Project Report

Receipt Manager

The receipt manager app allows users to enter information from their receipts into the app's database, where they can then retrieve that information in order to reflect on their spending. To add receipts, the user can use the app's menu to navigate them to the add receipt page, where they are able to enter the receipt's location/store, date, subtotal, and total price. They are also able to choose to add receipt items from this page, where they will be asked to enter the item's description, the quantity of items they bought, and the price of the item. They are also given the option of giving the items categories, such as food and/or entertainment. Entering the pricing information for the items will automatically update the total price of the receipt to match the items. If the user does not want to add the items to the receipt and just wants the receipt information, then they can add the total price themselves.

After entering receipt information, they can again use the app's menu to navigate to the receipt history page, where they will be shown a list of the receipts entered in the app. The list shows the basic information of the receipt, which includes the store the receipt came from, the date it was received, and the total price of the receipt. Clicking on any of the receipts will bring the user to a new page where they will get to see all the details of the receipt, including the items on the list and the details of the items. On the receipt history page, the user is also given the option of only looking at receipts for a specific period of time, such as just the current week or current month.

Going back the menu, the user can then navigate to the spending summary page, where they will be able to view graphs and stats on their spending habits. They will have the option of grouping their spending by item category or store name, which will give them a pie graph, or by date, which will give them a bar graph.

Finally, the user is able to get to the settings section of the app through the menu, which allows them to view the list of categories and stores that they use. They can add new categories and stores to be used with newly entered receipts.

Build Requirements

Be sure to sync the project with gradle.build. Because of the mavenCentral depencies used in the app, an internet connection is required to build the app for the first time. If Android Studio is unsuccessful in making the project from the Git repository, ensure that the correct build tools are installed.

Design and Implementation

The receipt manager app contains few activities, and is mainly operated from the main activity. Drawer navigation was implemented for the app's menu, and for the most part, clicking on any of the options will the menu will switch fragments in the main activity. The only exception to this is the add receipt option, which will take the user to a new activity. The only other activity used is when navigating from the receipt history fragment to the receipt detail activity.

The add receipt activity contains two fragments - the add receipt fragment and the add item fragment. Using a separate activity for adding receipts kept the code for adding receipts and items out of the main activity. The add receipt fragment contains an auto complete text view, a date picker, and two normal edit texts. The auto complete text view is where the user is meant to include the store name from their receipt, and it will give the user options to auto complete if they're starting to type in a location that is already in the database. The date picker is used to choose the date the receipt was received. The two normal edit texts are used for the subtotal and total costs of the receipt. We chose to have both a total and a subtotal for the receipts because we weren't taking into account any taxes or discounts on items, so if the total is different from the subtotal, then the user could input that. However, the user does not need to add anything to the total and subtotal sections as the subtotal is calculated from the items added to the receipt, and if

the total edit text is empty, it is made equal to the subtotal. This fragment also includes two buttons - an add items button and a finish button. Clicking add items will take the user to the add items fragment, and clicking the finish button will add the receipt, along with all its items, to the database, while letting the user know that they added a receipt through a toast message. If the store name they entered didn't match any in the database, then a new location is created for that store and is also added to the database.

The add items fragment makes use of three edit texts, a multispinner, a couple buttons, and a recycler view. The edit texts are meant for the user to enter the description, quantity, and price of each item purchased. The multispinner is used to let the user choose one or more categories for each of their items. After filling in an item's information, the user can click the add item button to add that item to the item list they're creating. Each item they add will be filling up the recycler view, which will be displaying the information they just entered, along with a button beside each item allowing the user to remove the item from the list. Clicking the finish button will button will bring the user back to the add receipt fragment where they will find the subtotal section has been updated to match the items they added.

The receipt history fragment makes use of a spinner and a recycler view. The spinner is used to filter the receipts being retrieved by a certain time frame. The time frame options are the past week, the past month, the past year, or all time. The automatic option is the past week so that the app isn't retrieving any unnecessary data. The information shown in the recycler view is the store name, date, and total price of each receipt. Clicking on any of the receipts will bring the user to the receipt detail activity.

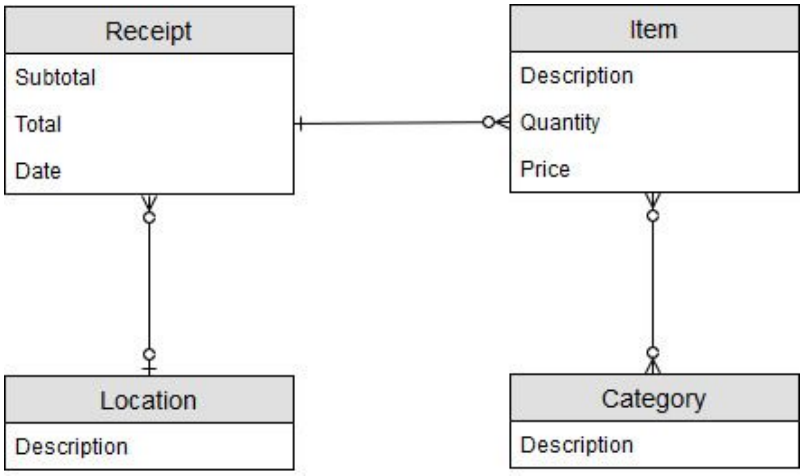
The receipt detail activity is a simple activity containing many text views and a list view. The text views show the receipt's store name, the date it was purchase, and both the subtotal and total price. The list view is used to show the list of that receipt's items. The user simply needs to click the back button to get back to the receipt history fragment.

The analysis fragment contains a spinner and a frame that holds other fragments. The spinner is used to go through the available stats and graph fragments. The options on the spinner are general stats, category, location, and date. The general stats fragments is simply made up of text views showing the user statistics such as their total dollars spent, the amount of receipts

they've stored, their most expensive receipt, the amount of items they've purchased, and the most expensive item they've purchased. The pie category fragment displays a pie graph showing the amount of money the user has spent on each category. The pie location fragment displays a pie graph showing the amount of money the user has spent at each location. The bar fragment displays a bar graph showing the user how much money they have spent on each day in the past two weeks. The past two weeks restriction was created so that the graph wouldn't get too cluttered. The graphs were all created using the AnyChart API which will be discussed further in a later section.

The settings section of the drawer navigation allows the user to go to either the category fragment or the location fragment. These fragments are nearly identical. They each contain an edit text, a button, and a list view. The edit text is there to allow the user to enter a new category or location. Clicking the add button will take what the user has entered and add it to the database. The list view simply lists the categories and locations that are already in the database.

The schema of the database can be best described by the following entity-relationship diagram:



Storing receipts and receipt items seems to be best done with a relational database. Only four primary tables are needed to fully normalize the database. An intermediate table for relational mapping is needed to handle the many-to-many relationship between items and categories. All of the above tables are represented as Java objects in the source code.

The app implements this schema with an SQLite embedded database. The code design pattern being used is inspired by a blog by software engineer Oyewale Oyediran. The design

pattern simplifies the code in regard to doing reads and writes. API methods require only a receipt object to write to the database. This keeps the code simple and modular. As an example, consider this snippet of code:

```
DAOContainer daoContainer = new DAOContainer(getContext());
daoContainer.open();
Receipt receipt = new Receipt(new Date(), 100.00, 115.00); //date, subtotal,
total
daoContainer.addPopulatedReceipt(receipt);
daoContainer.getReceiptDAO().fetchReceiptsInDateRange(ReceiptDAO.LAST_WEEK);
daoContainer.close();
```

To create a database connection in an activity or fragment, a DAOContainer is created, needing only the current context. Thus, a receipt can be added to the database with only one method. To read receipts from the database, one must retrieve the corresponding DAO and calls its methods. To close the connection, one must close the DAOContainer.

Features Included that Weren't Covered in Class

One of the more difficult features of the app was the use of AnyChart for Android Studio. The AnyChart API allows the creation of many different charts, which they list on github. We chose to make use of the pie chart and the bar graph they offer, as they matched our needs the best. There is sample code offered in their github repository, but it is quite limited, so we did need to make use of the documentation provided by the creators. The original documentation they offer is meant for use with javascript, which was generally easy to translate to android studio, but they didn't have any android studio specific documentation, which meant there was quite a bit of trial and error when trying to implement the charts we wanted.

A couple other features we included that weren't covered in the course were the android.widget.AutoCompleteTextView and the android.widget.DatePicker widget. The AutoCompleteTextView widget was actually quite simple to use, as it worked like an edit text, except it was provided with an ArrayAdapter of strings that it could use, sort of like a spinner, and it was passed a threshold number so that it knew how many letters needed to be typed before it could pass the user suggestions. We found how to use this code from the android

documentation. The DatePicker is also from the official android documentation. It uses a fragment to create a calendar popup which allows the user to select a date. We chose this method of input rather than simply letting the user enter a date in numbers to avoid dealing with any formatting errors.

Use Cases

Case 1: No added receipts on Spending Summary

The user opens the app for the first time and is greeted with the analysis fragment which is currently displaying the general stats fragment. There isn't any information showing because there aren't any receipts in the database yet. Choosing any of the other spinner options will show a blank screen as there currently isn't any data to show.

Case 2: No added receipts on Receipt History

The user clicks on the menu button, which opens up the drawer navigation. From the menu, the user clicks on the receipt history button, which shows them the receipt history fragment. It is currently empty except for the spinner. Clicking any of the spinner options will appear to do nothing as there isn't any data to filter.

Case 3: Adding categories

The user clicks the menu button again, and then clicks on the Categories button. This brings them to the categories fragment, which is currently showing a list of four categories: Food, Clothing, Entertainment, Bills. The user enters "Food" into the provided edit text, and presses the add button. A toast pops on screen letting the user know that this category already exists and asks them to enter a new one. The user types "Electronics" instead, and hits the add button. "Electronics" is immediately added to the bottom of the list.

Case 4: Adding stores

The user goes back to the menu and clicks on the Stores button. This brings them to the location fragment, which is currently empty except for the edit text and the add button. The user types the word "Sobeys" and clicks the add button. "Sobeys" is immediately added to the top of the list. The user types in "Sobeys" again and hits the add button. A toast pops up letting them

know that Sobeys already exists, and asks them to enter a different store name instead. The user types in “Superstore” instead and clicks the add button again. “Superstore” is immediately added to the end of the list.

Case 5: Adding base receipt

The user opens the menu one more time and clicks on the Add Receipt button. The user is brought to the add receipt location and is being shown the add receipt fragment, which is currently displaying empty fields. The user clicks the finish button and a toast pops up asking them to fill out the location and subtotal fields properly. The user starts typing “S” into the location field, and a list shows up below the edit text offering the user to choose from Sobeys or Superstore. The user clicks Sobeys and the field is filled out for them. The user clicks the finish button again and is once again asked to fill out the location and subtotal fields properly. The user finds they cannot enter any letters in the subtotal field, and they enter the number 1 instead. They click the finish button and another toast pops up telling them that a receipt has been added. The fields have also been reset to empty.

Case 6: Receipt History

The user goes back to the menu and clicks on the Receipt History button again. They are brought back to a similar page where they are still unable to see any receipts. The spinner is currently set to filter receipts only for the past week, so the user sets the spinner to all time. The user is then able to see a receipt for Sobeys and $1. The user clicks on the receipt and is brought to a page where they are able to see the details of their receipt. Those details are the store’s name (Sobeys), the receipt’s subtotal ($1), and the receipt’s total ($1).

Case 7: Spending Summary location graph

The user opens the menu and clicks on the Spending Summary button. This takes them back to the page they were at when they initially opened the app. They are now able to see stats shown for the receipt they entered earlier. Choosing the category option from the spinner, there is still a blank page because they have not yet added any items with categories. Choosing the location option from the spinner, the user is then shown a pie graph of where they have spent their money. If the user clicks on the graph, a tooltip shows up letting them know the dollar

amount that they've spent. Choosing the date option from the spinner shows a blank page because they don't have any receipts with dates attached to them yet.

Case 8: Add Receipt with date

The user goes to the menu and clicks on the Add Receipt button again. They are shown the empty fields of the add receipt fragment again. The user types Bluenotes into the location field, then clicks on the date field. A calendar pops up with the current date highlighted. The user clicks the OK button at the bottom of the calendar. The current date shows up in the date field. The user then clicks on the add items button.

Case 9: Add items

The user is shown another screen with empty fields. Without filling in any of the fields, the user clicks on the add item button. Doing so causes a toast to appear asking the user to fill in the item description, quantity, and price properly. The user enters Jeans in the description, 1 for quantity, and 24.99 in price. The user then hits the add item button and the item's details show up at the start of a list. The user clicks the remove button that showed up next to the item, and the item disappears. The user enters in all the previous information and clicks add item again, and the item reappears in the list. The user then clicks the finish button, and is brought back to the add receipt page. The subtotal field has been updated to match the price of the jeans. The user clicks the finish button and is told that a receipt has been added.

Case 10: Receipt details

The user goes to the receipt history page, and they are able to immediately see their new receipt. They can see the name of the store, the date on the receipt, and the total price of the receipt. When they click on the receipt, they can see all the previous details along with the subtotal, and the item on the receipt. Clicking the back button brings them back to the receipt history page.

Case 11: Automatic location adding

The user then goes to the Stores page. The page has been updated, and Bluenotes is now part of the stores list since it was added earlier.

Case 12: Spending Summary date graph

The user goes back to the spending summary page and can see that the stats and the location graph have been updated again. The category graph is still empty, but the date graph now has data in it.

Case 13: Adding item with categories

The user heads back to the add receipt page and enters Best Buy in the location, and chooses Nov. 1 as the date. The user then clicks the add items button and is brought to the add items page. The user enters headphones for the description, 2 for quantity, and 12.99 for price. The user then clicks on the category multispinner, and chooses both electronics and entertainment as categories, and then clicks add item. The item is then listed on the screen. The user then types in RDR2 for description, 1 for quantity, and 79.99 for price, then chooses entertainment as a category. The user then clicks the add item button and the second item is listed underneath the first one. The user clicks the finish button, and is brought back to the add receipt page.

Case 14: Entering total for tax amounts

The user can see that the subtotal has been updated, but realises it doesn't account for taxes. For that reason, the user enters 120 into the total field, then presses the finish button.

Case 15: Receipt History spinner for filtering by date range

The user goes to the receipt history page, but cannot immediately see the receipt they just entered. They use the spinner to select the past year option, and they can now see their new receipt. They click on the receipt, and they can see all the details of their receipt, including the difference between the total and subtotal, and the categories for their items.

Case 16: Spending Summary

The user goes to the spending summary page where the graphs and stats have all been updated, and there is now a graph for the category section. The user closes the app by pressing the back button.

Implementation Issues

Multispinner from **thomashaertel**

- A multispinner solution was needed for adding multiple categories to an individual item. While there were several options, very few of them worked or had up-to-date documentation. This multipinner does work, but unfortunately the documentation is not in depth. A few hours of troubleshooting were needed to get it working correctly.

Database Integrity

- While implementing the database was fairly straightforward, ensuring the integrity of its data was quite difficult. Unlike a fully-fledged MySQL or SQLServer database, SQLite lacks utilities for for viewing the current state of tables and executing queries. DB Browser for SQLite is the best solution for this, but it is inherently imperfect, since database files must be imported from the emulator each time one wishes to browse.

Flows and Fragments

- Our solution to multi-page flows, specifically in regard to adding a receipt, was to dynamically swap out fragments when needed. To do this, we made a secondary outer activity to handle the add receipt and add items fragments. At the time, this seemed like a good solution. However, it made handling reconfigurations very difficult.

Known Bugs

- Graphing fragments are slow (especially on emulator).
- Dollar amounts do not format correctly on graphs.
- Our solution to screen reorientations was to prevent reconfiguration on rotate. This is obviously not ideal, and if we had more time we would provide a more robust solution.
- Text overflowing on statsFragment and with too many categories on an item.