FINAL PROJECT REPORT
UMB, CS443 GROUP PROJECT
ON
CURRENCY CONVERTER

Date: 12/20/2017

Presented To: Professor Bo Sheng

Prepared By: Nirajan Nepal

Nirmal Nepal

Project Statement

Our android application name is Currency Converter(CurCon). The application basically gives two services to the user.

1) **Converts the rate of one Currency to another Currency.**
   The main concept of the part one of the app is, user can look up exchange rate from one unit to another. This is the first and default main page of the app. There are 32 currencies provided by European central Bank site which can be converted from one to another. App looks up the exchange rate of the present day. For example, if the user uses the app on 12/21/2017 to calculate the rate from GBP to USD, then the app provides the rate of 12/21/2017.

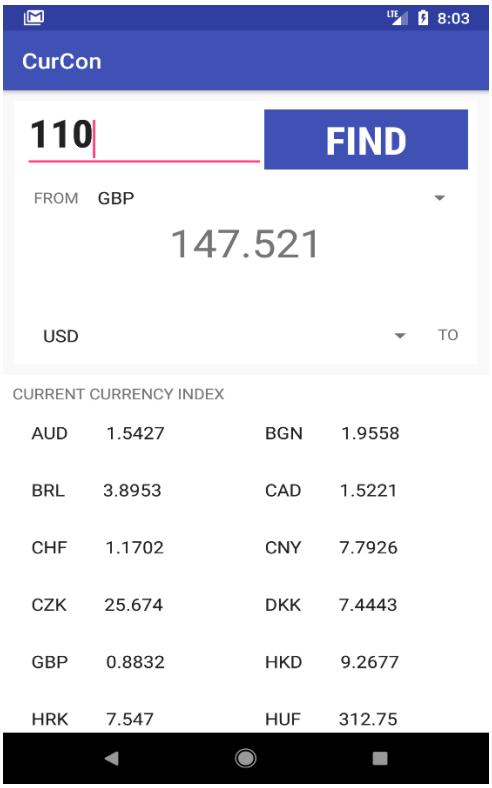


Figure 1

Figure 1 shows the screen shot of the GBP converted and calculated to USD on 12/20/2017. Part one of the app further provides current currency index(CCI) of all 31 currencies based on EUR currency.

2) **Provides the history of user selected Currency from 90 days.**
   Part two, the app provides historical statistics of the currency. From the default main screen, user can touch or select any currency which will then open page two of the app providing the detail of the currency. There are three sections in part 2 screen. At the top, currency name and its CCI is shown. In the middle section, there is a graph. There is a bar, user can stretch or shorten bar drawing the line which represents the days. The graph is drawn by the app by taking the days provided by user through the line. For example, if user touches USD

(12/20/2017) in first page from CCI section then at the top section CCI of USD is shown. In the middle section, there is graph representation of USD from the last 90 days. In the last section of this part, there are again CCI.

Figure 2

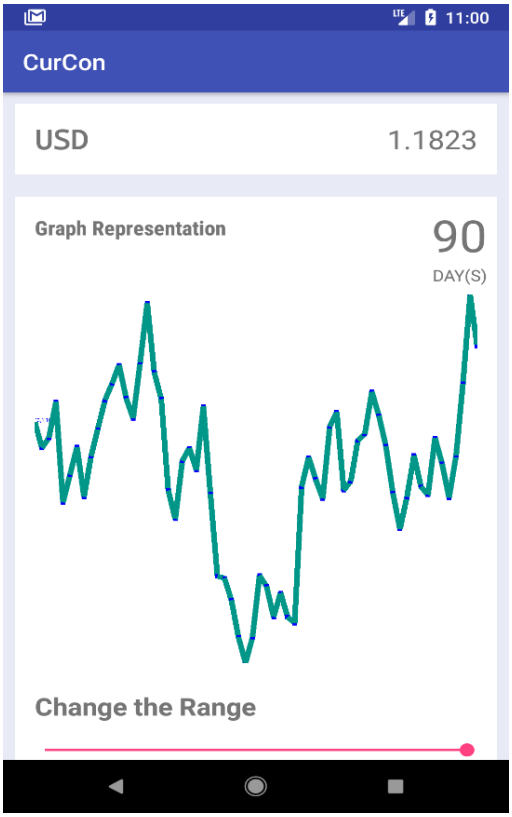


Figure 2 shows the CCI of USD (12/20/2017) and the graph representation of CCI up to last 90 days.

Usually, general people these days demand services which gives easy and simple data because they don't want to work hard. In addition, we found out most of the people these days want analyzed data so that they can save time. Due to these reasons, we were motivated to make app which gives data and analyzed statistics in a simple way and easy to understand.

There are various potential users of this application. Travelers surely will benefit from this application. It is because travelers will plan their travelling ahead and wants to have currency exchanged. They will look up our application and see the exchange rate for today. Also, they will look the pattern for the exchange rate up to last 90 days. Thus, they will predict which will be the right day to exchange so that they will save money. International students, whose parents are sending money often can look up our app and look up analyzed part of the app to find the right day to send money to their children studying abroad. Foreign Investor, can also look up our app and get easy data. Students doing study abroad can use this simple app and get easy data to get the currency exchanged. Online shoppers will benefit by looking up the exchange rate as well as the historical data to find out right time for the shopping.

There are several other similar applications in the android market. Our application provides *analyzed historical data with graph representation which makes different* from other application in the market. This helps users save time and energy because they don't have to do analysis.

Application Design

Our application has two Activities in total. Therefore, there are just two main views, but they are very powerful, as we implement fragments.

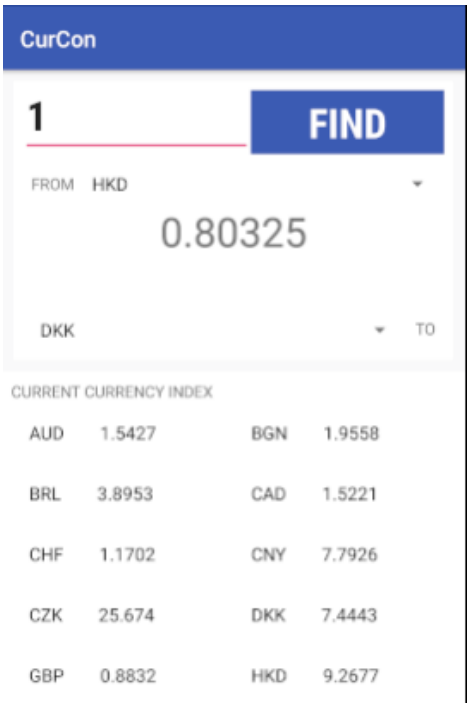
Main Activity

1. Currency Converter
2. List of today's Exchange rate

Individual Activity

1. Historical Analysis of Currency
2. Graph Analysis
3. User Interface for Comparisons

These are pretty much what we have shown. However, this design looks complicated once we look at the fragment level.



Below presented is the Main Activity that does simply how we designed our app in our design phase. There are two modules working their own separate work. (1) The upper module does the currency conversion, while the (2) second module lists out all the currencies based on today's exchange rate.

MainAcivity governs the conversion module and the list module. The topmost fragment is modelled as **ConversionFragment** (as its name suggests is a fragment that is responsible for currency converstion), and below is the **ItemFragment** (as its name suggests is a fragment responsible of containing Items, which in our case is the RecycleView holder for list of currencies exchanges as of today.

Each of these currencies come from fixer.io[1] API. These currencies are directly mapped from their /latest endpoint.

We do not just list these currencies so people can know the currency values. However, we have added a feature when some user clicks on one of those currencies, which is that they will be

[1] http://fixer.io

directed to the other activity, known as **IndividualActivity**. This activity, as its name suggests, is responsible to display Individual currency's information.

For instance, given that a user clicks Australian Dollars, they would go to this activity.

This activity has 3 main components that have their own function.

[1] IndividualInfoFragment is the topmost fragment solely responsible for displaying the information for that currency. It is interesting to understand the development of this fragment. This fragment accepts information of currency through one of its methods and displays it.

[2] **NinetyDaysCurrencyGraphFragment** is responsible for handling all the graph related activities happening in its parent activity (IndividualActivity).

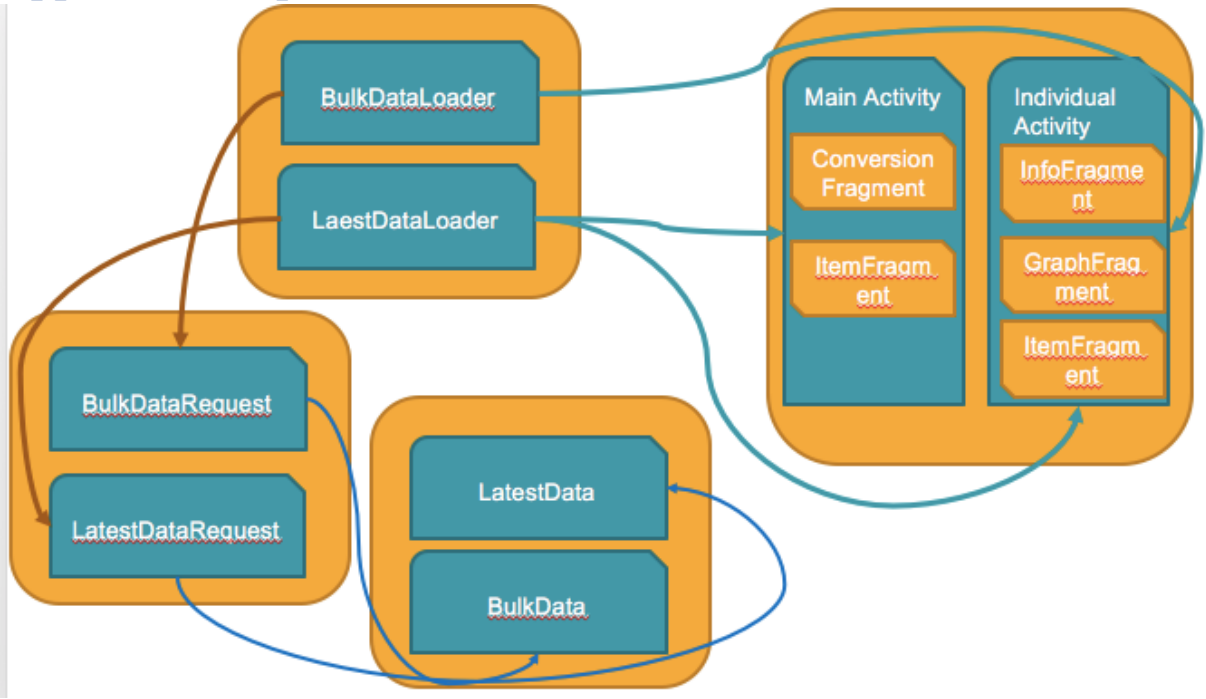
[3] **ItemFragment** is the reusable fragment used in the **MainActivity** as well. However, the role here is different. When some of the currencies are clicked, the clicked currencies are then *displayed in the graph.* This feature was designed because we wanted our users to explore the comparisons of currencies in the graph.

For instance, in the diagram on the left, it can be seen that these two currencies, **AUD** and **CAD** are compared. They are shown in the graph. The user can select as much feature

It is amazing to know that we have another feature that lets user select the range of days they want to see the rates for the currencies comparisons. For instance, we have this **seekbar** which we allow user to move, and based on how far the user has moved, it will show the days of comparisons.

In the diagram, it shows for 90 days.

Application Implementation and Evaluation



The above diagram represents the relation between the components in our app.

BulkData is a data that is downloaded from the *European Central Bank* from this link[2]. This data is downloaded via **BulkDataRequest**, an **Asynctask** implementation.

LatestData is a data that is downloaded from API call from fixer.io. This data is downloaded by **LatestDataRequest**, an Asycntask implementation.

MainActivity or IndividualActivity, according to what they need (as represented by the arrow), implements **Loader** for these above data. When these data are loaded, these activities are notified of the data retrieval process completion task, and are provided with the data.

For instance, MainActivity

1. calls the data retrieval procedure `DataLoader.processLatest()` from onCreate() method.
2. This internally calls Asyntask, and puts the interface LatestDataLoader, whose implementation is the MainActivity, as the interface to the AsynTask.
3. In the onPostExecute() method, it calls the LatestDataLoader method, onDataLoaded().
4. OnDataLoaded() procedure then calls the fragment it needs to, and then loads up the data for the application to show.

The similar design is implemented in the IndividualActivity as well. However, Individual Activity also requires the BulkData (the data that contains 90 days of currency history), we implement another interface **BulkDataLoader** to do the job. It works in the similar fashion.

---

[2] https://www.ecb.europa.eu/stats/eurofxref/eurofxref-hist-90d.xml?92ee9aee51657ebcf9391d406a54976e

We rely on our own XML parser called **BulkParser** that parses the data we got from XML, and then create Concrete Data Object.

Where do we store the data? Since the data is around 40Kilobytes, we downloaded XML every time the app starts. However, we wanted to optimize it, and we then use the reference of this XML data until the app is then closed. We store it in the HashMpap, whose key is the currency, and the value is the currency's value for 90 days (stored in ArrayList). In other words, we cache the data.

The graph is displayed in the **LineGraphView,** view that extends SurfaceView. Here, we call a method onDraw() that then calls the method run() in the **LineGraphDrawer**. This draws all the work. This class designs its own algorithm to scale, point out x coordinate, y coordinate and figure out what color to show. Based on this information they draw.

One of the reasons why we implemented Fragment is because of its ability to communicate with the parent activity. Since we need a constant communication between the **ItemFragment** that contains each currency and the **LineGraphView** holding fragment, we have them as fragment. When one of the items is clicked, all the information is then send to the other fragment to draw.

Testing:

We tested on our own hands for the functional test. However, we did quite some unit testing in order to implement parser for the XML as well as the JSON. The reason for this is it was too time consuming to restart the app in order for these parsers to develop.

We also had to test multiple times how the graph plot would work. In order to do this, we created a fake dummy data which were point and then fed to the app.

But the testing is not implemented well. Once the feature what we needed ran, we did not bother any.

Functionality:

We expect our app to work well. We included the feedback from the class to add a value EditText, where user can type the currency value they want for conversion. We expect the graph should work too.

Performance:

We expect our app to perform well. There should not be very rendering, as we used the technique of inheritance to store the point data. However, one of the things we did to compensate our performance is that we had some TextView display "Graph is Loading" to replace the hidden work of downloading the XML from the ECB's website.

Errors Identified:

There were many errors that were identified throughout the project. For instance, we had to replace ListView with the RecycleView. One of the things we wanted to go in GUI for MainActivity is that we wanted make top fragment responsible for conversion. However, when someone slides down, we wanted our app to show the list of currencies. It wasn't possible in the list view.

The point drawing in the surface view required quite attention. The scaling, x-coordinate and y-coordinates were all designed carefully. Our graph is shown from bottom to top, but surfaceView canvas expects the coordinate starts from top-left.

Also, XML parsing was quite of a work too. Figuring out how it is done to how we should implement it was quite of a work. We were able to solve.

We however have not been able to show the native currency icon for each of the currencies. We also expect to fix the issue that when the currency is selected for conversation, we reset the currency value to 1. Then only the user is able to change this value and then do the currency change.

References

We used Currency Conversion based on http://fixer.io. They take data from European Central Bank site. All currencies are displayed by http://api.fixer.io/latest. We got historical data from https://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/index.en.html. There were two basic learning tools to build this app. They were android documentation which we looked at:
https://developer.android.com/reference/org/w3c/dom/Document.html

Another learning tool was from class material. As always google helped up to find the proper API.

Experiences and Thoughts

Overall, we collected great experiences by developing this android application. Team chemistry between us was fabulous as we did our parts respectively and combined our work to make one finished project.

It was interesting about this app. The initial project was just a conversation. But, in order to implement more as professor had expected later, we had to have quite a decent job. It was never

an easy task to be able to have many components communicate with each other. It was also challenging to create a graph. We also had to plan how so many components should work and communicate. We also had to give a thought on how we should design our app so that we can get performance advantage in this short amount of time.

So, in order to do these all, it took quite a work. However, because we had constant communication with each other, we gave ourselves a role, and we worked together. This was quite an amazing experience to learn not just android development, but also how we can code together.

Also, after creating this application we are motivated to make other application because we have many other ideas in our mind and we want to use those ideas to make other application too.

We really appreciate pushing us hard for creating this app.