Michael Ivan - UCLA - UID: 504589159 - October 2016

**Abstract**

I tend to have thoughts throughout the day on various things. In an effort to not forget them I write them down on whatever is nearest, be that notebook margins or scratch paper or similar. Grouping and condensing these notes is almost impossible. In addition, I tend to forget where they are. I have a friend who records every beer he has tried in a small notebook that he carries with him. When I go to the gym, I bring a small notebook to record my workouts.

Physical notebooks have limitations: they can be difficult to update, search or copy. In order to hold information on different topics, either multiple notebooks need to be carried or all notes need to be contained within the same notebook. They also require something with which to write. I personally like to minimize what I carry to my keys, phone and wallet.

By using Android Studio 2.2 and my Samsung Galaxy S5 I have developed a phone app, called “Scatternotes”, to contain the information from various notebooks, while expanding the interaction with these notes, as well as creating a foundation for more notebooks to be added. This app is a concise tool to be used for tracking a couple of different categories of notepads. It is much better than maneuvering a basic memo app to do more than it was designed to do.

**Overview**

When I was working on my undergraduate I had an aversion to databases. They were large mountains of tediousness and complexity. When I decided to pursue my masters I knew that I had to learn databases. Even if they did turn out to be what I thought of them, I still knew that experience would be beneficial. I learned SQL and MySQL in CS143 in Fall quarter of 2015. The monolith was demystified for me. There still could be the tediousness and complexity in database setup and queries but overall the picture was clearer. I’m not sure if tables are the absolute best way to store data but it still made sense to me. I like the ideas of every row having the same types of columns. This makes querying and design straight forward.

In coming up with ideas for my Capstone project I knew I wanted to do a phone app. Mobile is the future. With cloud computing becoming more interwoven with everything, more users will be using their phones as terminals. Mobile devices will also run more and more powerful stand alone applications. Knowledge of the inner workings of Android apps is not a waste. This was one of the reasons why I chose to do an Android app.

As I was going over the various ideas I had for phone apps, there was a common theme. Almost all ideas that I had included a need for persistent data storage. Since most of the apps would be storing the same kinds of data I decided that a database would be necessary. I had learned that all Android devices have SQLite on them, which worked out well since I knew SQL. I combined a couple of the different ideas that I had into one app. I had thought of creating a memo app, an app to keep track of beers tasted, and an app to keep track of gym workouts. All of these had many similarities in what it would take to make them. This would also cut down on the number of apps I would use to keep track of these things.

I went with a form type of UI as this would be a good interface to databases for users. While they didn’t have to know how the backend worked it doesn’t hurt to model the input close to the storage. With the experience of making this app, I’m more comfortable with databases and will probably always include them in future apps that I make. The only other alternative I know of would be to create text files to store information. That is something I would rather avoid. Managing files can be tricky and parsing text can be error prone.

I knew that I had to include the main database operations in my app because they’re also basic use cases for my app. A user would like to create, modify and delete memos. There’s no reason that a simple to-do list needs to stay around forever. Sometimes I don’t know everything about a beer when I’m tasting it, so I’d like to be able to add things in at a later time. With SQLite I could allow columns to be null and filled later, if at all. Doing this with text files as my storage would be difficult. I’d have to be careful not to overwrite adjacent data.

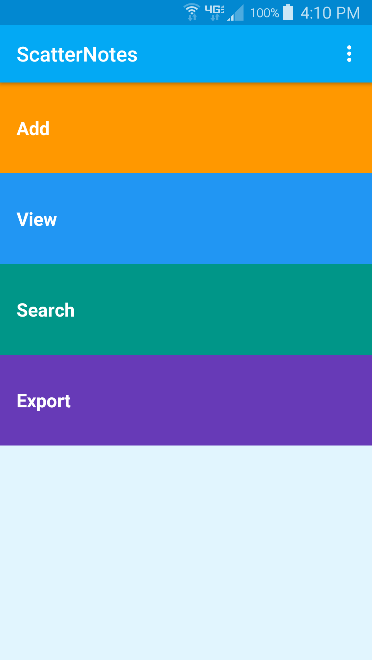
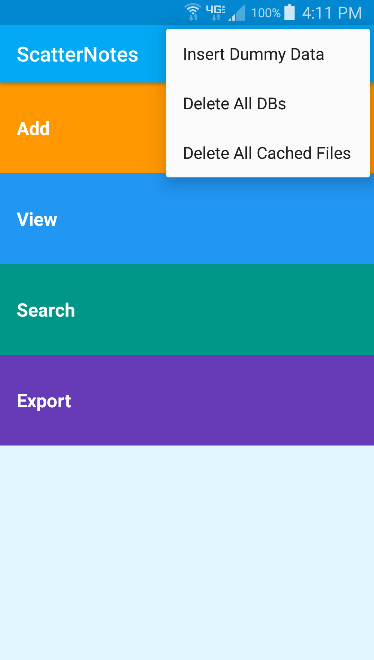
**Process**

Before this project I had never made a phone app of any kind. I did have java, xml and SQL experience. I used various online resources, most of which was Udacity courses to learn the basics of Android. Once I had sufficient knowledge, I went about making a preliminary app with one table in a database. I slowly expanded on that adding new features and new tables. The name came from a combination of “scatter brained” and “notes scattered around”. There are four main note types: memos, beers, workouts, and movies. SQLite was used as the database for Scatternotes.

Android apps consist of one or more activities. Each activity has a UI component and a backend component. An xml file is used to create the base layout and java is used to handle the events from that UI. In order to transition to another activity an “intent” has to be sent, which can then be received by another activity to do what was intended. An example in Scatternotes would be View activity sending an intent to start View Memo activity. Scatternotes would internally see this intent to launch and begin the View Memo activity. Intents can also send parameters or extra data to the receiving activity. This is used to send table row ids, column values and queries between activities.

**Flow of activities**

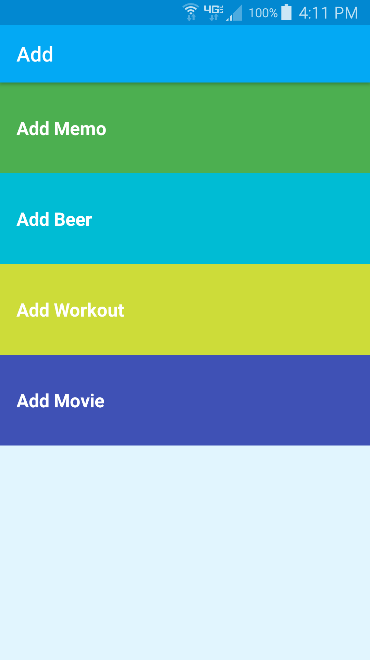
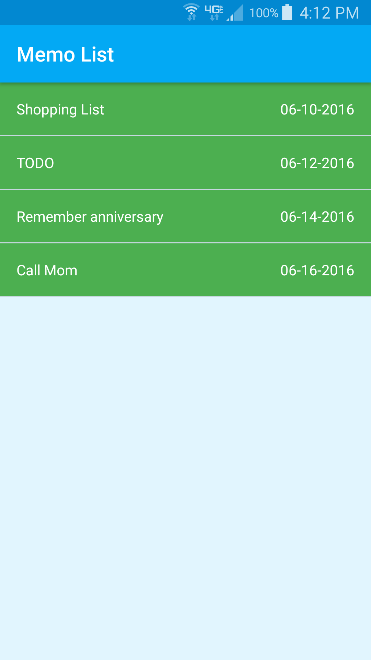
*Main*

This is the main screen of the app. The three vertical dots in the upper right corner are a menu with three options. These are mostly here for debugging. “Insert Dummy Data” creates a few entries into all of the databases. “Delete All DBs” deletes and recreates the tables used in the app. “Delete All Cached Files” is used to delete temporary files created for the purpose of exporting databases via email.

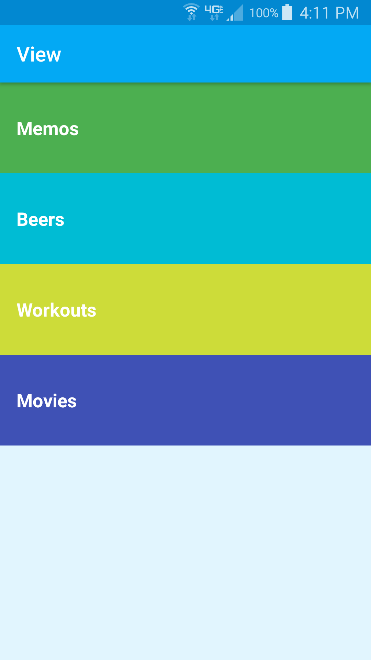
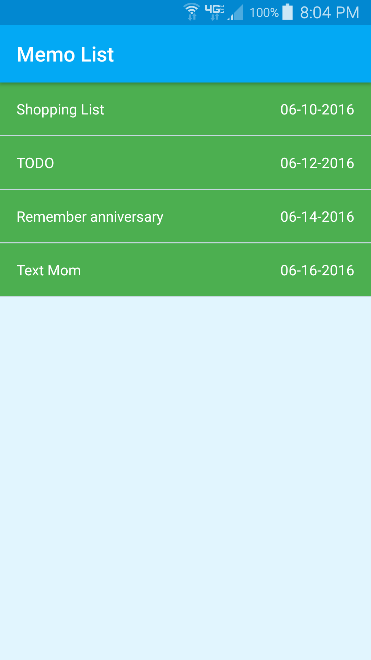
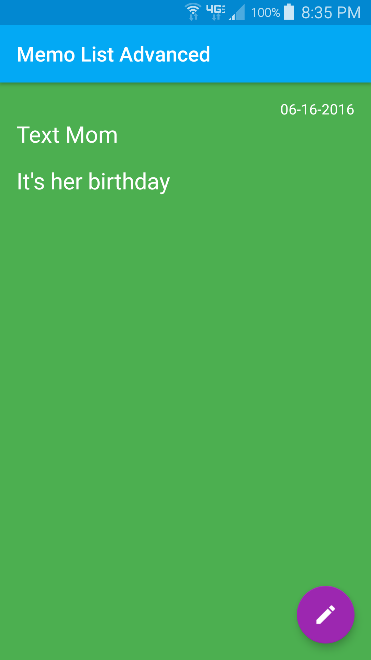
There are four main activities that can be started from here: Add, View, Search, and Export. Add is used to create a new note for the specified database. View is used to see the chosen database contents in a scrollable list form. Search is used to search within a database and will display results on another activity in a list. Export is used to send CSV file representations of the database via email.

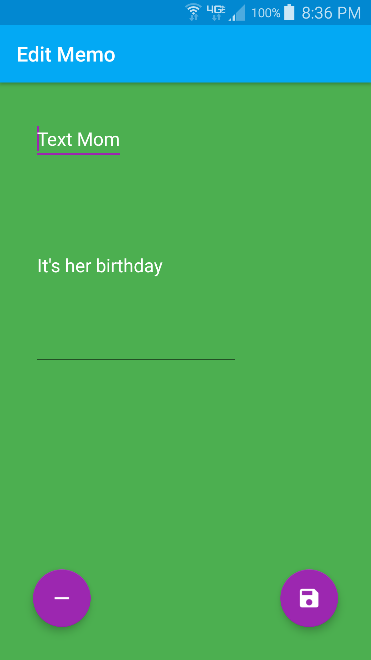
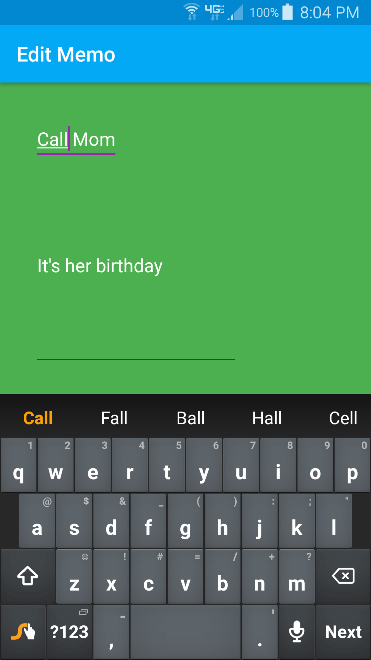
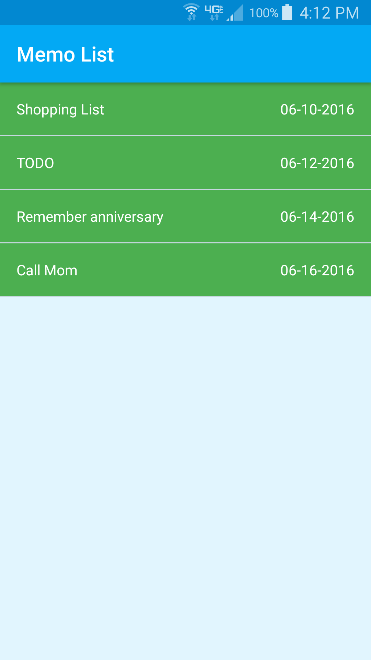
*Add*

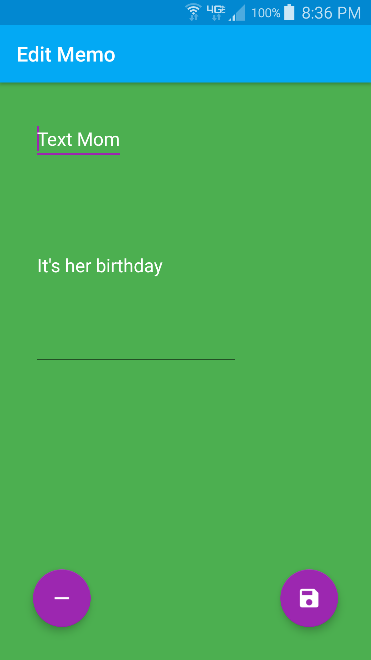
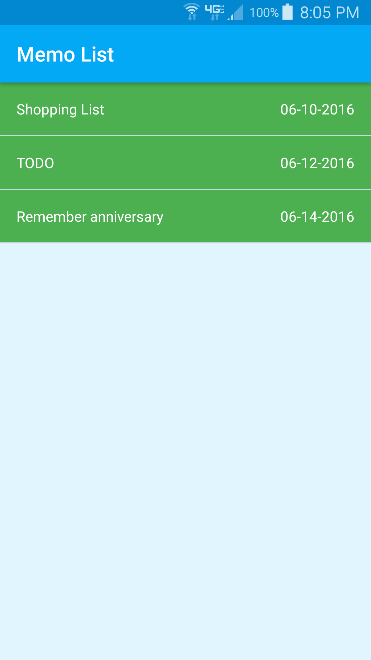
Above is an example of the Add activity. Here we are creating a new memo. First we click “Add” from the main screen. Next we are taken to a list of possible notes to create. “Add Memo” is chosen and we are taken to the relevant screen. Clicking the save icon in the lower right corner takes us to a view of all memos created. This is the View Memo activity. If a required field is left empty then a Toast message is displayed and there’s no attempt to insert a new row.

*View*

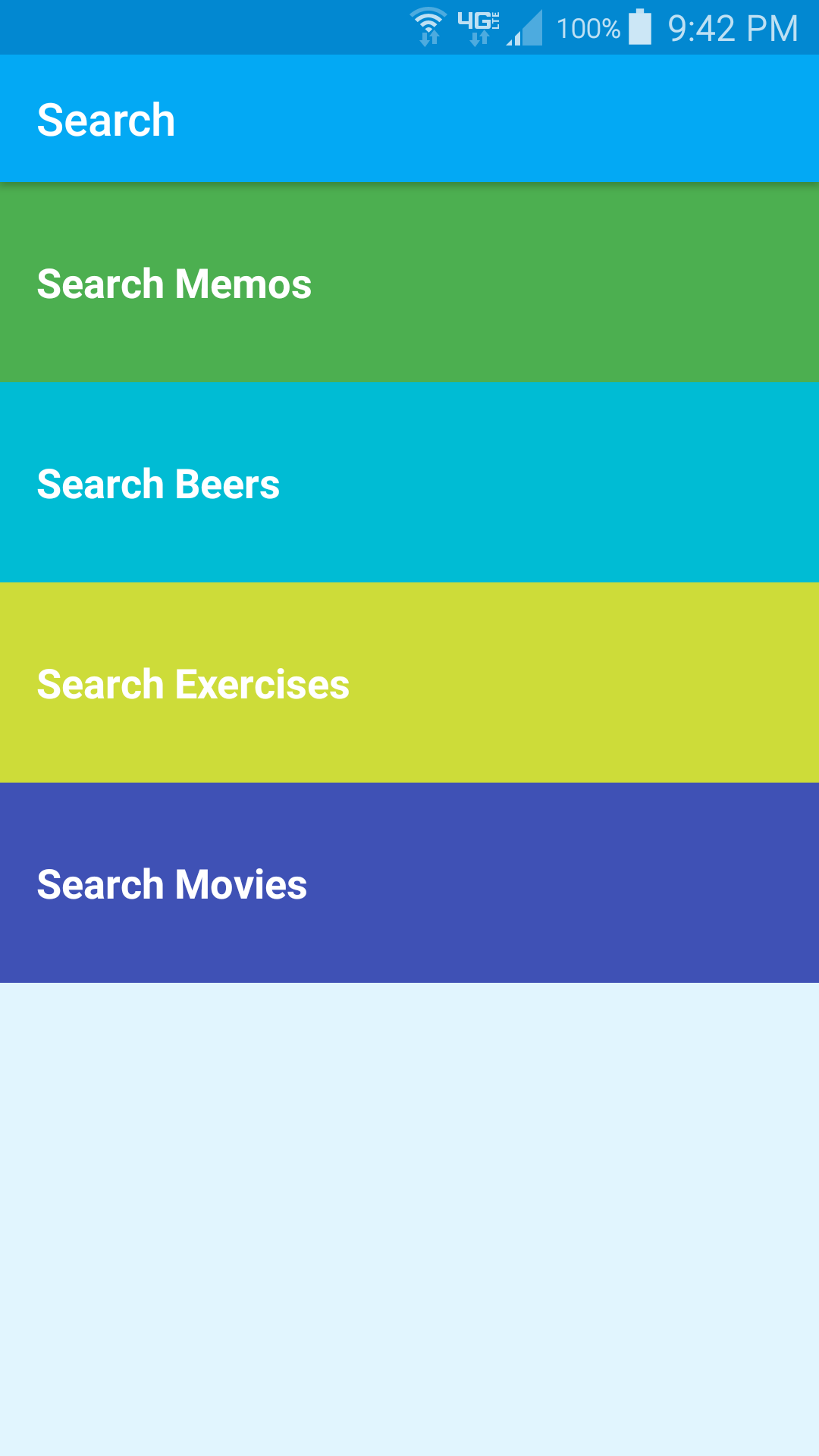
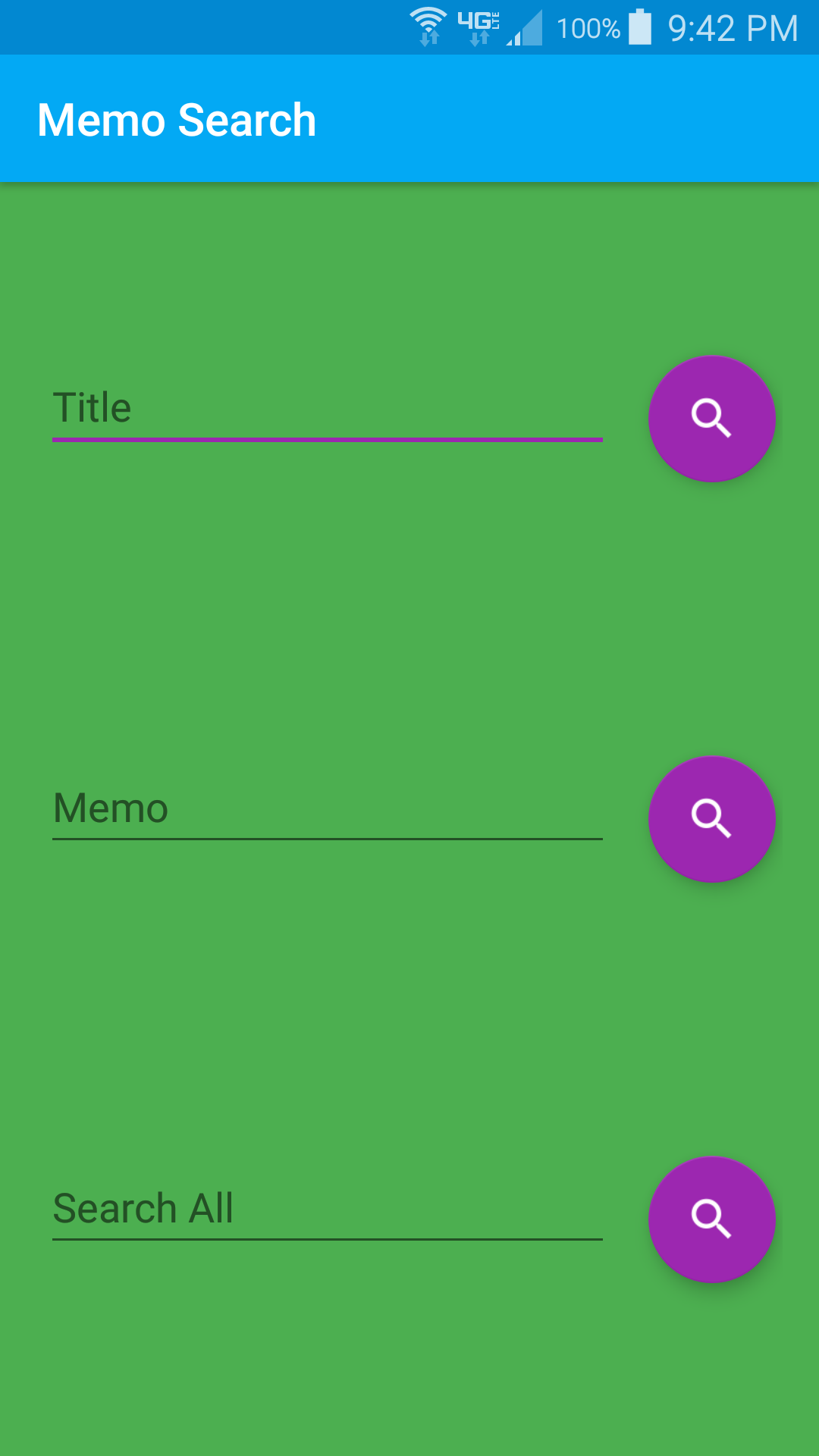
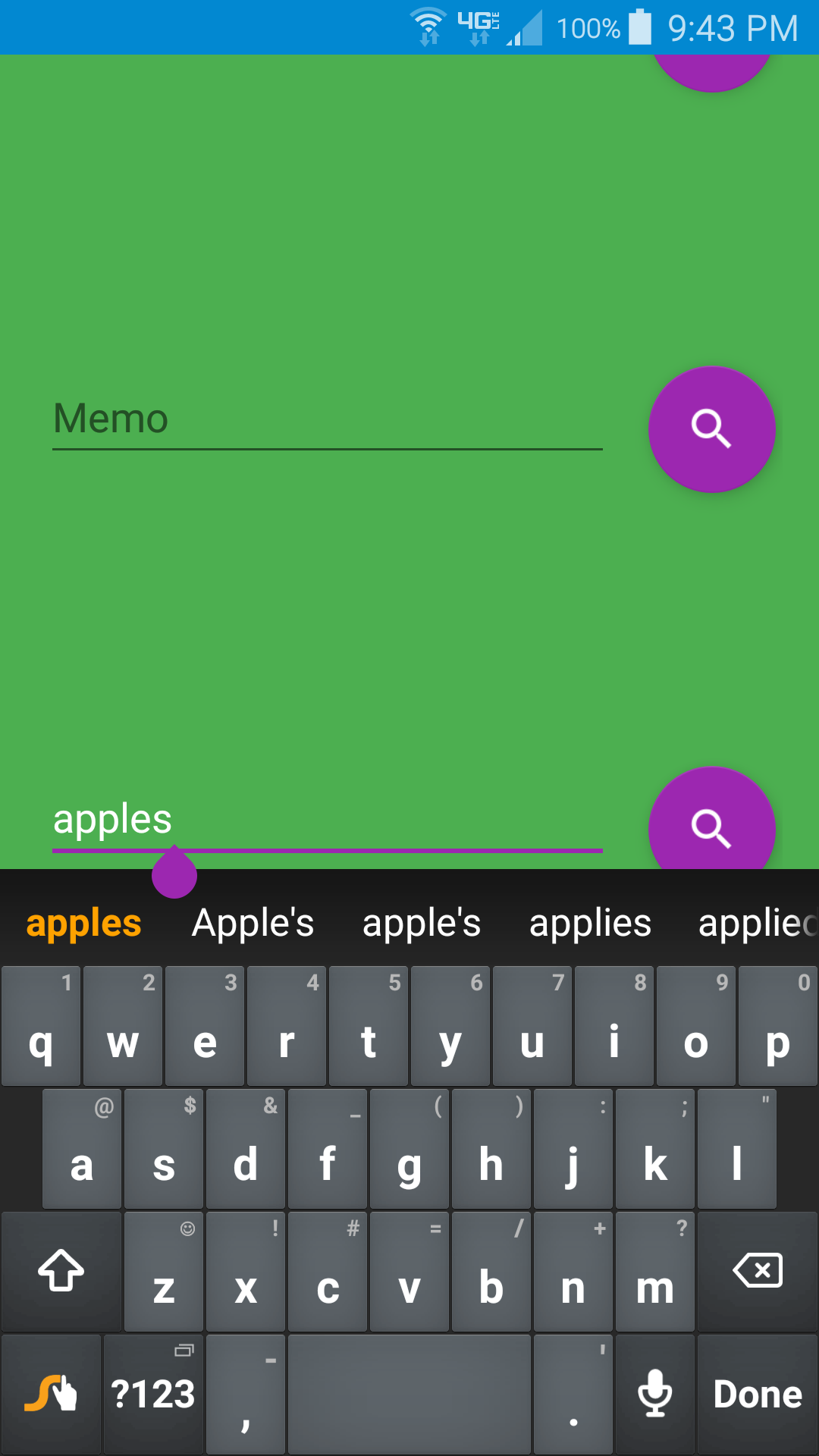
  

Above is an example of flow from the View activity. The “View” text box is clicked from the main activity. From there “Memos” is selected which brings up the Memo View activity. The memo with the title “Text Mom” is clicked bringing up the Memo View Advanced activity, which shows the title “Text Mom” along with the note of “It’s her birthday”. From here the edit icon in the lower right corner will bring up the Memo Edit activity. There are two buttons on this activity. The minus sign is used to delete this memo, which will return to the Memo View activity as shown below. If the save button is clicked the edit screen is brought up in which the title or note can be edited. The example shows “Text Mom” being changed to “Call Mom”. After the change has been made and the save button clicked, the Memo View activity is returned.

This is the flow of deleting an entry from the edit activity. While on the desired memo, clicking the minus sign will delete the memo and return you to the memo view activity.

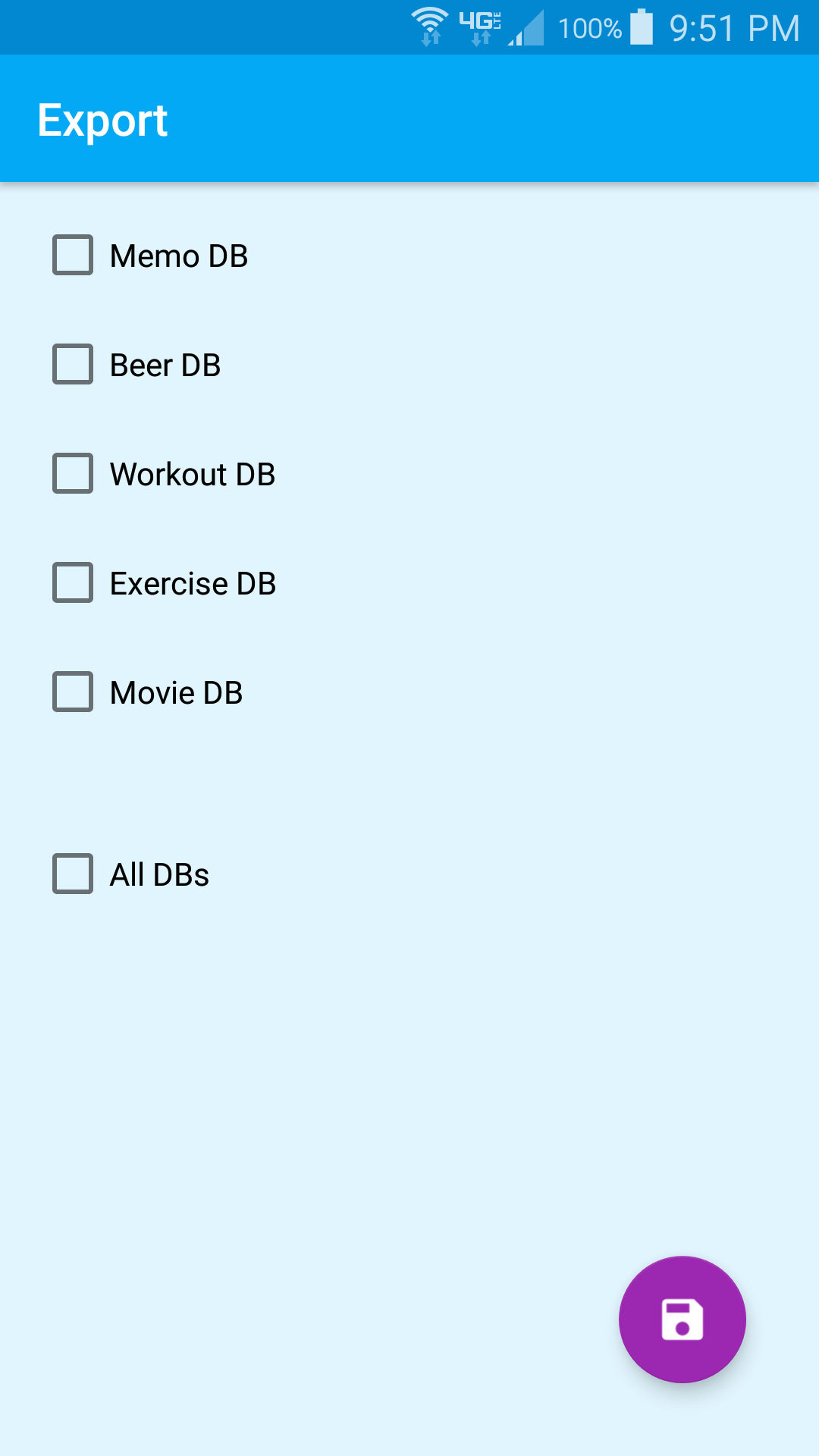
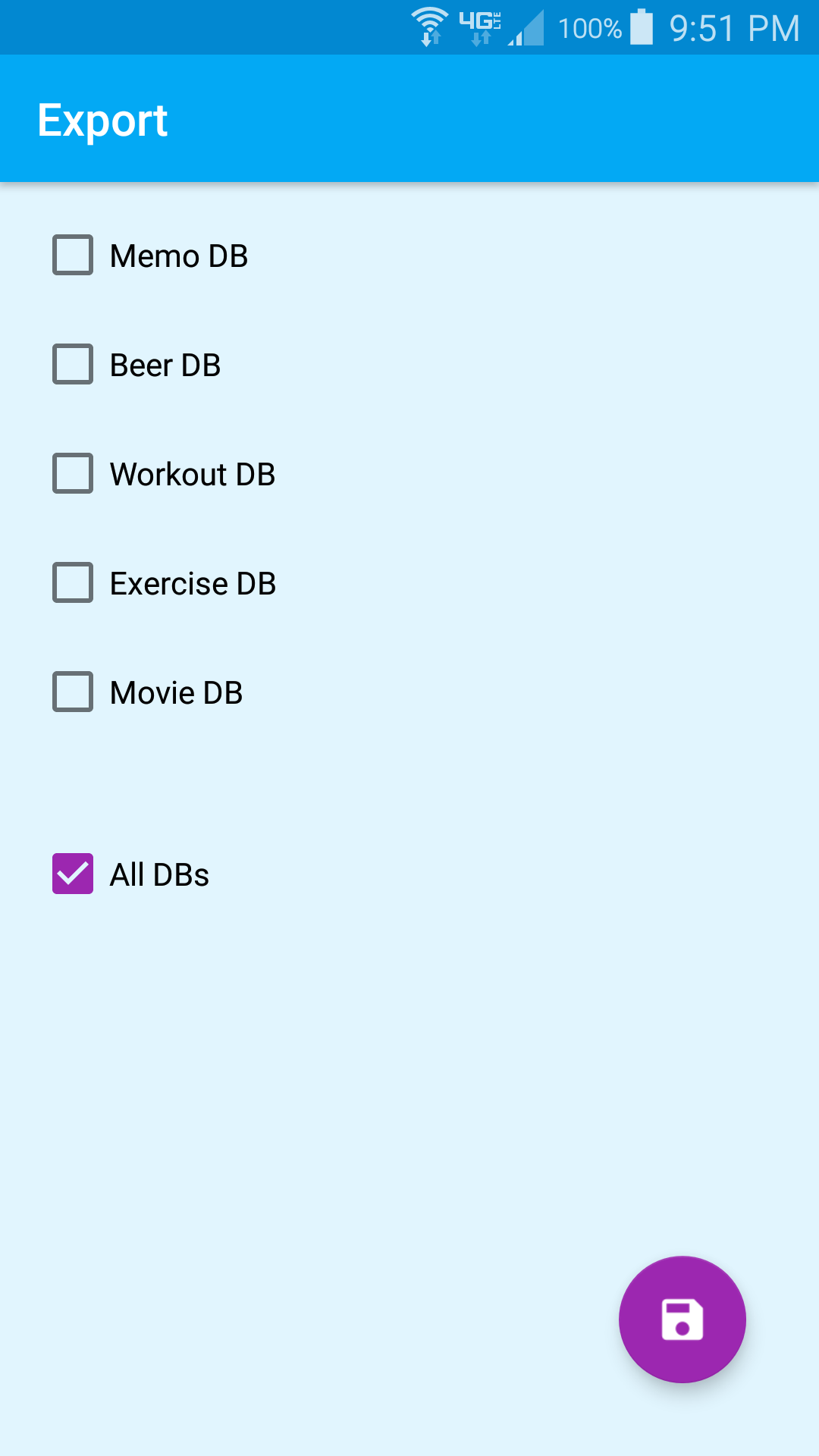
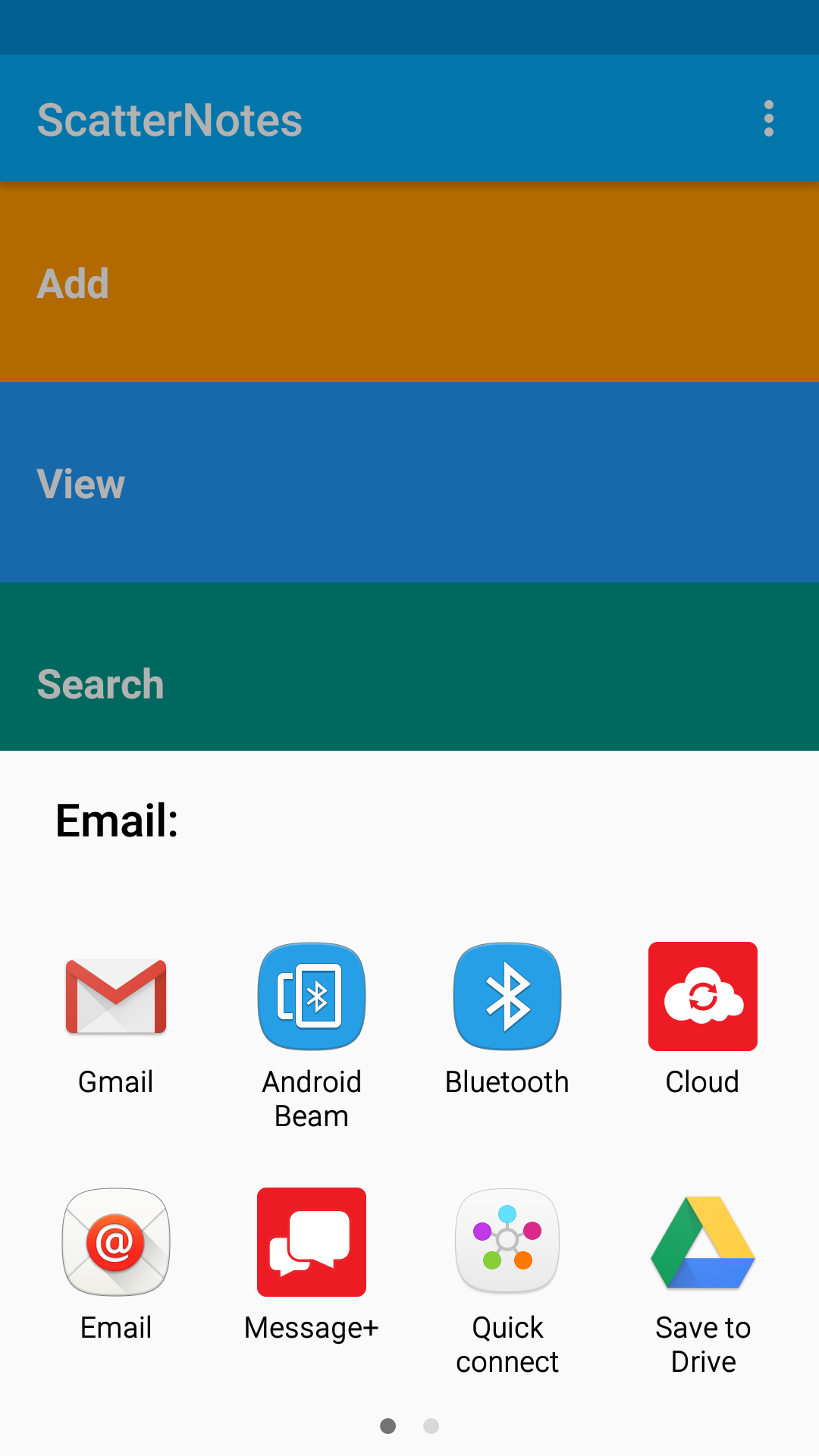
*Search*

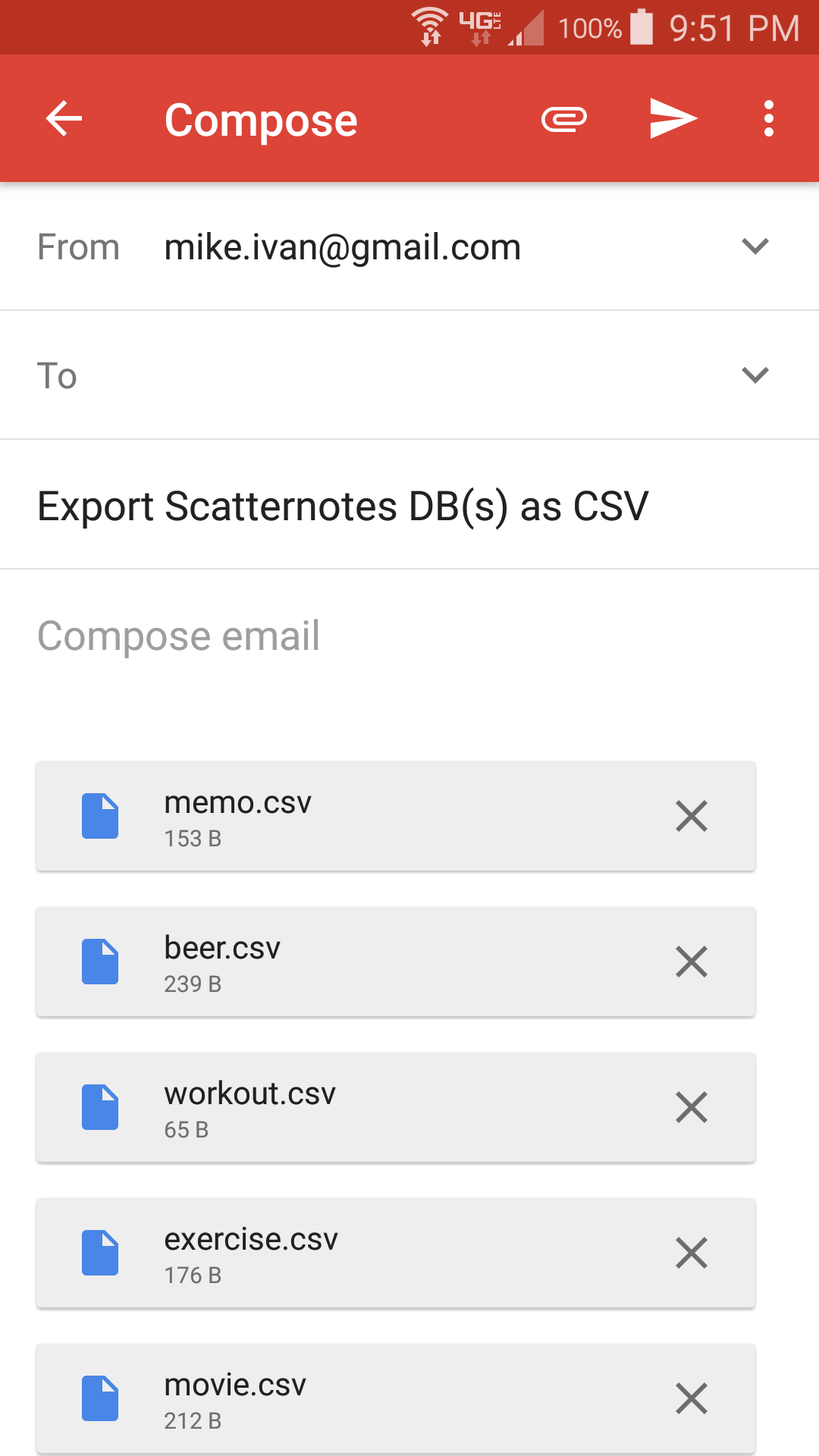
  

Above is an example of using the Search activity. To begin, “Search” was clicked from the main activity. “Search Memo” is clicked which brings up the Memo Search activity. Into the Search All field is entered “apples” and the search button corresponding to the Search All field is clicked. This brings up the Memo Search Results activity which only has one memo that matches. Clicking initiates an advanced view of the memo. From here the edit button can ben clicked if desired.

*Export*



This is the Export flow. “Export” was clicked from the main activity. From there the Export activity is launched with check boxes for which databases are to be exported. Clicking “All DBs” will override the status of any other check box. In the example “All DBs” is selected which will solicit the user for the email app to use. Gmail is selected and the corresponding databases are attached as CSV files.

**Schemas**

All tables are contained in the database “notes.db”. All tables have an integer id that serves as a primary key. This key is auto generated and is hidden from the user. The id is used for internal functions. An alternative to a system generated row id, would be to create a primary key based on user input, but that could lead to failed table row creations due to conflicts. The conflict would have to be presented to the user and they would have to rectify the situation. Another alternative option of basing a primary key on user input would be to allow duplicates. This could cause confusion when doing searches or displaying. Ultimately, I decided to take this out of the hands of the user. There also could be instances where a user might want multiple entries with the same name or title. With a unique id for each row this is possible.

All tables also have a date component. This is auto generated on entry creation. This is immutable by the user or the application. Although the workout and exercise databases are interlinked, the exercise table contains redundant date data for cases of editing and display purposes. This is to limit the number of cross-queries to retrieve the data.

Another commonality between the tables is that they all have a contract class defining the names of their columns and a database helper class to regulate creation, upgrade and downgrading.

Memo table

CREATE TABLE memo ( \_id INTEGER PRIMARY KEY AUTOINCREMENT,

title TEXT,

notes TEXT,

date TEXT NOT NULL);

The memo table is for storing simple messages containing a title, date and a string of characters. The above is the schema used. Both the title and the notes can be empty if the user chooses. The reasoning behind this was two use cases: a reminder and a notepad. The memo can be used as a simple reminder by just viewing the title. Or the memo can be used for a list of things, which doesn’t require a title.

Beer table

CREATE TABLE beer ( \_id INTEGER PRIMARY KEY AUTOINCREMENT,

company TEXT,

product TEXT NOT NULL,

rating INTEGER NOT NULL DEFAULT 0,

notes TEXT,

date TEXT NOT NULL);

The beer table is used for storing information about beers tasted. The two main columns for this database are the product and rating. At the very least a beer entry should have a name (the product) and a rating for said beer. For some, a rating is all that is necessary and notes don’t need to be made. Also, a brewing company is not always apparent.

Workout table

CREATE TABLE workout ( \_id INTEGER PRIMARY KEY AUTOINCREMENT,

date TEXT NOT NULL);

The workout table is used to associate exercises with a particular workout. Since this table is intended to encompass entries in another table, only the id and date are needed.

Exercise table

CREATE TABLE exercise ( \_id INTEGER PRIMARY KEY AUTOINCREMENT,

parent\_id INTEGER NOT NULL,

exercise TEXT NOT NULL,

date TEXT NOT NULL);

Currently the bulk of the exercise table is in the exercise column. This is used to store a brief description of the exercise. In later versions this could be greatly expanded to include repetitions, sets, and other more specific characteristics. The parent\_id is used to associate an exercise with a workout. The parent\_id is an \_id from the workout table.

Movie table

CREATE TABLE movie ( \_id INTEGER PRIMARY KEY AUTOINCREMENT,

title TEXT NOT NULL,

year INTEGER NOT NULL DEFAULT 2016,

rating INTEGER NOT NULL DEFAULT 0,

notes TEXT,

date TEXT NOT NULL);

The movie table is used to record data about movies seen. It’s similar to the beer table in that the title and rating are deemed a defining trait of the row. Also, a year is considered important, but given a default year since the year of the movie may not be known. This can be edited later.

**Files**

The following are the files needed to be changed in order to implement a table in Scatternotes:

AndroidManifest.xml is a required file in the root of an Android project. It is used to define activities, their names, and file providers. If an activity isn’t given a specific name, it defaults to the application name.

The three pairs of file: AddAdcitivty.java and activity\_add.xml, ViewActivity.java and activity\_view.xml, and SearchActivity.java and activity\_search.xml, will all need to be modified in a similar way. A new clickable text box will need to be added to the layout file and the corresponding java code will need to be added to send an intent to the correct activity.

ExportActivity.java and activity\_export.xml would need a new check box added for the new database and the corresponding file creation code.

The following files use the memo database as an example, but the same types of files will need to be created:

MemoContract.java is a file with the database contract. This consists of constants for the name of the table, names of the columns in the table, and any other constants used in the table, like ratings for the beer and movie tables.

MemoDbHelper.java is the class which handles the creation, upgrade and downgrade of the memo database.

MemoAddActivity.java and activity\_memo\_add.xml are linked to by the main Add activity in which the user can fill in the memo fields and then insert into the memo database. These two files need to be created according to the schema of the database and then use the appropriate database objects.

MemoEditActivity.java and activity\_memo\_edit.xml are used to edit the specified memo which is passed in by intent. From this activity we can delete or update the memo. After this activity is done the Memo View activity is returned.

MemoEntry.java is a class used to represent some of the data of a memo in order to create a custom ArrayAdapter for use in a ScrollView. This class is also used by onclick events in order to store the necessary information to be sent through intents.

MemoAdapter.java is the class that extends ArrayAdapter<MemoEntry>. This allows us to associate an adapter with a ListView element, in this case it’s a ScrollView. A ScrollView will handle the task of creating a scrollable list of textboxes in an activity. By extending ArrayAdapter for MemoEntry we can control how items in the list are displayed. The file memo\_list\_item.xml is used by the MemoAdapter in order to do so. The xml file is used to create the layout and MemoAdapter uses its overridden getView function to set the correct TextViews to those of the MemoEntry selected.

MemoViewActivity.java and item\_list.xml are used to create a scrollable and clickable list of all memos in the table. All of the table view activities reuse item\_list.xml as it is just a ScrollView, but they still use classes relevant to displaying data in the respective tables.

MemoViewAdvActivity.java and memo\_list\_item\_advanced.xml are used to display the full contents of a row in the memo table, except the id, which is never shown to the user. This also sets up a button that will transition to the edit page of the currently viewed memo.

MemoSearchActivity.java and activity\_memo\_search.xml are accessed by the main search activity. These files set up the fields in the memo table to be searched. In all search activities there is a field for search all, which will search for the term in all of the fields listed on the page. The only exception to the search all is for the exercise table, which only has one searchable column. The search query is created in this activity and then sent as a string to Memo Search Results activity.

MemoSearchResultsActivity.java and activity\_memo\_search\_results.xml are used to display the results of the query sent by Memo Search activity. Each result in the query is clickable and will open a Memo View Advanced activity with the clicked memo.

**Lessons Learned**

Android apps can be daunting when first learning. It’s much more than just learning Java or xml. I had no idea what the R class was. Each part of the UI had its own xml component with many attributes that could be set. In addition to that each of those could be altered by the Java code with its own set of methods to set attributes. A simple text box has about 70 attributes that could be set. Things dealing with text size, input, margins and various other settings.

For the Add Workout activity I used a calendar widget. Android takes care of everything with displaying and handling the calendar. From the backend I simply asked for the selected date, which was given as a Java date object and then I used SimpleDateFormat to get the date the way that I wanted. If needed, I could see using this widget in other tables.

Stackoverflow and Google contained many answers to the questions that I had. Sometimes the answers were that I had to use a different version of a class or a completely different way of doing it due to API deprecating features. If I couldn’t easily find something with a simple search, then the extensive Android documentation from Google was enough to figure it out.

I used three methods of debugging when developing this app: display information in a text box, logging information to the main Android logs, and Toast messages. I used the text boxes initially to make sure the information would display, which I could style at a later date. I used logging mostly for inserting dummy data into the databases. This was due to a lot of data being added in a quick amount of time. As it was initiated from a menu, updating to a text box didn’t seem like a good idea. Plus a lot of information could potentially be put into the log files, including values for the columns I was trying to insert. I had no idea that the little text pop-up on the screen was called a Toast message. This was helpful in everything. I would create clickable buttons and insert a Toast just to see if they were being called correctly. I would also use Toasts between activity transitions to make sure intents were sending and receiving the correct values.

Sending values via intents can get confusing. I should have mapped out my activities earlier than I did. A flow chart of activities with Java and layout files along with where they were sending intents and what was contained in those intents really helped to keep things straight. There were times when I would send an intent with a simple integer value for a table row but the receiving activity would get some garbage value. This was because I was inserting an intent with a value of “WorkoutTableEntry.\_ID” but the receiver would expect “ExerciseTableEntry.\_ID”. Excessive time was spent debugging this only to find the string name inconsistency.

**Future Work**

While I was able to complete the main features that I wanted for this project, there are still ways to improve and expand. Error handling, style modifications, and UI flow are areas that can always be improved.

The table view activities are just a list. A search box could be created somewhere on the page that will search all fields of the current table. This would have the same functionality as going through the main search activity, but would provide a better UI experience. In addition, on the view pages, there could be a drop down menu that allows the user to sort the list based on certain columns in an ascending or descending order. For example, a user could sort the workout view from oldest to newest date.

Currently on the movie and beer search pages the rating can only be searched on a specified value. This could be improved to search based on ratings above or below a threshold. Adding another spinner for equals, less than, greater than, less than equals and greater than equals could be implemented and vastly improve searching. This would allow the user to search for all movies with a rating better than 3. In the current implementation the user would have to do multiple searches, one for each integer value of rating, to get the same results.

The tables could be expanded to include pictures. A picture of the beer tasted could be attached to the review of the beer. Or perhaps a picture of a shopping list could be included with a blank memo, thereby eliminating time needed for typing.

Content providers could be implemented for the database functions. This would add another layer of abstraction between the UI and backend. Currently the UI directly calls these functions. This would facilitate other applications using the data in the tables.

The exercise table is a string. Real exercises can be more complex with various numbers of repetitions and sets. There’s also a distinction between cardio and strength exercises. This was the reasoning behind keeping it a string at first. The main point was to be able to associate many exercises with one parent in the workout table. This could instead be extended to the workout table being the parent of many different exercises spread across different tables each representing a different exercise or type of exercise.

Export could be expanded to deliver data to cloud storage or other alternatives. Export could be changed to include an import option that would allow inserting data from a file, web page or cloud. Currently the only way to insert into the database is by the menu option to “Insert Dummy Data” or by individual user input. Perhaps the user has Scatternotes on another device and would like to sync with a new one. As it stands, there’s no way to do that with this app.

Scatternotes could always be extended to have more tables for representing other types of notes. A wine category could be added, which would be a lot like the beer one, except for fields for year, type, and so on. Related to this Scatternotes might be expanded to allow users to custom create their own types of notes. This certainly would be non-trivial. The user-generated schema would have to be stored in a database, potentially involving multiple tables to contain the column names and types, such as text or integer. Each layout would probably be an empty xml element which would be built up by the backend code. This would have to happen for every single activity, involving multiple queries from different databases to build up the basic UI. Intents would have to be used to send the custom database names around and generic activities would handle them. Everything would have to be dynamic.

**Resources**

Stackoverflow – stackoverflow.com

Udacity Android Development for Beginners

Udacity Android Basics: Multiscreen Apps

Udacity Android Basics: Data Storage

Google Material Icons - <https://design.google.com/icons/index.html>

Material Design Color Palette - <https://www.materialpalette.com/>

Android Developers - https://developer.android.com/index.html