CMPE 230 - Project 3 Documentation

Karahan Sarıtaş (2018400174)
Cahid Arda Öz (2019400132)

June 24, 2021

Contents

1 Problem Statement

We are asked to develop a QT program that displays conversion rate information about crypto-currencies. We extract the information from an API designed by coingecko.

2 Solution

Our solution consists of Main function and DataExtractor class. Within Main, we create our QT objects such as grid layout and main window widget. Then we create a DataExtractor object. DataExtractor reads a file consisting of the list of crypto currencies that will be displayed. It obtains the name of the file from an environment variable named MYCRYPTOCONVERT.

Then DataExtractor sends network requests to the API and receives information. We wait for each request to complete receiving a reply before continuing our procedure. Once a request has received its reply, DataExtractor stores the related information in a QString and proceeds with the next task.

After getting information from DataExtractor, we parse it within the Main function with the help of QRegularExpression and store the currencies in vectors. Then we create a grid and set the horizontal and vertical labels accordingly.

Lastly, we add numerical values to the grid and add the grid layout to the main window, completing the procedure.

3 Implementation

Our implementation consists of a main function in which we parse the information and create our grid, and a DataExtractor object that makes the http request to get currency-rate information.

1. Main Function
2. DataExtractor

We used necessary standard libraries and QT libraries such as QGridLayout, QApplication, QLabel, QStringList, QString, QtNetwork, QUrl and QRegularExpression.

First, within Main function we generate a QWidget,and QGridLayout. We create a DataExtractor object to get the currency-rate information. Using regular expressions, we parse the information QString into cryptos and corresponding currency values. After storing them in vectors, we fill the rows, columns and labels of the grid. Then we add the grid layout to the main window.

3.1 Data Extraction

As stated above, DataExtractor object is responsible for both reading the file obtained from environment variable MYCRYPTOCONVERT and sending the network requests to the API.

DataExtractor consists of a constructor, destructor, public slots and private fields. Public slots are namely replyFinished(), CoinsList(), getData() and getCryptos(). Private fields are two QNetworkAccessManager pointers, a vector of strings named as cryptos, and two QString objects to store the crypto-currencies and list of coins respectively.

1. DataExtractor(std::string os, QWidget *parent = 0): This is the constructor of DataExtractor object. It takes a string that represents the operating system. We get the information as to the OS within the main function and then create a DataExtractor object. Within the constructor, we also read the file that contains a list of crypto-currencies. Then two network requests are sent to the API and we wait for the responses. First request aims to fetch the list of coins available at coingecko. Second request aims to gather currency-rate information for necessary crypto-currencies. After receiving the responses, we store the information in strings, which are private fields of our object. Basically, constructor performs all important tasks for the extractor object.

2. DataExtractor::replyFinished(QNetworkReply *reply): This method stores the result of our second network request (crypto-currency rate information) to a string.

3. DataExtractor::CoinsList(QNetworkReply *reply): This method stores the result of our first network request (coins list) to a string.

4. QString DataExtractor::getData(): Getter function that returns the data received from the API.

5. std::vector<std::string>DataExtractor::getCryptos(): Getter function that returns the list of crypto-currencies to be displayed in the grid.

6. DataExtractor:: DataExtractor(): Destructor deletes the QNetworkManager pointer to avoid memory leaks.

3.2 Regular Expression

Within Main function, we parse the information received from the API by using QRegularExpressions. We used two different regular expressions to capture the crypto pattern (including the currency rate information), and to capture the currency pattern respectively.

1. cryptoPattern: `"\"([\\w-]+)\":{([^}]*)}"`

2. currencyPattern: `"\"([\\w-]+)\":(\\d+\\.?(e-?)?\\d*),?"`

After the parsing procedure is complete, we store the tokens in a 2D vector. Each row of the vector represent the currency information for one crypto-currency. Order of the crypto-currencies is the same with the order of crypto-currencies stored in the currency vector returned by the DataExtractor.

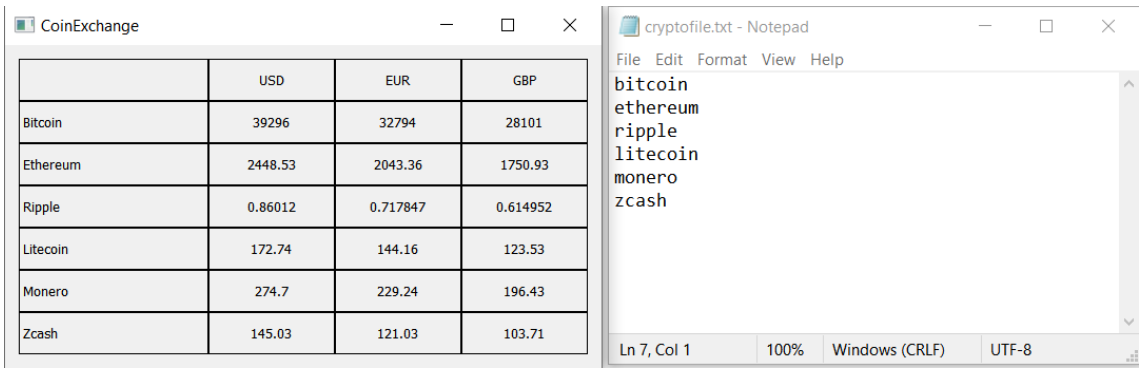
After storing the results we proceed with the next step, creating a grid with appropriate width and height settings.

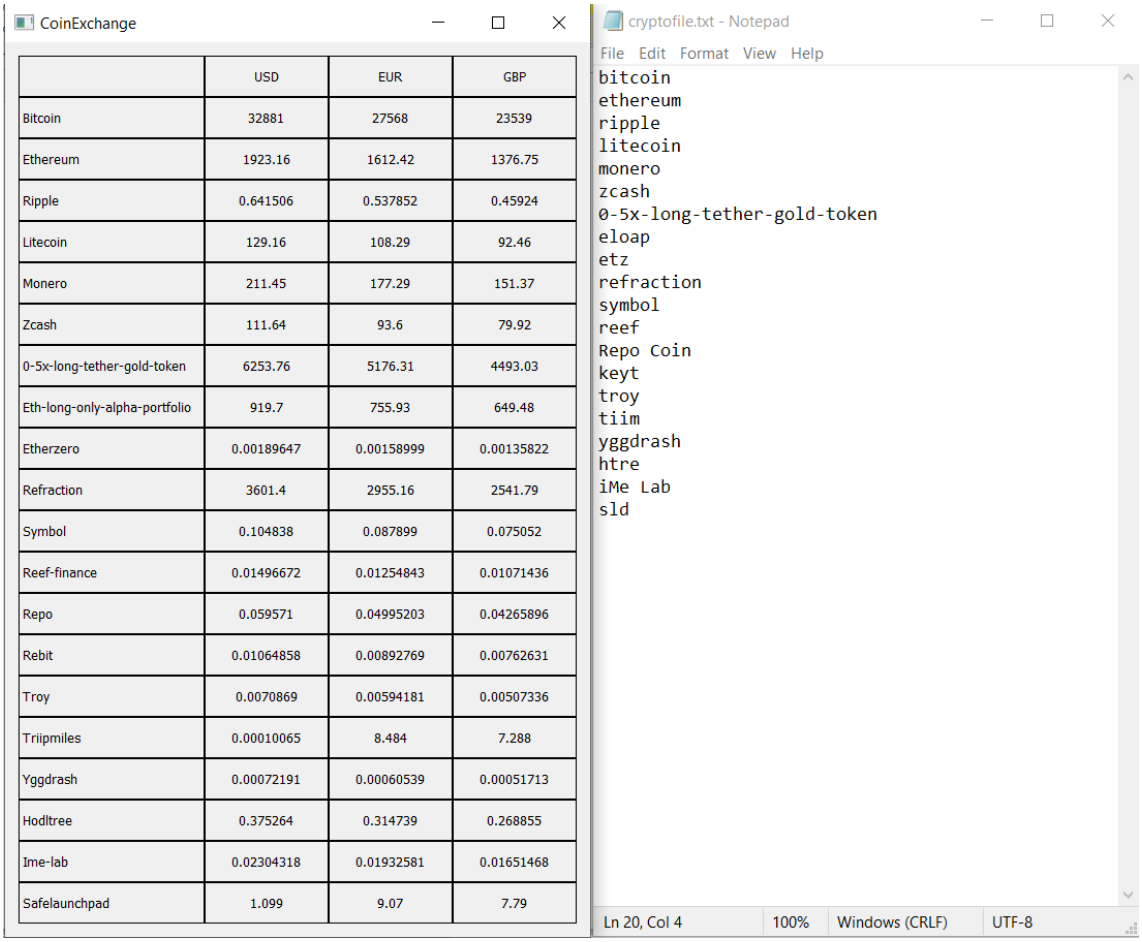
3.3 Grid Generation

To generate the grid layout, we iterate through our currency and crypto lists. Once the labeling is done, we can proceed with inserting data to the grid. To do so, we simply iterate through our 2D vector and fill the cells of the grid layout. Then we set minimum width and minimum height for columns and rows respectively.

Lastly, we add our grid layout to the main window by using .setLayout() method.

4 Examples





	USD	EUR	GBP
Bitcoin	32881	27568	23539
Ethereum	1923.16	1612.42	1376.75
Ripple	0.641506	0.537852	0.45924
Litecoin	129.16	108.29	92.46
Monero	211.45	177.29	151.37
Zcash	111.64	93.6	79.92
0-5x-long-tether-gold-token	6253.76	5176.31	4493.03
Eth-long-only-alpha-portfolio	919.7	755.93	649.48
Etherzero	0.00189647	0.00158999	0.00135822
Refraction	3601.4	2955.16	2541.79
Symbol	0.104838	0.087899	0.075052
Reef-finance	0.01496672	0.01254843	0.01071436
Repo	0.059571	0.04995203	0.04265896
Rebit	0.01064858	0.00892769	0.00762631
Troy	0.0070869	0.00594181	0.00507336
Triipmiles	0.00010065	8.484	7.288
Yggdrash	0.00072191	0.00060539	0.00051713
Hodltree	0.375264	0.314739	0.268855
Ime-lab	0.02304318	0.01932581	0.01651468
Safelaunchpad	1.099	9.07	7.79

5 Conclusion

Looking at the visual material we get at the end, we conclude that our program solves the given problem. We extracted the information correctly and generated a QT grid as stated in the description.