Project Report

Group members: David Alexander Adams and Aleix Molla

Project Description

The app, E-receipt, provides the user the option of storing and accessing receipts on their phone or tablet. In an ideal situation, the app would display a unique bar code to the device that would establish a connection between the user and the store's database. At the checkout, the user would simply scan the bar code, and the receipt will be sent to the phone once the transaction is completed via Bluetooth. A full version of this app could also implement Android Pay within the app, such that the user does not have the exit the app.

The app is ideally targeted for consumers who like to keep track of their receipts and finances. In the demonstrated version of the app, pre-loaded receipts will be simply loaded from a remote server (link provided below) through a JSON object and parsed into a database. The user can choose to load, display, and/or graph the monthly spending.

Link: http://137.149.157.18/CS2130/e-receipt/

Operating Instructions

Main Activity provides the user with four options

- Load Receipts
    - Data is loaded from a server
    - User can view receipts
- Show Receipts
    - It is only possible to displays receipts if Load Receipts was previously clicked
    - User is able to load a list of receipts from several months
    - User can click on an individual receipt to display the items purchased and their price
- Show Budget

    - Application will display a graph of monthly spending from the last five months
- Add Receipt
    - User can scan their QR code at the cashier (feature not implemented yet)

Use Cases

(*Assuming user clicks load receipt, selects an individual receipt, traverses back to the main menu, clicks show budget, traverses back to the main menu, and finally click add receipt)

1. Splash Activity is launched, and terminates after the specified delay time
2. Main Activity is launched
3. Receipt List Activity is launched
4. Receipt Activity is launched
5. Graph Activity is launched
6. Add Receipt Activity is launched

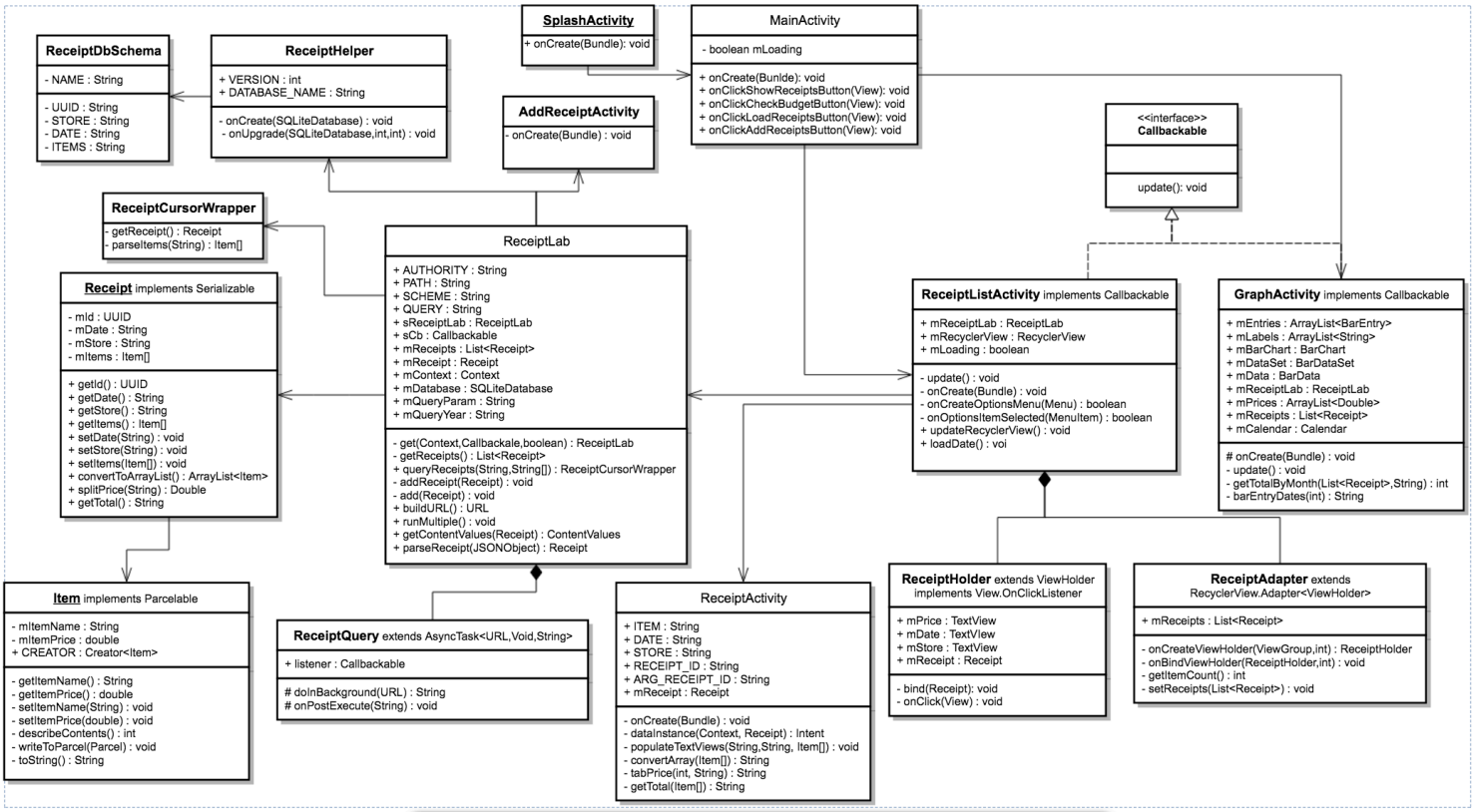

Image 1: UML Diagram

Image 1 represents displays a UML diagram that E-receipts implements.

Activity Description

The following snapshots demonstrate each activity that a user can interact with.

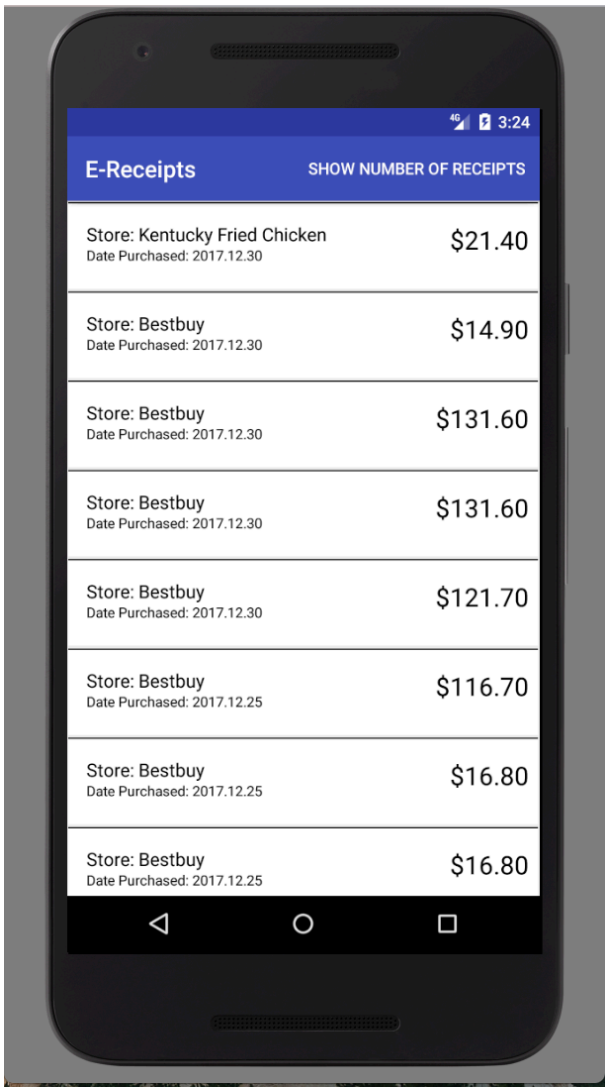

Figure 1: MainActivity

Figure 2: ReceiptListActivity

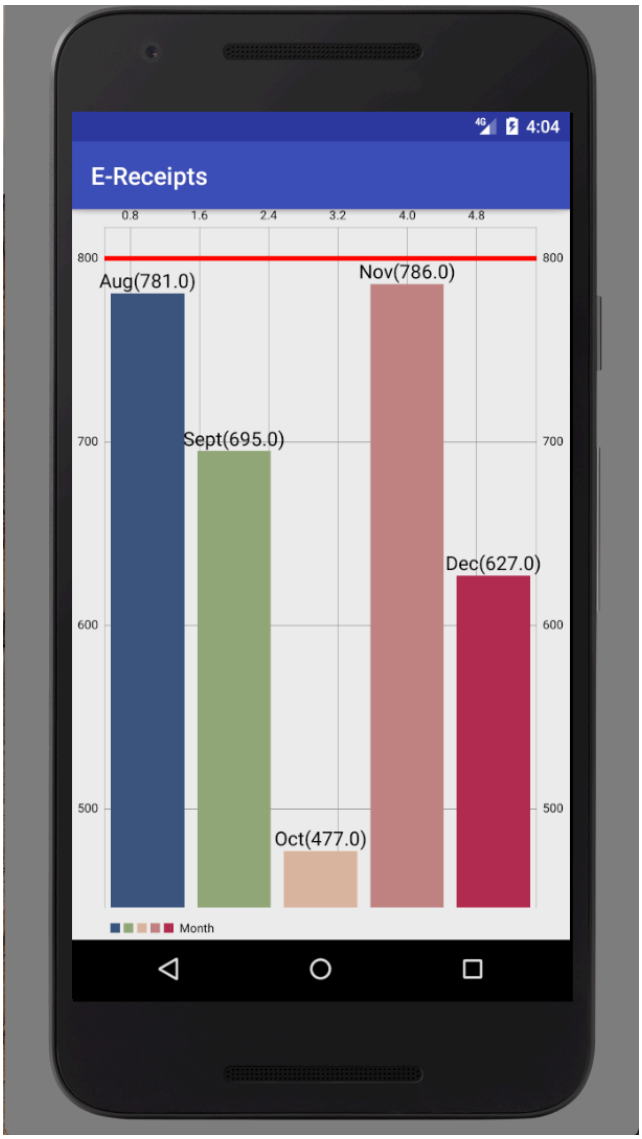

Figure 3: GraphActivity

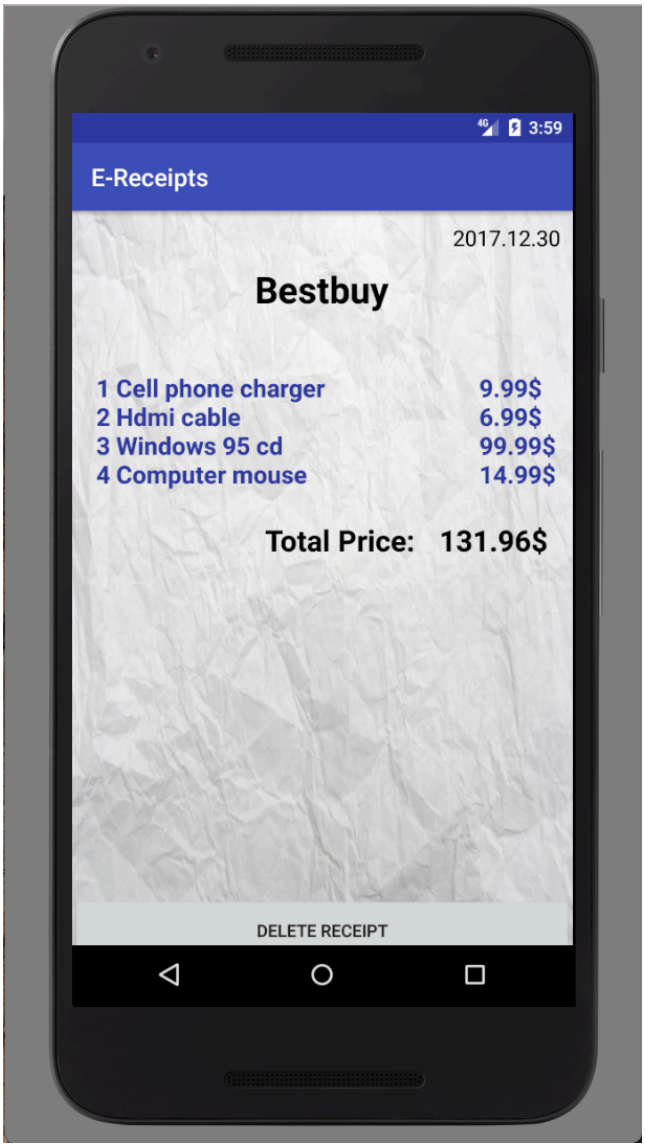

Figure 4: ReceiptActivity

Figure 5: AddReceiptActivity

Prior to loading, the application displays a splash screen.

Figure 1 displays a screenshot of the main user activity. The user can interact with the app by clicking the four buttons provided in the activity.

Figure 2 displays a screenshot of the receipt list activity. The receipt list is displayed using a Recycler View. Each list displays the name of the store and the total price from that store.
The activity also displays the number of receipts that are currently stored into the database. This count can be displayed by click on the menu icon/button “Show number of receipts”.

Figure 3 displays a screenshot of the graph activity. The total of all purchases is calculated from each month and displayed as a bar graph from the last five months. The red line represents a limit line that currently cannot be adjusted.

Figure 4 displays a screenshot of the receipt activity. Receipt activity mimics an actual receipt. The name of the store, the date of purchase is displayed on the top. Each item and its price is displayed. The total is calculated at the bottom.

Figure 5 displays a screenshot of a the add Receipt activity. As of the current implementation, the application only displays a non-functioning QR code to mimic potential further implementation.

Design and Implementation

The app uses the following classes:

Callbackable interface: This interface has one method that is called when the information from the server has finished being added to the database. This method is called in "onPostExecute" in class ReceiptLab. The interface allows the user to access the receipts after they have been added to the database.

Item: This class serves as a part of the receipt. Essentially, the receipt contains information about the items bought and the price associated with each item.

MainActivity: This class serves as the first page (activity) that the user sees and interacts with after the Splash Screen has finished loading. The MainActivity has four buttons: load receipts, show receipts, budget, and add receipt.

- The load receipts button makes a call to the serves and fetches receipts from the last thirty days. (Note: There may be more than one receipt from one day).
- The show receipts button simply loads up an activity that displays a list of receipts that are loaded from the database.

- The budget button provides a visual representation of the total spending of a month
- Add receipt button displays an image of a barcode that later with further development can transfer information.

ReceiptActivity: This class displays one receipt from a particular store and date. It uses textviews to displays the name of the store and the date, and a scroll view to display the list of items and the prices associated with them.

ReceiptLab: This class retrieves the information from the database and creates receipts to be added to the database. ReceiptLab has an inner class that parses JSON arrays.

ReceiptListActivity: This class handles displaying a recycler view of receipts and their prices. Each receipt in the recycler view can be clicked to display a receipt with the individual items and the prices associated with them.

DataBase:

- ReceiptCursorWrapper
- ReceiptDbSchema
- ReceiptHolder

The choice to implement the AsyncTask inside ReceiptLab was a design choice that revolved around the idea of one classing handling receiving and parsing data, and created a receipt object.

Implementation Issues and Bugs

Database Issues: We created just one table with all the information we need from the server but with our items, the data was in an array of paired Item objects of Strings and Doubles together. Since the database only receives primitive data types (String, char, int, etc.), we needed to store them as a JsonArray String and parse it back to Items every time we did a query for every receipt. A solution suggested by Dr. Andrew Godbout was to create a second Database Table

with those Item elements that shared the UUID of their Receipts in the other Database Table. Given the time constraints and the complexity, we did choose this option.

Creating Receipts: As we cannot correctly simulate any Store machine to send the receipt data, we need to make calls to a randomly generated receipt from a server, creating specific design and layout implementations that won't be used in the original intended App.

Implementing with Array: The server that generates receipts is parsed into an array (not array list) to be displayed when a receipt is clicked from the recycler view (of receipts). The issue that we had was not being able to load a parcelable array. We tried using a serializable but the activity launched from the recycler view would crash. Therefore, we had to change our Items to be held in an ArrayList since that was not causing any issues.

Class complexity: The downside of calling to a server, parsing data, and creating a receipt object adds to the complexity of reading a class.

Merging two years: There is currently a bug with calculating the total for more than one year. For example, if the user were to load receipts where two months from two different years were to be displayed, the totals for those two months would add up and be displayed rather than the individual totals being displayed.

Important Features

- View a list of your receipts sorted by date
- Select an individual receipt and view all the items and the total amount spent in that particular store
- View a graph of your spending from the last five months