our usage of dependency injection has been expanded. As is, one need to store the new SearchHandler and NetworkHandler somewhere but also inject ThingsDatabase with a context that will only be provided by the application class. All this requires initialization and that is where TingleApplication comes into practice. This class will initialize Dagger2, store the handlers and then provide means to access all these handlers.

By storing the handlers in TingleApplication the lifetime of them will likewise be the same as the application itself. Previously, this initialization was done in ThingsDatabase, as the application only used dependency injection to inject repositories and hence it was also reasonable to place it there. However, this does not make sense any longer as of the introduction to the new handlers. And so, this responsibility had to be refactored into another abstraction.

1.4 SearchHandler

The newly introduced SearchHandler in the final version of Tingle is responsible for searching and sorting.

1.4.1 Searching

Previously, the user could search for a given thing based on "what" the item was, In other words, the name of the item. The feature would then return a toast of its location. This simple solution did its task correctly but had its flaws. The user was unable to do anything with the search result, as it was only a toast telling the user a given thing's location. The user was moreover unable to search for items based on other factors such as locations. However, the greatest problem occurred when the user had multiple items with same variables such as the name of the thing or its location. The first item it found was returned regardless of other items with same variables.

These issues have been addressed with SearchHandler which allows a user to search based on what or where and the returned list would contain all items that matched a given input. The search feature did not require an exact match as it would only compare the inputted value with and item and if there were some match, the item would be taken.

By way of example consider the following search example with generic items and a sequence of inputs:

- itemA : name = a
- itemB : name = ab
- itemC : name = c

—

- Inputted value = a
- returned items:
- itemA : name = a
- itemB : name = ab

—

- Inputted value = ab
- returned items:
- itemB : name = ab

And so, one does not have to enter the whole name or location of a given item. This feature provides a way for the user to search for exact items but also items that are common in variables. "What" searching is selected by default can be changed within the menu on the action bar.

1.4.2 Sorting

Another main feature of SearchHandler is sorting. The user can sort a list of things based on three variables:

- Name of items (what)
- Location of items (Where)
- Date added of items (Date)

This feature provides the user an overview based on what is demanded. The utilized sorting algorithm is selection sort although this can easily be replaced as described in section 1.3. The reason for choosing selection sort was not because it is the most efficient or fastest algorithm considering its best case runtime is N to the power of 2 compared to other algorithms such as mergesort that have a runtime of n log n. Selection sort was selected due to its simple implementation compared to other algorithms and considering the scale of the application as well as the number of items a user may actually store, this algorithm may be sufficient enough. It is very unlikely that a user may store billions of items. In the case of aforementioned, one may prefer other sort algorithms such as Mergesort or quicksort.

1.5 NetworkHandler

The newly introduced NetworkHandler in the final version of Tingle is responsible for handling network related tasks as well as looking up items based on a scanned barcode at https://www.outpan.com/. It must be noted that NetworkHandler is not doing the fetching of an item itself, but propagate the work to another class called "OutpanFetcher" which is responsible for connecting to Outpan.com and retrieve information based on the scanned value.

NetworkHandler will, on the other hand, check if the current device has an active internet connection. This check is done in two parts. The NetworkHandler will first check if the user is connected to a given network. This could be WiFi, LTe, 3g etc. This check does not indicate that a given device is actually connected to the internet and therefore another check is executed. The second check will essentially connect to a given well-known website such as www.google.com. Based on a defined timeout timer, if a response is delivered one can assume that the device is connected to a network that has access to the internet.

The method used in the second check is also utilized when a connection request is made to outpan.com . The reason is to cope with issues that could be related to outpan.com being unavailable or sudden loss of connection. Every network related operations such as fetching are executed in another thread to avoid blocking the Main thread or the UI thread.

2 User interface

The final version Tingle consists of the following pages:

- MainPage:

  Portrait mode can be seen on figure 5 in the appendix.

  Landscape: Landscape mode can be seen on figure 7 in the appendix.

- ListPage

  Portrait mode can be seen on figure 6 in the appendix.

  Landscape: Landscape mode can be seen on figure 7 in the appendix.

- DetailedThingPage

  Portrait mode can be seen on figure 10 in the appendix.

  Landscape mode can be seen on figure 11 in the appendix.

The MainPage allows the user to see the last added item and it is also here the user can add new items based on user entered data or by scanning a barcode. ListPage allows the user to get an overview of every added item. The user can search, sort or delete items on this page. The last page is the DetailedThingPage. This page allows the user to add, modify or see supplementary information of an existing Thing. The user can also share the selected thing to another person from this page.

A complete manual is added to the appendix section 6.1.

2.1 Usability design choices

A couple of design choices has been made to improve the usability of Tingle. Every button, text boxes, and interactive widgets have been, at best, placed at the bottom of the screen so the user can comfortably reach them. However, as for Google's own design guidelines searching, back button and menu have been added to the action bar.

It has also been decided that Tingle should have a standardized dual screen landscape mode. That means whenever the user flips onto landscape mode the App would show two fragments. For an example, The MainPage and ListPage will be shown together if the user within one of these pages and is in landscape mode. For DetailedThingPage, a simple list of items will be shown when in landscape mode.

This simple list does not contain search, delete or sort features but is only there to let the user select other items and to provide a general overview of items. This design choice is made based on the fact that there is more horizontal space for an additional fragment that can be used to provide some sort overview but also the fact that most phones have a screen size of at least 4.7

inches. One could argue that this limits the usability for phones with smaller screens which are true, but this can be fixed by creating checks based on screen size of the current device and only enable it for phones with large screens. This check has not been implemented in the final version of Tingle as it was deemed unnecessary.

3 Functionality extensions compared to first mandatory handin

The following functionality has been added

- Adding items via barcode. Item information is fetched from Outpan.com based on the scanned barcode
- Multi-selection/deletion of items
- Search feature based on what or where of items
- Sort feature based on what, where or date of items (Only in Ascending order)
- Detailed page of a single item (Displays all related information of a given item)
- Date picker/changer for a selected item
- Binding a picture to an item
- SQLite database for persistence data storage
- Sharing feature that allows user to share a given item via her social media/messaging apps (Facebook,SMS etc)

4 Test

4.1 Unit testing of model

Automated unit tests have been performed on the model of Tingle. That is, testing the repository and database functionalities as well as search and sort functionality. Mockit framework has been used to create mocks when testing ThingsDatabase and SearchHandler (See figure 4 in the appendix for test results of the final version of Tingle).

4.2 Manual testing of View and application

Taken the current state and complexity of the final version of Tingle, system tests and integration tests have been performed manually. However, one may want to use Android instrumentation tests to perform automated UI tests when the complexity of Tingle increases.

4.2.1 Sequence of tests

Below are five samples of utilized test sequences when one are to perform manual testing of the application

Register item with barcode

- Start application in portrait mode.
- Enter the location of an item in 'where' text field.
- Press "Add thing with barcode".
- Confirm (if existing) scan application has started up and then scan the item.

  If no scan app exists, confirm that button is disabled. End Test. – Result: OKAY

  If no there is no internet connection a toast must appear. indicating no connection when pressing "add thing with barcode button". End Test. – Result: OKAY
- Confirm scanned information based on the updated "last thing added" text view and also on the list page.
- End Test. – Result: OKAY.

Multi selection and deletion

- Start application in portrait mode.
- Add three or more items.
- Go to list page by pressing "Show all" button from the main page.

- Confirm items added (Number of items and their information).
- Select two or more items but leave one item behind by pressing the checkboxes.
- Extend the menu on the action bar and press "Delete".
- Confirm items deleted(Items that were checked are gone and the unchecked item is still shown in the list).
- Exit app or go to another activity/fragment.
- Return to the list page and confirm that the deletion was persistent.
- End Test. – Result: OKAY.

Searching based on where

- Start application in portrait mode.
- Add three or more items where two or more items are in some sense lexicographically similar (eg: a, ab abc or ba, bac, bak, baccla).
- Go to list page by pressing "Show all" button from the main page.
- Confirm items added (Number of items and their information).
- Confirm search type is set to what .
- Enter first char into the search field (eg: 'a').
- Confirm the results. It should display every existing item that starts with the letter 'a'. and hide all items that do not.

  Enter same char with uppercase. The result must be the same.

  Enter additional chars (eg: 'ab') and confirm if the result set has been refined based on the new search string.
- Delete the entered search string and confirm that all existing items have been re-added to the list.
- End test. – Result: OKAY.

Sorting

- Start application in portrait mode.
- Add five or more items.
- Go to list page by pressing "Show all" button from the main page.
- Confirm items added (Number of items and their information).

- Select "Sort what" in the menu on the action bar.

  Confirm all items has been sorted based on "what" in ascending order.

- Select "Sort where" in the menu on the action bar.

  Confirm all items has been sorted based on "where" in ascending order.

- Select "Sort date" in the menu on the action bar.

  Confirm all items has been sorted based on "date" in ascending order.

- End test. – Result: OKAY.

Modify existing item and commit changes

- Start application in portrait mode.
- Add one.
- Go to list page by pressing "Show all" button from the main page.
- Confirm item added.
- press on the item to go to its detailed page.
- Confirm item information (What and where).
- Modify every fields and option.
- Commit changes by pressing the save button on the action bar.
- Exit application (App state must be destroyed).
- Start the application and confirm changes has been stored persistently.

  Main page "last added item" shows the modified what and where.

  List page shows the modified what and where.

  Detailed page shows the modified what, where and all other. changes.

- End test. – Result: OKAY.

5 Bugs

So far no app breaking bugs can be found in the final version of Tingle. Sorting does very rarely but randomly seem to stop working and requires the user to restart the application. I have tried to track it down, but it seems to be difficult to replicate the sequence such that the bug will happen.

6 Appendix

6.1 Manual of Tingle

Accessing ListPage
To get to the listPage in portrait mode, one can press the "show all" button or flip the screen to go to landscape mode.

Accessing DetailedThingPage
To get to the DetailedThingPage press the "show all" button or flip the screen to go to landscape mode to access the ListPage. Press on an existing item (Not the checkbox) to get to the DetailedThingPage

Adding items - Manual way
To add an item one can enter the name and location of the item in the two edit boxes and press the "Add thing" button. The recently added header will then be updated with the newly added. item.

Adding items - Barcode way
To add an item with a barcode one must enter the location of the item in the "what" edit boxes and press the "Add thing with barcode" button. A scanning application will be started and a barcode can be scanned here. When scanned, one will be returned to Tingle and the recently added header will then be updated with the newly added. item. *Notice that the device with Tingle installed must contain a third-party scanner application. If non-such application exists, this feature will be disabled by default.*

Search for item
To search for a stored item, go to the ListPage by pressing on the "Show all" button. Press the magnifying glass button and write the desired search string (not case sensitive). By default, searching will be based on "what" but this can be changed to "where" by pressing the menu button at the top right corner of the screen and thereafter select "Search for where".

Sorting ListPage
To sort stored items in List page, go to the ListPage by pressing on the "Show all" button.Press the menu button at the top right corner of the screen and thereafter select the desired sorting method.

Delete items
To delete a stored item, go to the ListPage by pressing on the "Show all" button. Press the checkbox for the item to be deleted. Press menu button at the top right corner of the screen and select "Delete". One can also delete multiple items concurrently by checking additional checkboxes.

Modify existing items
To modify a stored item, press on an existing item in the list page. Do your modification and press the "save" button on the top right corner of the screen. (Represented as a Square disk)

Sharing existing items
To share a stored item, press on an existing item in the list page. Press the "share" button on the top right corner of the screen. Select your messaging app and you will be redirected to that desired social/messaging application.

Adding pictures of existing items
To add a picture to a stored item, press on an existing item in the list page. Press the "camera" button. You will be redirected to your camera application (Or prompted if you have many). The picture that was taken can be seen above the "camera" button.

6.2 Class Diagrams

Class Diagrams have also been attached as separate files.

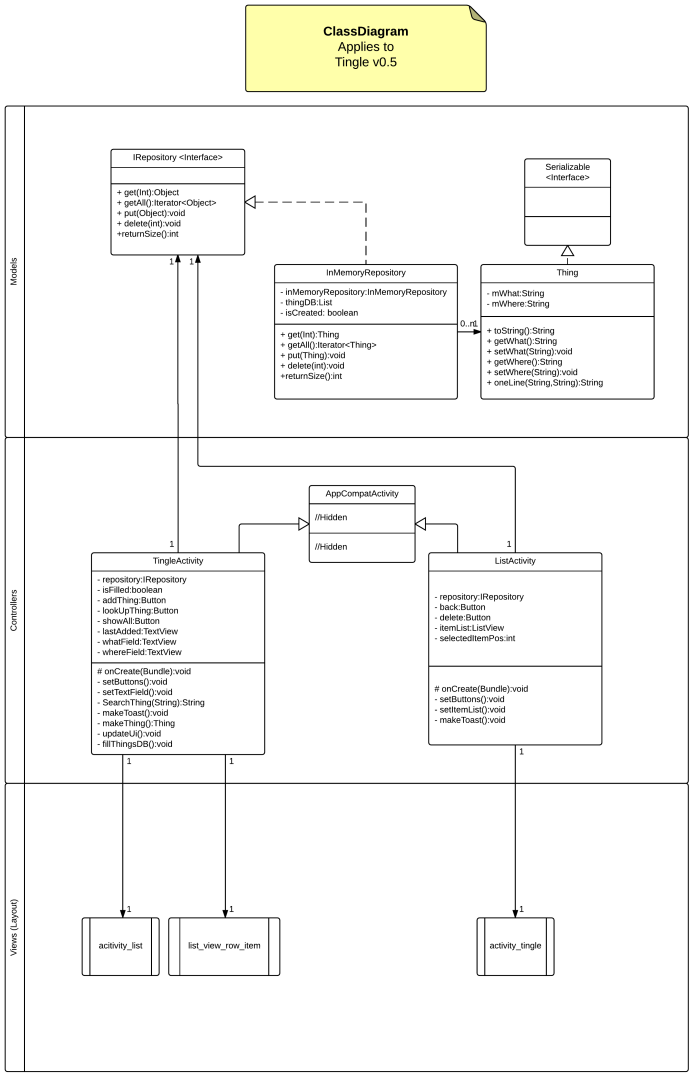

Figure 1: Class Diagram of Tingle V2

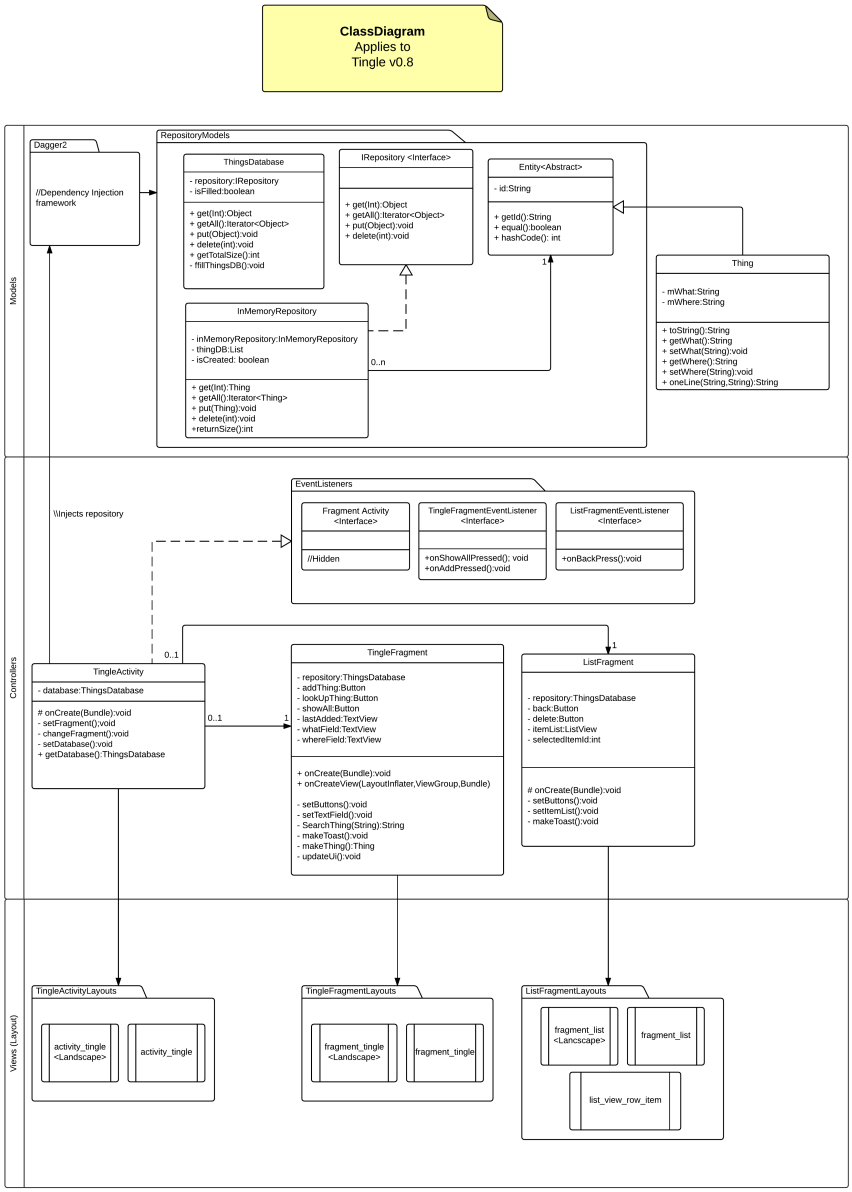

Figure 2: Class Diagram of Tingle V4

6.3 Test Results

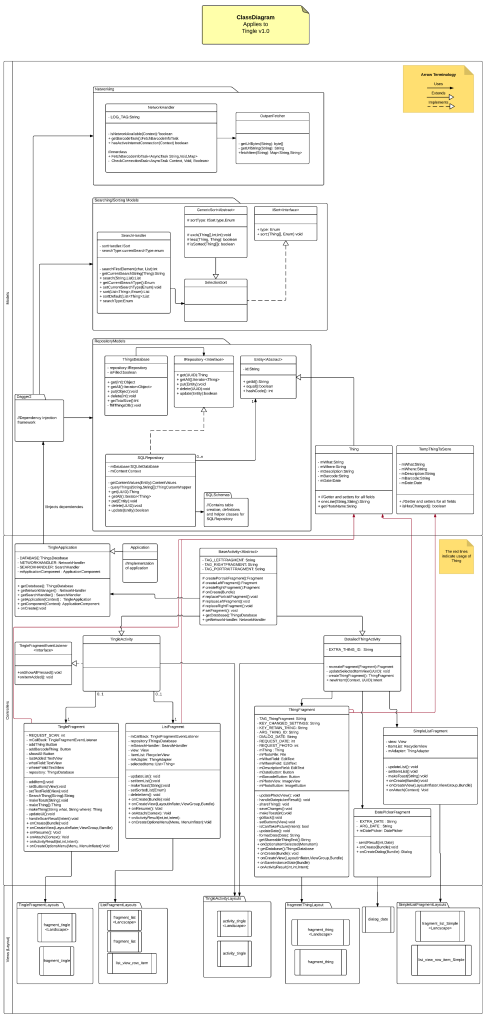

Figure 3: Class Diagram of Tingle V1.0

6.4 UI pictures

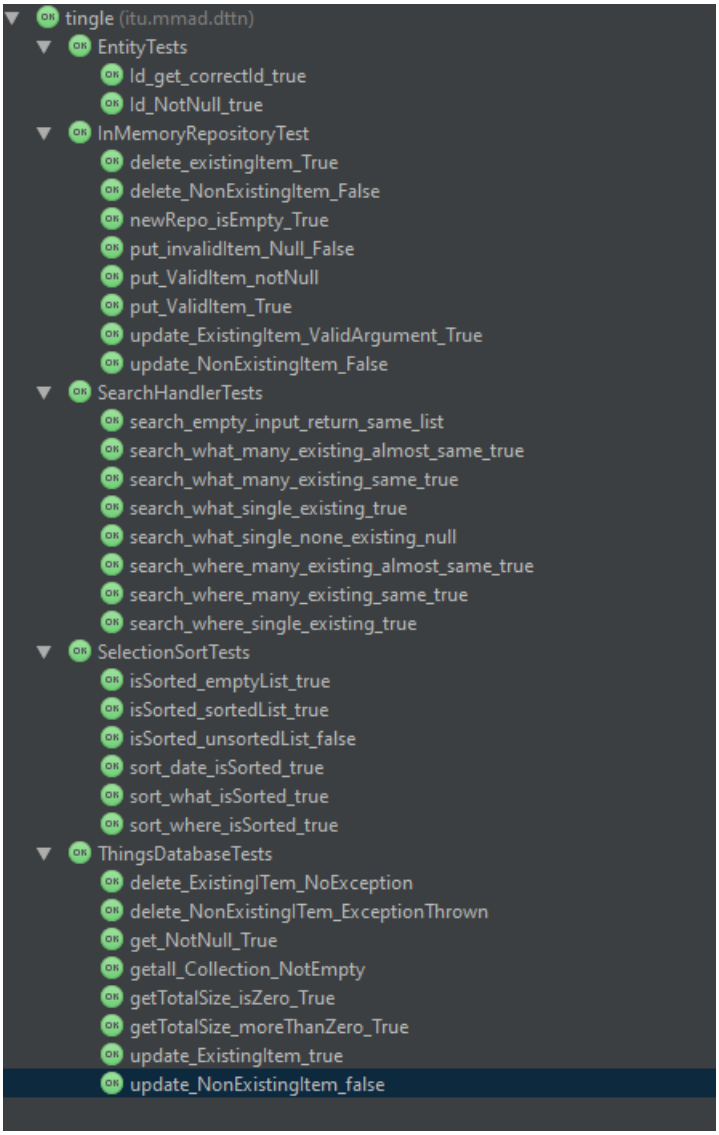

Figure 4: Unit test results of Tingle V4

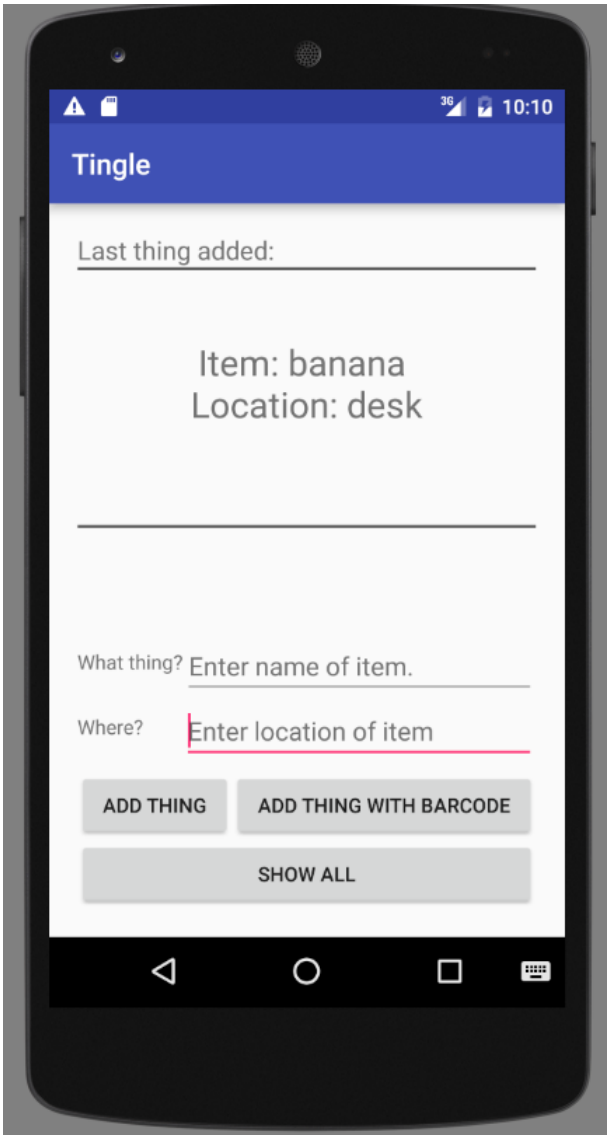

Figure 5: Main page in portrait mode

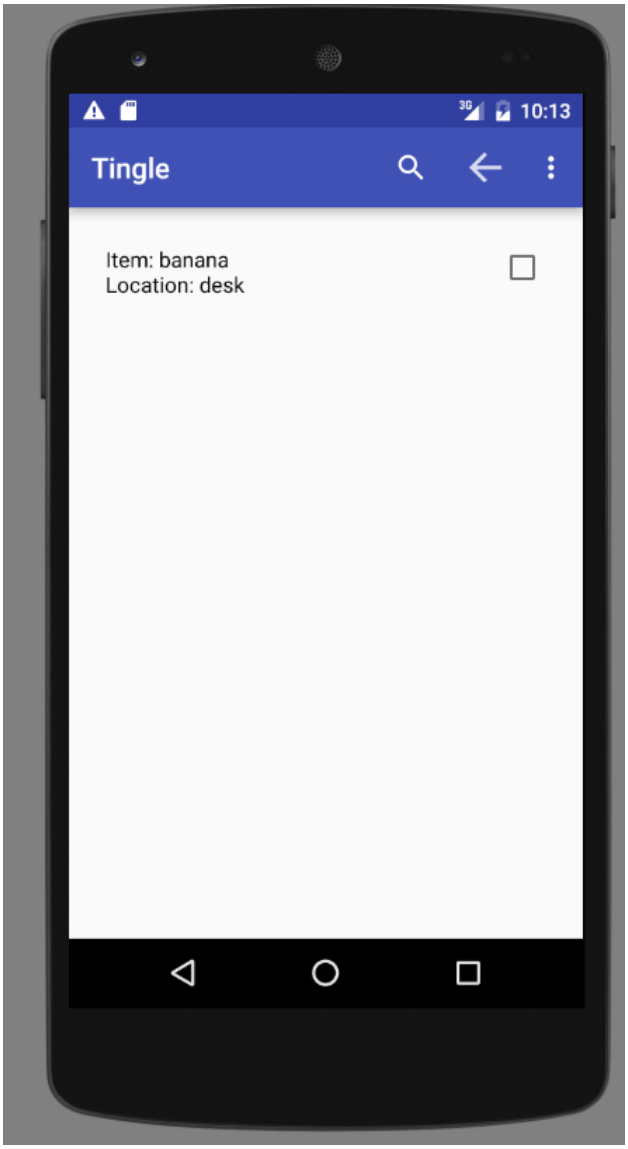

Figure 6: List page in portrait mode

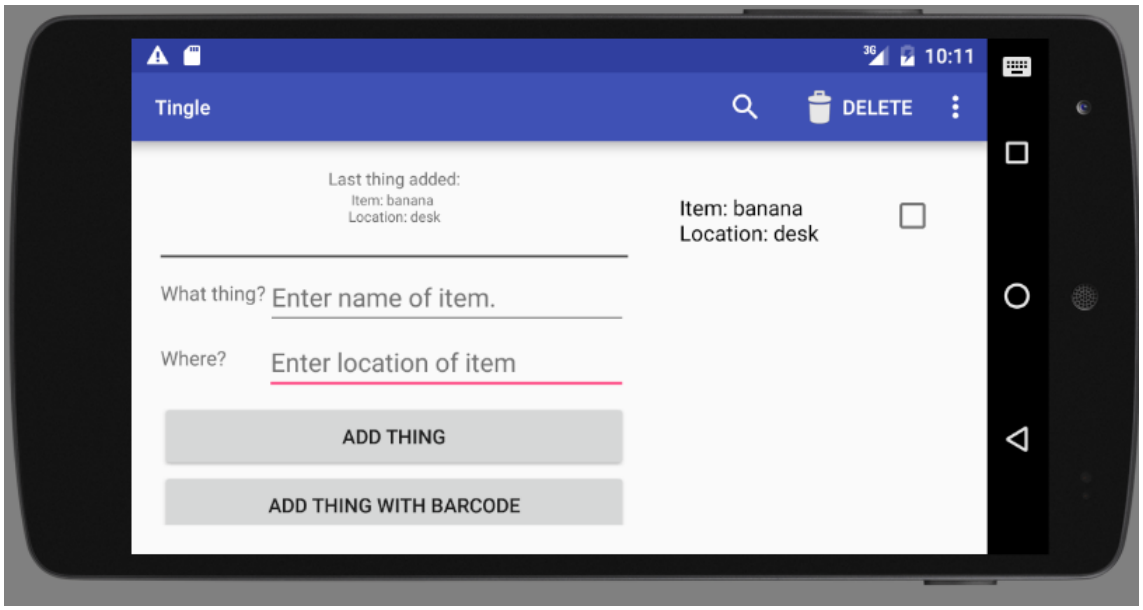

Figure 7: Landscape with MainPage and ListPage mode

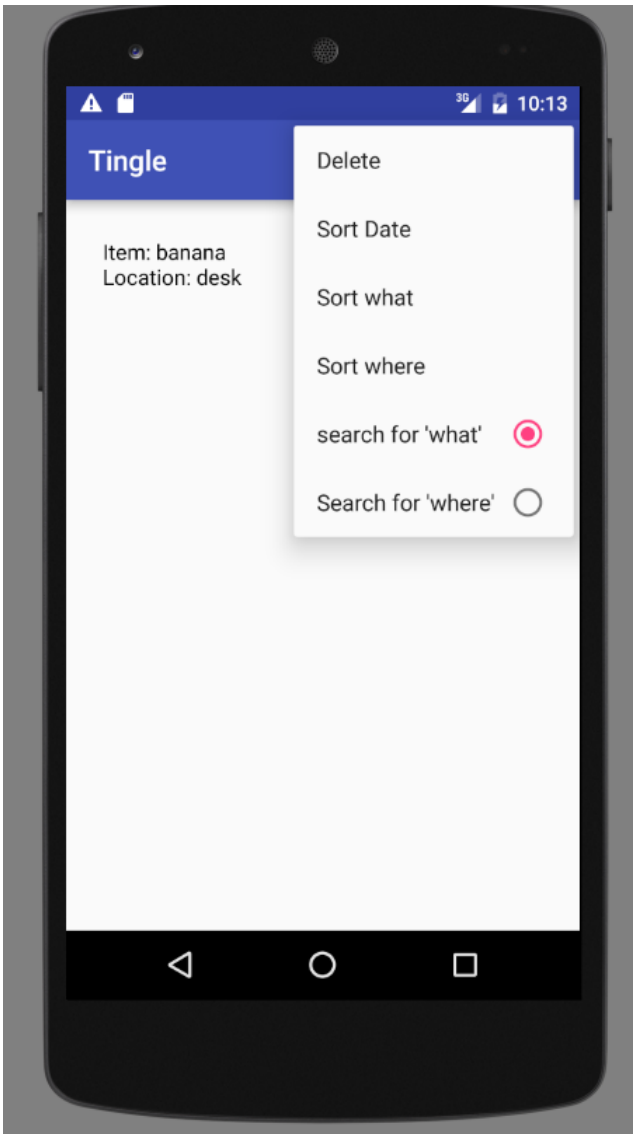

Figure 8: Menu in ListPage

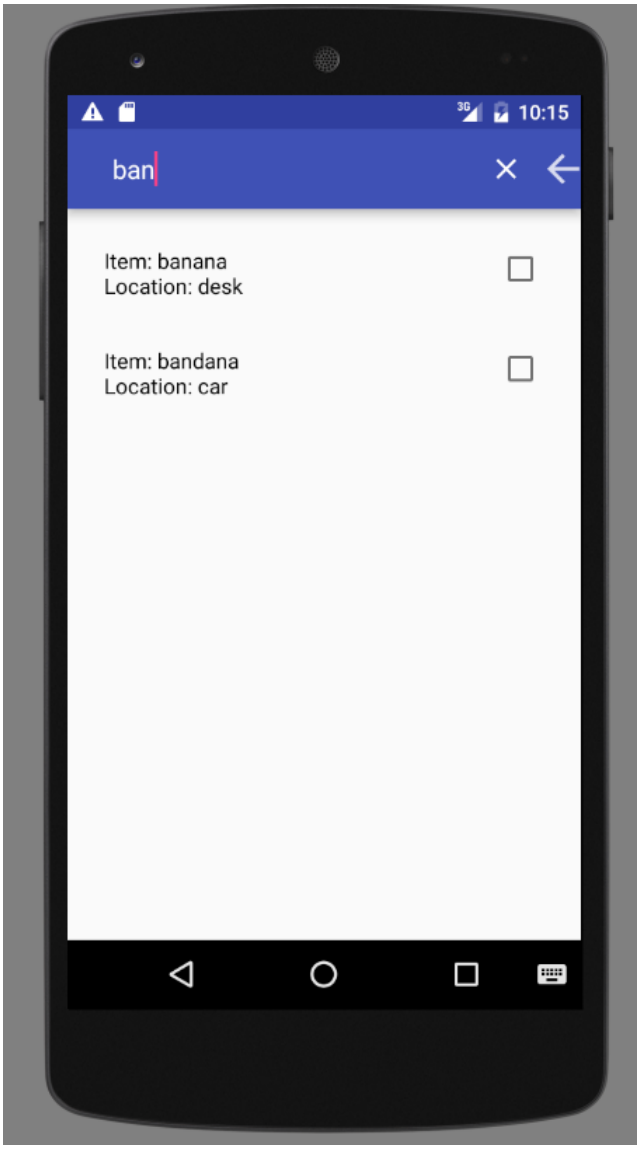

Figure 9: Searching in ListPage

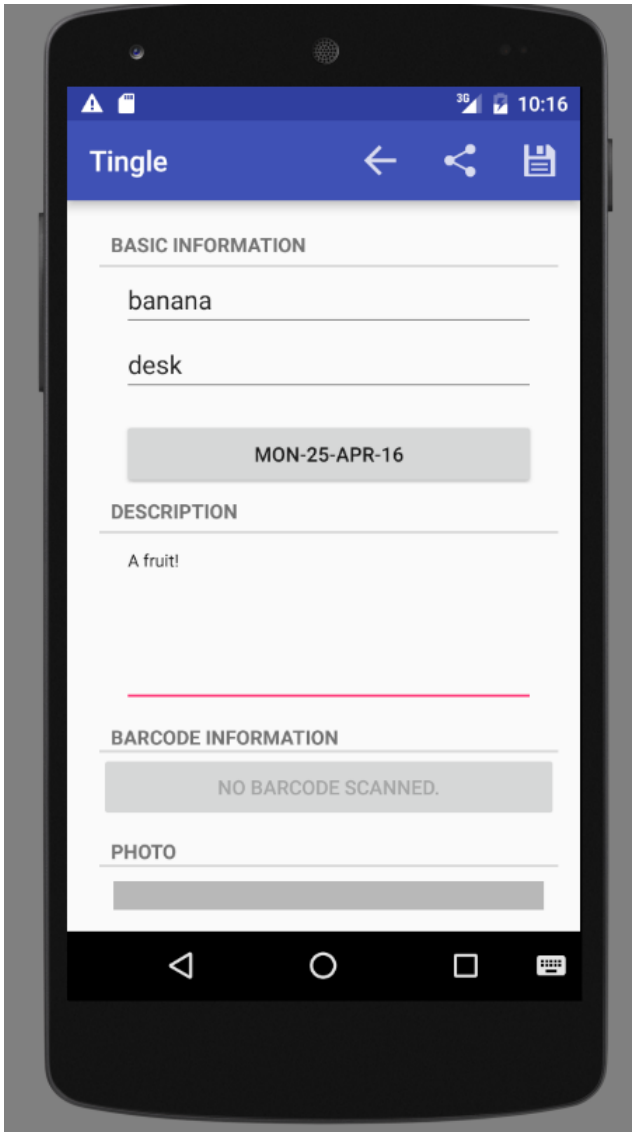

Figure 10: DetailedThing page in portrait mode

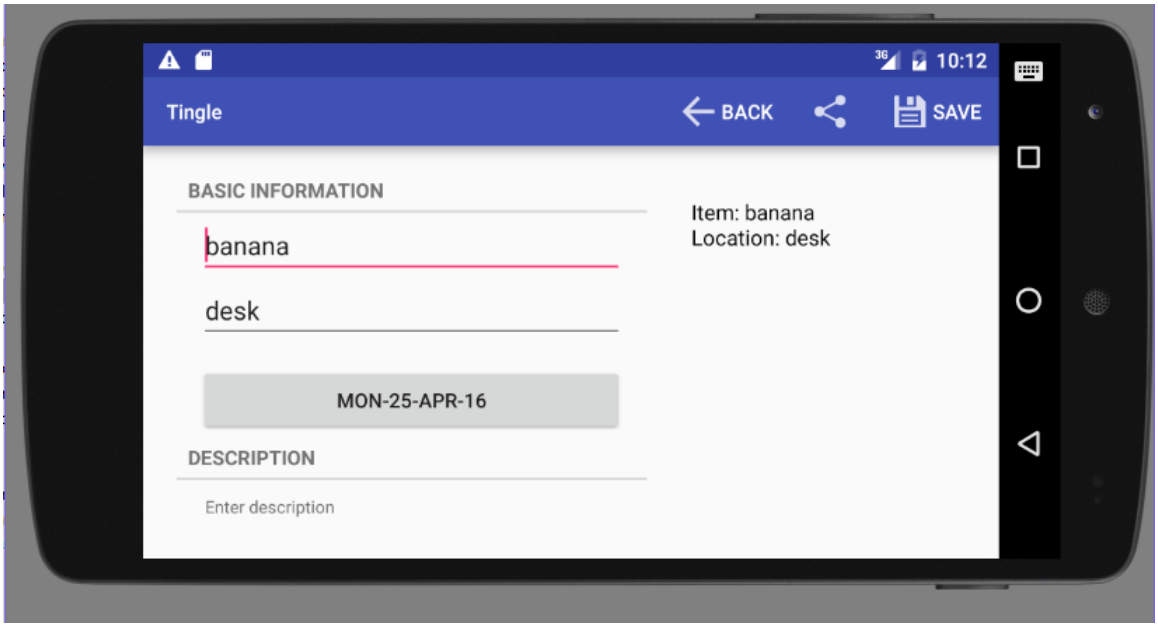

Figure 11: DetailedThing page in landscape mode

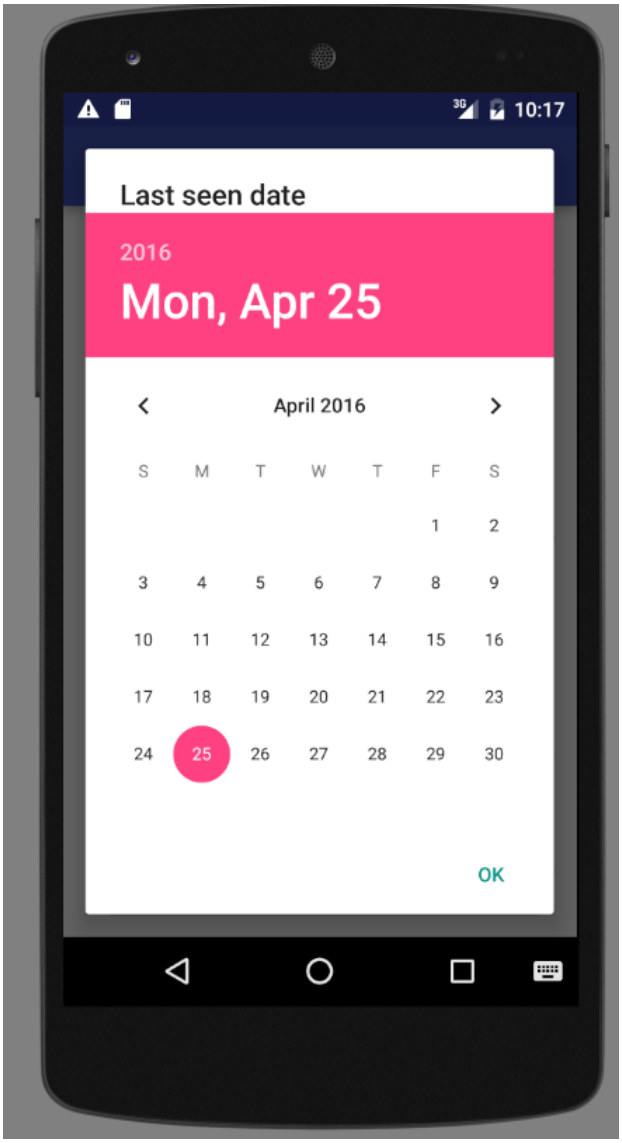

Figure 12: Date picker in DetailedThing page