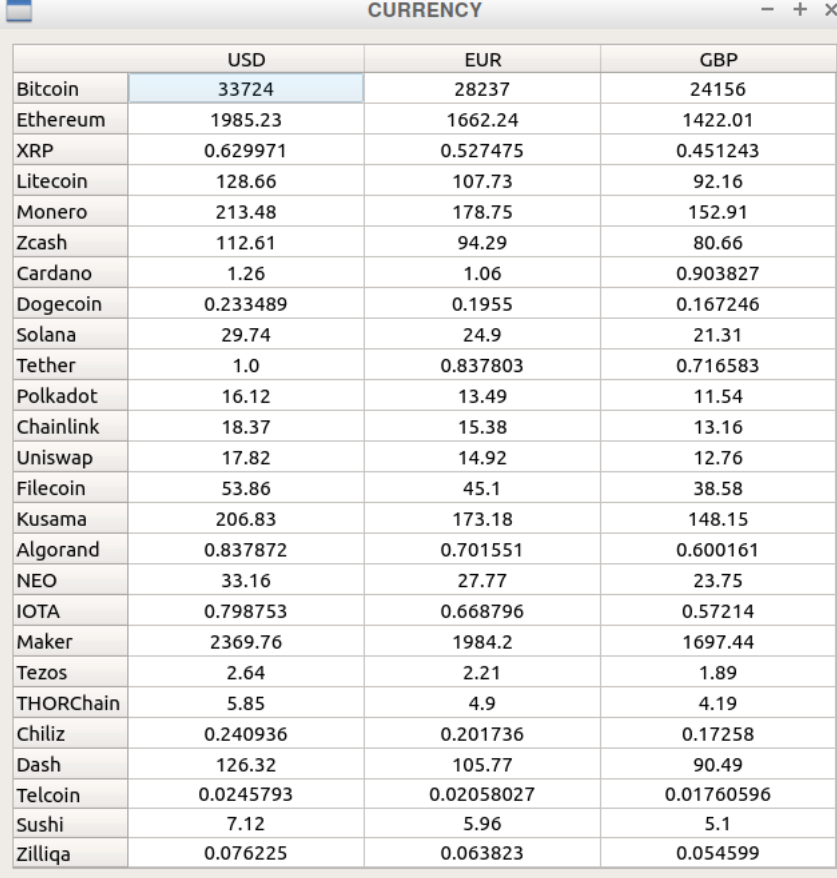


## CMPE 230 – Project 3 Report

Halil Burak Pala – 2019400282

### Overview

In this project, we are required to implement a simple QT program that displays conversion rate information about cryptocurrencies. At the end of the implementation, this project has a view like this:



	USD	EUR	GBP
Bitcoin	33724	28237	24156
Ethereum	1985.23	1662.24	1422.01
XRP	0.629971	0.527475	0.451243
Litecoin	128.66	107.73	92.16
Monero	213.48	178.75	152.91
Zcash	112.61	94.29	80.66
Cardano	1.26	1.06	0.903827
Dogecoin	0.233489	0.1955	0.167246
Solana	29.74	24.9	21.31
Tether	1.0	0.837803	0.716583
Polkadot	16.12	13.49	11.54
Chainlink	18.37	15.38	13.16
Uniswap	17.82	14.92	12.76
Filecoin	53.86	45.1	38.58
Kusama	206.83	173.18	148.15
Algorand	0.837872	0.701551	0.600161
NEO	33.16	27.77	23.75
IOTA	0.798753	0.668796	0.57214
Maker	2369.76	1984.2	1697.44
Tezos	2.64	2.21	1.89
THORChain	5.85	4.9	4.19
Chiliz	0.240936	0.201736	0.17258
Dash	126.32	105.77	90.49
Telcoin	0.0245793	0.02058027	0.01760596
Sushi	7.12	5.96	5.1
Zilliqa	0.076225	0.063823	0.054599

My implementation can be investigated in four parts:

1. Getting coin ids, names or symbols from an environment variable named "MYCRYPTOCONVERT".
2. Designing the UI of the program through QT.
3. Sending a request to and getting a response from a web server to get names, ids and symbols of these coins.
4. Sending a request to and getting a response from a web server to get conversion rates of these coins.

### General Structure

This project contains four files:

1. **main.cpp**: This file runs the application.
2. **myclass.h**: This file contains definitions of the class which contains the layout of the UI and handles the http requests.
3. **myclass.cpp**: This file contains the implementation of the class which contains the layout of the UI and handles the http requests.

4. **project.pro:** This project file is for qmake. It creates the necessary Makefile to run this program.

The Qt class which contains the table and handles http requests is named MyClass. It is a child of Qt's QWidget object. It has three public fields, one constructor, two slots and three private fields. At the end of the implementation, their general structure is expected to become as follows:

**vector<string> cryptos:** This vector contains the names, symbols or ids of the coins which are given in the file which is pointed by MYCRYPTOCONVERT.

**QVector<QString> \*ids:** This vector contains the ids of the coins. After sending the request to the corresponding API, we will get ids and names of the coins to use them in our project.

**QVector<QString> \*names:** This vector contains the names of the coins.

**QTableWidget \*tableWidget:** This is the table widget which will show all the names of the coins and their conversion rates to USD, EUR and GBP.

**QNetworkAccessManager \*idGetter:** This network manager is for getting all of the ids of the coins to send a http request to API which gives conversion rates and getting all of the names of the coins to show their name in the vertical header of the table.

**getIds(QNetworkReply \*reply):** This slot is connected to the network manager "idGetter". This slot does the corresponding action of the idGetter.

**QNetworkAccessManager \*manager:** This network manager is for getting the prices of all coins.

**replyFinished(QNetworkReply \*reply):** This slot is connected to the network manager "manager". This slot does the corresponding action of the manager.

## Implementation

First part of the implementation is getting the list of coin names, symbols or ids from an environment variable "MYCRYPTOCONVERT". In my implementation, the file which this variable points to can include name, symbol or ID of the coin, it does not matter. To get this coins from that file, I used the C++ method get\_env(). By the following lines of code that are in main.cpp, I got the file name that this variable points to and received the names, symbols or ids from this file and put them into a vector named cryptoslist:

```
const char* env_p = getenv("MYCRYPTOCONVERT");
ifstream input(env_p);
string line;
vector<string> cryptoslist;

while(getline(input, line)){
    cryptoslist.push_back(line);
}
```

After getting the coin names, symbols or ids, I should have get ids of all of the coins to send a http request to the coingecko API in the following way:

**[https://api.coingecko.com/api/v3/simple/price?ids={id's of the coins}&vs\\_currencies=usd,eeu,gbp](https://api.coingecko.com/api/v3/simple/price?ids={id's of the coins}&vs_currencies=usd,eeu,gbp)**

where I should replace **{id's of the coins}** with ids of all of the coins separated by commas. Also, I should have got names of all of the coins to place them into the vertical header of my table. To get them, I should have sent a http request:

**<https://api.coingecko.com/api/v3/coins/list>**

after which I get names, symbols and ids of all of the available coins in the database of coingecko. So, in the myclass.cpp, after setting and initializing the table, I sent a http request <https://api.coingecko.com/api/v3/coins/list> and got the content of this website which includes all of the coin symbols, names and ids in the following way: {"id":"bitcoin","symbol":"btc","name":"Bitcoin"}. After getting the list of this coins, I searched this list by help of following RegEx pattern:

`\{"id\":"([^\"]+)\","symbol\":"([^\"]+)\","name\":"([^\"]+)\\"}`

whose **first group** is for searching the id, **second group** is for searching the symbol, and the **third group** is for searching the name of the coin. In the following part of the code, I respectively replaced first, second, and third groups with the string given in the file and search for a pattern, then after getting names and ids, put them into the vectors ids and names:

```
string symbolPtrn = "\\{\\\\"id\\\\":\\\\"([^\"]+)\\"\\",\\\\"symbol\\\\":\\\\"([^\"]+)\\"\\",\\\\"name\\\\":\\\\"([^\"]+)\\"\\"}";
QString symbolPattern = QString::fromStdString(symbolPtrn);
QRegExp symbolrx(symbolPattern);
if(symbolrx.indexIn(data,pos) != -1){
    id = symbolrx.cap(1);
    symbol = QString::fromStdString(symbolrx.cap(2));
    name = symbolrx.cap(3);
} else{
    string namePtrn = "\\{\\\\"id\\\\":\\\\"([^\"]+)\\"\\",\\\\"symbol\\\\":\\\\"([^\"]+)\\"\\",\\\\"name\\\\":\\\\"([^\"]+)\\"\\"}";
    QString namePattern = QString::fromStdString(namePtrn);
    QRegExp namerx(namePattern);
    if(namerx.indexIn(data,pos) != -1){
        id = namerx.cap(1);
        symbol = namerx.cap(2);
        name = QString::fromStdString(namerx.cap(3));
    } else{
        string idPtrn = "\\{\\\\"id\\\\":\\\\"([^\"]+)\\"\\",\\\\"symbol\\\\":\\\\"([^\"]+)\\"\\",\\\\"name\\\\":\\\\"([^\"]+)\\"\\"}";
        QString idPattern = QString::fromStdString(idPtrn);
        QRegExp idrx(idPattern);
        if(idrx.indexIn(data,pos) != -1){
            id = QString::fromStdString(idrx.cap(1));
            symbol = idrx.cap(2);
            name = idrx.cap(3);
        }
    }
}
ids->push_back(id);
names->push_back(name);
```

After getting names and ids, I first created vertical header of the table with the names of the coins, and then created the url of the request which will give the prices of the coins with ids of the coins. Then send the request to the API to get the prices:

```
// Here, I constructed my actual vertical header after getting the names of the coins.
// Names are shown in the vertical header.
QStringList verticalHeaderElements;
for(QString name : *names){
    verticalHeaderElements.append(name);
}

tableWidget->setVerticalHeaderLabels(verticalHeaderElements);

// Here, I created the part of the string of the http request which contains coin ids.
QString requestline = ids->at(0);
for(int i = 1 ; i < ids->size() ; i++){
    requestline = requestline + "," + ids->at(i);
}

// Here, I created my full url for http request to get the prices of the coins.
QString urlpart1 = "https://api.coingecko.com/api/v3/simple/price?ids=";
QString urlpart2 = "&vs_currencies=usd,eur,gbp";
QString url = "";
url.append(urlpart1);
url.append(requestline);
url.append(urlpart2);
// Then sent the request.
manager->get(QNetworkRequest(QUrl(url)));
```

After this step, now I was ready for getting the prices of the coins. To do that, I followed a very similar way. In the response of the API, structure of a price element is as follows:

```
"bitcoin":{"usd":33252,"eur":27879,"gbp":23810}
```

To get these elements, I got help from RegEx again, by help of a RegEx expression which gives prices of the coin in USD, EUR and GBP, I got these prices and place them to the corresponding places in the table. The code of this part is as follows:

```
for(QString id : *ids){  
    // By help of RegEx, I get all the prices of each coin.  
    QString pattern = "\\\""+ id + "\\":\\\\"usd\\":(\\d\\.\\.\\.\\d+e[+-]\\d+|\\d+)+"  
    |"\\.\\.\\.\\d+|\\d+)\\\",\\\"eur\\":(\\d\\.\\.\\.\\d+e[+-]\\d+|\\d+\\.\\.\\.\\d+|\\d+),\\\"gbp\\":(\\d\\.\\.\\.\\d+e[+-]\\d+|\\d+\\.\\.\\.\\d+|\\d+)";  
    QRegExp rx(pattern);  
    QString usdStr, eurStr, gbpStr;  
    if (rx.indexIn(data, pos) != -1 ) {  
        usdStr = rx.cap(1); // USD price  
        eurStr = rx.cap(2); // EUR price  
        gbpStr = rx.cap(3); // GBP price  
    }  
    else {  
        // If the API does return something like "12866-lauder":{} ", it means API does not return prices.  
        // So, in that case instead of price, I showed "NULL" instead of price.  
        QString pattern = "\\\""+ id + "\\":\\{\\}\\}\";  
        QRegExp rx(pattern);  
        if (rx.indexIn(data, pos) != -1 ){  
            usdStr = "NULL"; // USD price  
            eurStr = "NULL"; // EUR price  
            gbpStr = "NULL"; // GBP price  
        } else{  
            // In case of an error, string "Error" is displayed.  
            usdStr = QString("Error");  
            eurStr = QString("Error");  
            gbpStr = QString("Error");  
        }  
    }  
  
    // According to fetched prices, I set the corresponding price entries of our table.  
    QTableWidgetItem *usdItem = new QTableWidgetItem();  
    usdItem->setText(usdStr);  
    usdItem->setTextAlignment(Qt::AlignCenter);  
    tableWidget->setItem(k,0,usdItem);  
  
    QTableWidgetItem *eurItem = new QTableWidgetItem();  
    eurItem->setText(eurStr);  
    eurItem->setTextAlignment(Qt::AlignCenter);  
    tableWidget->setItem(k,1,eurItem);  
  
    QTableWidgetItem *gbpItem = new QTableWidgetItem();  
    gbpItem->setText(gbpStr);  
    gbpItem->setTextAlignment(Qt::AlignCenter);  
    tableWidget->setItem(k,2,gbpItem);  
  
    k++;  
}
```

To set the UI of this program, I created a `QTableWidget` object, put it into a layout and stretch it so that it expands to all window.

## Edge Cases, Error Handling and Problems

For some coins, API does not return any price information. In those cases, I search the response again by help of RegEx if such situation occurs and if it is the case, price information of that coin is appeared to be "NULL" in the table.

Name of some coins include some special characters which are used in regular expressions, such as “\$”. These coins’ names, ids and symbols cannot be extracted from the API’s coin list because of deficiencies in the regular expressions. So, their name cannot be displayed in the table, and hence their prices cannot be seen also. In this project, it is assumed that such coins will not be given as testcases.

Another issue is that if the coin list in the MYCRYPTOCONVERT is more than about 300 or 400 lines, when I try to extract prices of these coins, API gives a “400 Bad Request” error since request header becomes too large. In that case, names of the coins can be displayed but their prices cannot be extracted. Instead of prices, string “Error” is displayed. In this

project, it is assumed that file that will contain coin names will not be that large so that coin prices can be displayed.

### **Running The Program**

The zip folder of the project contains this pdf document and a folder named *230project3*. This *230project3* folder contains the project file *230project3.pro* which creates the *Makefile* to run this program. After navigating to this folder *230project3*, we can run this program by terminal in an Ubuntu machine by applying following commands respectively:

1. `qmake`
2. `make`
3. `./230project3`

Before running the program, please make sure that there is no file other than *myclass.cpp*, *myclass.h*, *main.cpp* and *project.pro* in the folder.

### **Conclusion**

In this project, I implemented a basic app which shows some crypto currencies and their prices in USD, EUR and GBP. By help of this project, I gained some experience in implementing UIs via QT, sending requests to and getting responses from APIs, and reaching some paths that are specified by some environment variables in the computer via C++.