Department of Electronic and Telecommunication Engineering

Internet of Things

EN3250

Currency Converter Service

Project Report

University of Moratuwa

170258L : R.H.R. Jayarathne
170543G : M.K.T. Sampath
170698J : L.T.A. Wijayaratne
170375R : D.R. Marasinghe

This report is submitted in partial fulfilment of the requirements for the module
EN3250 - Internet of Things

Contents

1 Introduction **1**

1.1 Overview 1

1.2 Objectives 2

1.3 Scope 2

2 Methodology **3**

2.1 Architecture 3

2.2 Functionalities of NodeMCU 4

2.2.1 Initial setup 4

2.2.2 Hosting Web Interface for User 6

2.2.3 Authentication and authorization flow 6

2.2.4 Setting up the MQTT Broker 7

2.2.5 Decoding compact received data 10

2.2.6 Sensing alerts and notifying the user 11

2.2.7 Supporting the LCD 16x2 display 11

2.2.8 Power Saving Mechanisms of the NodeMCU 11

2.3 Functionalities of the Node-RED Application 12

2.3.1 Extracting Data from the Exchange Rate API 12

2.3.2 Integrating Firebase 14

2.3.3 Obtaining technical analysis charts from prices 16

2.3.4 Creating the Node-RED dashboard 16

3 Results and Discussion **18**

4 Conclusion **18**

5 Annexes **19**

1 Introduction

1.1 Overview

Foreign exchange market is a global market, where people around the world buy and sell different foreign currencies everyday for various purposes. One purpose of the traders in the market is engaging in day-trading activities, and is performed by normal people who intend to gain profits and earn from small price mismatches that are available for short time periods. Day traders' activity is influenced by historical fluctuations of foreign currency prices as well as the information gained from several technical indicators. These indicators are generated by performing various mathematical transformations on the time series charts of currency prices. This information is of much value for Day Traders and they usually buy this market information from their trusted services.

This project focuses on providing foreign exchange market day-traders, a set of useful information, by using a currency API that provides live currency prices of various currencies around the world. Usually, historical currency price data are provided freely on a daily basis only, but these rates change by minute values, every few seconds. Day traders require these fast changing values and must pay a considerable price to access this information. Therefore, by storing the live currency data in a Firebase Realtime database, we are able to provide more information to Day Traders and other users.

The users of our application are able to log in using their email and password, and specify the currency type which they are interested in. Next, they are able to specify maximum (ceiling) and minimum (floor) values for the exchange rate (as a percentage of the exchange rate at the time of request) and receive notifications when the currency price crosses either of these thresholds. The notifications will be sent via emails as well as by a Buzzer, which would ring to notify the users. Therefore, the notifications would reach the user at times when he/she is offline as well, through the Buzzer ringing. These ceiling and flow values allow Day Traders to evaluate the performance of their investment over time, without having to monitor the variation of exchange rates manually.

An LCD screen displays the exchange rates of 6 types of currencies as stated below.

1. GBP - Great Britain Pound
2. JPY - Japanese Yen
3. USD - US Dollar
4. KWD - Kuwait Dinar
5. AUD - Australian Dollar
6. EUR - Euro

The display is refreshed every 30 seconds and shows whether each exchange rate has increased or decreased over the last 30 seconds.

Since the foreign exchange market is an international market, it operates 24 hours a day and five days a week. However, during the weekend the market closes its trading activities. The functioning of this application is turned off an hour after the trading activities are closed for a particular week and reopened an hour before the trading activities are resumed for the following week.

1.2 Objectives

This project aims at making exchange rates of a user's preference, freely available with updates at 30 second intervals. The overall platform with user authentication and live dashboards helps boost the activities currency markets.

- Inform the user when the investment has exceeded a certain percentage of profit (ceiling) or diminished below a certain percentage of loss (floor), even when he/she is offline via an alarm.
- This will allow them to make quick decisions and responses regarding their investments. To send detailed emails of the exchange rate behaviour at times when the thresholds have been crossed.
- Display historical variations of currency exchange rates and several technical analysis charts through live dashboards. These help the Day-Traders make useful predictions.
- To display rapid variations of currency exchange rates via an LCD display and whether the exchange rate has increased or decreased within the preceding 30 second interval.
- Ensure that the NodeMCU, and the resource constrained devices in this application operate in an energy efficient manner.

1.3 Scope

The scope of the project includes an application using IOT concepts, tools and standards to provide timely information to Day Traders. This provides the users with the following.

1. Exchange rate of 6 currencies - updated every 30 seconds
2. Buzzer and Email notifications when set thresholds for exchange rates are exceeded
3. A dashboard displaying the variations of exchange rate and Technical indicators over time (updated hourly).

The basic components of which the application comprises are given below.

1. **A realtime database** (using Google Firebase)
2. **An open-source API** providing currency exchange rates
3. **A Node-Red Dashboard** to display variations of the four technical indicators
4. **A Buzzer** to notify the user when the set thresholds for exchange rates have been exceeded
5. **An LED Display** to display exchange rate variations
6. **A Node-MCU** for accepting requests of the human users as well as to send control signals to a Buzzer and the LED Display
7. **A client mobile phone** as the user interface

The MQTT protocol is used to facilitate communication between NodeMCU and the Node-RED application. The deep sleep mode of the NodeMCU is used to save power of the NodeMCU and the other constrained devices, during the weekend hours when the forex market is closed.

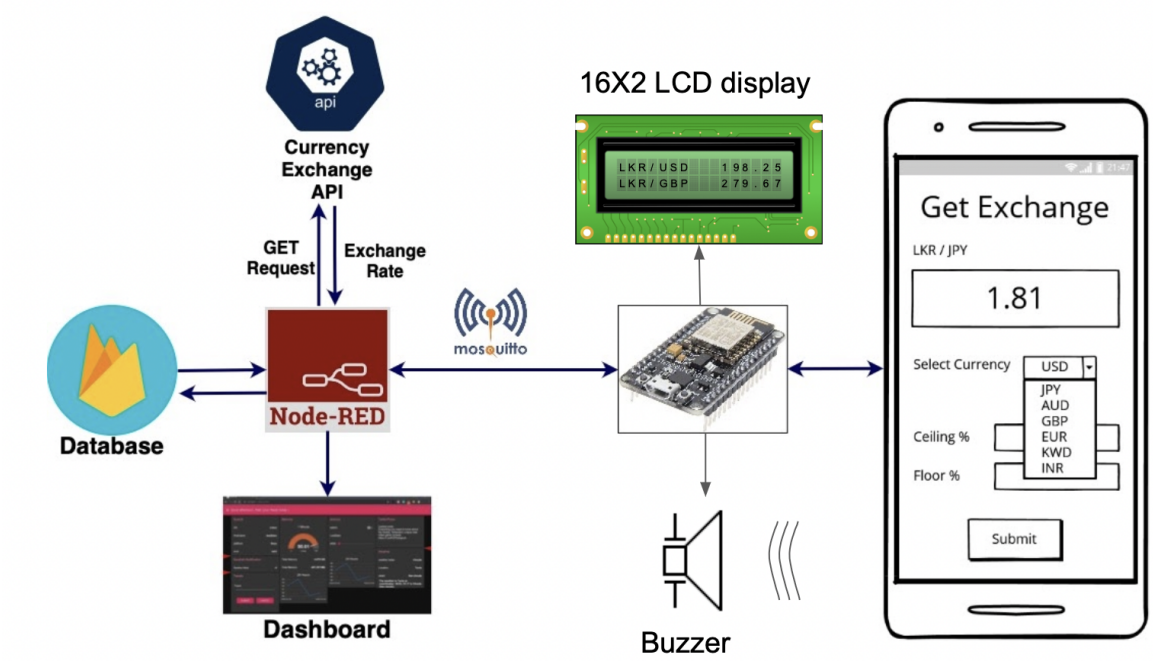


Figure 1: Architecture of the entire solution

2 Methodology

2.1 Architecture

The login credentials of the users of this application are entered to the database from the backend by the database administrator. The user is able to log in to the application using his/her mobile phone via a website hosted on a NodeMCU. The log in credentials are communicated to a Node-RED application via an MQTT broker and validated with the credentials stored in the Firebase database. On successful authentication, the Node-RED would publish a Success message token to admit the user.

Next, the user has the ability to select a currency of preference and specify the ceiling and floor thresholds. These values will also be communicated to the Node-RED application through a MQTT topic, and stored in the Firebase database.

A timer which is sending API requests at hourly intervals extracts the currency exchange rates and stores it in the database. The proposed architecture for this project requires exchange rate data every 30 seconds to accurately capture the minor changes in exchange rates, but the free version of the API only provides hourly exchange rate prices. We store these hourly exchange rates in the firebase database and use a gaussian random number generator to vary the hourly exchange rates by minor values to simulate rapid fluctuations of exchange rates similar to their real variations. The ceiling and floor values are checked against these simulated exchange rates at 30 second intervals, and if they are exceeded, an email is sent to the user. The exceeded message is sent to the NodeMCU via the MQTT broker and the Buzzer is sounded as well. This notifies the user in case he/she is offline.

The hourly exchange rate data is then used by the Node-RED application to calculate the following four

technical indicators.

1. Simple Moving Average (SMA)
2. Rate of Change oscillator (ROC)
3. Moving average convergence/divergence (MACD)
4. Relative Strength Index (RSI)

These indicators calculated using exchange rate data over 40 hours are plotted on a live dashboard.

2.2 Functionalities of NodeMCU

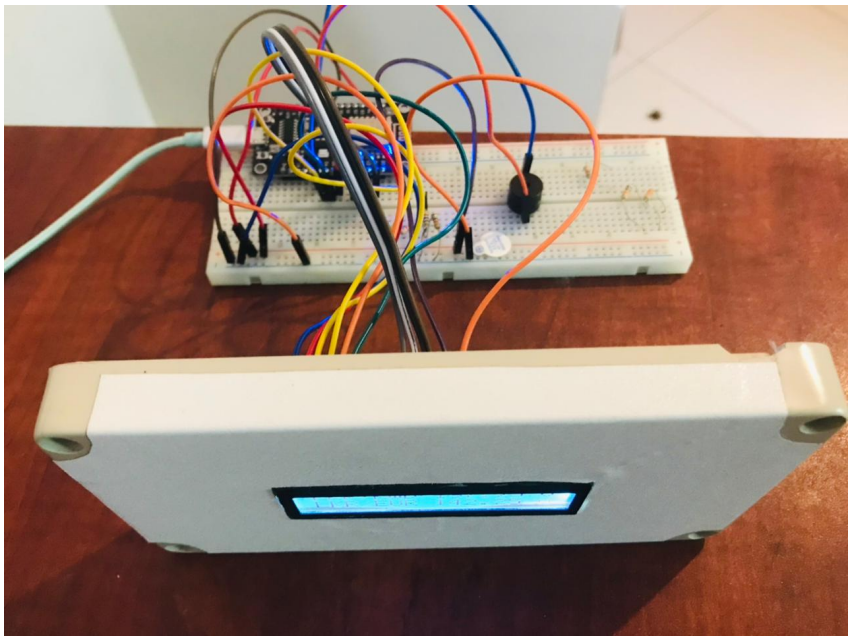


Figure 2: Setting up ESP8266 with an LCD display and a buzzer

The NodeMCU used in the IOT system performs multiple tasks to facilitate the connectivity between the user's device, the LCD display, and the buzzer. ESP8266 NodeMCU was used in the testing process. The flow chart in the figure 5, describes the functionality of the NodeMCU.

2.2.1 Initial setup

When the NodeMCU is powered up, it follows a sequence of steps to make sure it is connected to the internet. In order to implement this sequence, 'WiFi manager' library was used.
As the first step,it sets it up in *Station mode* and tries to connect to a previously saved Access Point.
In case it is unsuccessful, NodeMCU will act as an access point with the SSID 'IoT6B_G05' and create a web server (Default IP **198.168.4.1**). The user can connect to this access point through any WiFi enabled device and go to the above mentioned IP address using a web browser. The web browser will display the available WiFi access points near the user. By providing the WiFi password of the preferred access point, NodeMCU can connect to the access point. Now the NodeMCU is connected to the internet.

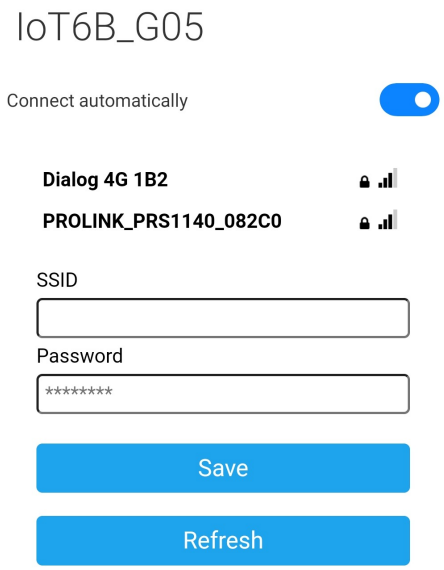


Figure 3: WiFi login interface for user

2.2.2 Hosting Web Interface for User

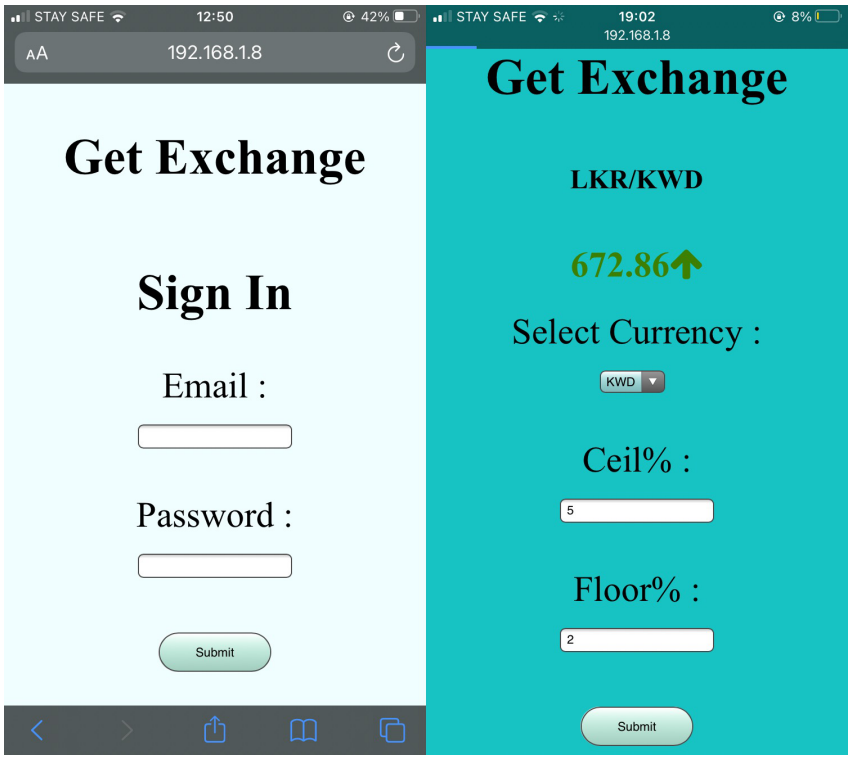


Figure 4: Login page and home page for the user

The above code calls two functions that return HTML scripts, which enable two HTML pages in figure 4 to be displayed. When the user loads **192.168.1.8** in the web browser, the login page on the left will appear in the mobile. After the authentication details are submitted, the user is redirected to the home page given that the authentication is successful.

In the home page. The user can select a preferred currency out of six available currencies. Additionally, the user can select two percentages to set the ceiling and floor prices compared to the current price. The current price of the selected exchange rate is then displayed on the home page.

2.2.3 Authentication and authorization flow

An authentication and an authorization flow is included in the IOT system. This flow is included in figure 5. The first step of authentication is the initial login page that requests the user to enter an email and a password. After obtaining the username and the password, NodeMCU sends authentication data to Node-RED through MQTT. Node-RED acts as an authorization server and generates an application access token. This access token is sent back to the NodeMCU through MQTT. NodeMCU then stores the access token along with the user name. Whenever NodeMCU is sending user specific data to Node-RED, the access token is also sent along with it to verify the validity of the user.

2.2.4 Setting up the MQTT Broker

The Mosquitto MQTT [1] server/broker was used to establish communications between the NodeMCU and the Node-RED application.The messages were published and received in the string data type and each component of the message was separated by a "$" sign for encoding. This allows several types of information to be transmitted using a single message.

The NodeMCU subscribes to four MQTT topics (a., b., c., and d. below), and publishes to two MQTT topics (d., and e.).

a. IOT_6B/G05/UserAuth

The login credentials of the user are sent to the Node-RED application for validation through this topic. The format of the published message is as follows.

$$\underbrace{1623921792}_{\text{timestamp}}\underbrace{\$user@gmail.com}_{\text{user\_email}}\underbrace{\$user\_pass}_{\text{password}}$$

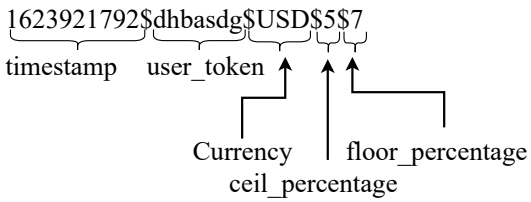
b. IOT_6B/G05/AuthResponse

The Nod-RED publishes the response status of the user authentication to this topic and is received by the NodeMCU. The format of the published message is as follows.

$$\underbrace{1623921792}_{\text{timestamp}}\underbrace{\$first.last@gmail.com}_{\text{user\_email}}\underbrace{\$success}_{\text{Status}}\underbrace{\$dhbasdg}_{\text{user\_token}}$$

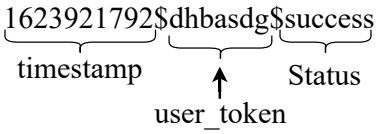
c. IOT_6B/G05/UserNeeds

The Nod MCU publishes the currency type subscribed to by a particular user, along with the ceiling and floor values requested for notifications. The format of the published message is as follows.



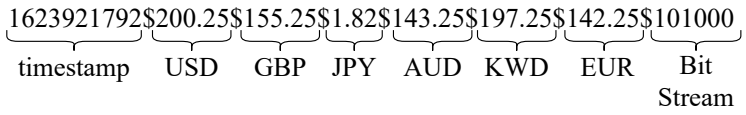
d. IOT_6B/G05/UserNeedsResponse

The response token from the Node-RED, stating the validity of the request message sent to the IOT_6B/G05/UserNeeds topic, is published to this topic. The format of the published message is as follows.



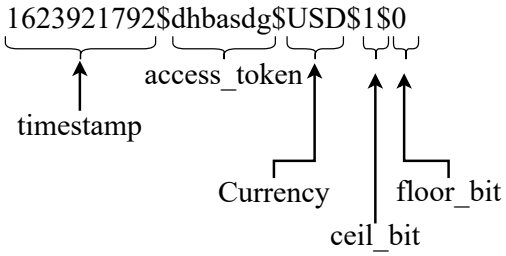
e. IOT_6B/G05/CommonData

This topic receives the exchange rates of all 6 currency types and is updated every 30 seconds from the Node-RED. These values are received by the NodeMCU, and displayed on the LCD screen. The exchange rate of the selected currency type is displayed on the web interface for the user. The format of the published message is as follows.



f. IOT_6B/G05/BuzzerNotification

Messages are published to this topic by the Node-RED application when the ceiling and floor prices, specified by the user, have been exceeded by the exchange rate. The format of the published message is as follows.



The components of the encoded MQTT messages are as follows.

- **timestamp** - Time at which the data is sent in Unix epoch format. This helps to validate timely data and to discard older messages.
- **ceil_bit** - A flag bit which is set to 1 if the ceiling price is crossed, else zero
- **floor_bit** - A flag bit which is set to 1 if the floor price is crossed, else zero

- **Bit stream** - Each of the 6 bits corresponds to the 6 currency types and is set to 1 if there is an increase in the exchange rate, 0 otherwise.
- **access_token** - Indicates that the application is authorized to access user's data [2]

MQTT transmissions occur at random times, and the inclusion of the timestamp in the messages facilitated them to be checked to identify old messages. Those which took more than 20 seconds for transmission are discarded. Combining the data using this encoding scheme was required, because several components of the data are used at a certain instance. Further, this method of encoding and transmitting data allowed the LCD display to be updated in one transmission.

This simple encoding scheme allows more data to be transmitted with little overhead which can help reduce the power usage of the NodeMCU.

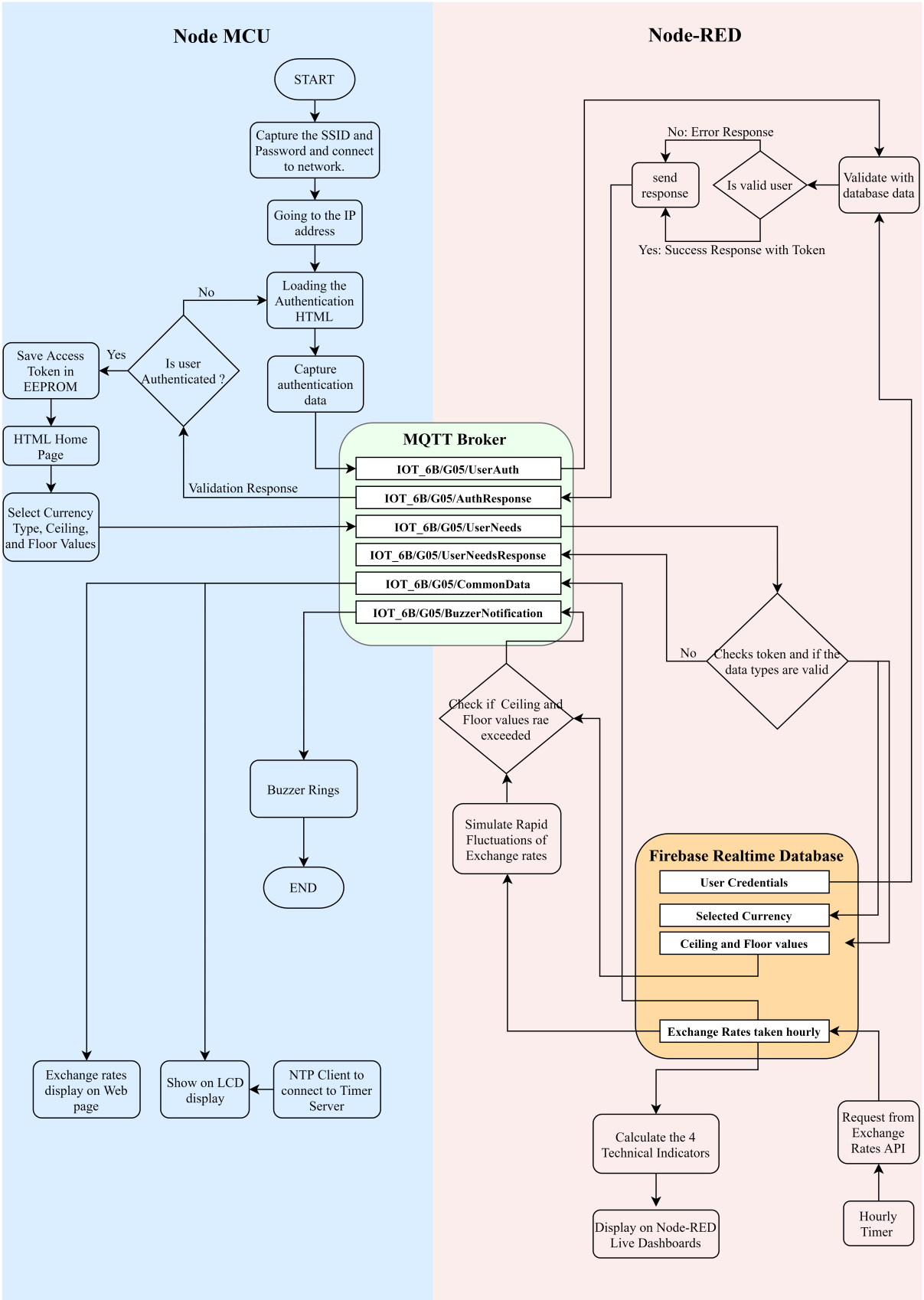


Figure 5: Complete technical process flow chart of the Methodology

2.2.5 Decoding compact received data

The goal here is to decode this large string containing several information components, separated using "$" signs and to assign each of them to separate variables in the NodeMCU, as shown below.

1623921792$200.25$155.25$1.82$143.25$197.25$142.25$101000

timestamp = 1623921792, USD = 200.25 , GBP = 155.25, JPY = 1.82, AUD = 143.25, KWD = 197.25 , EUR = 142.25

When a subscribed topic receives a data item, the **callback** function is called. The function considers the topic from which data is received and directs the data to the relevant purpose. For example, the **process_notification** function takes data from the **"IOT_6B/G05/BuzzerNotification"** topic, **process_Data** function takes data from the **"IOT_6B/G05/CommonData"**, etc. An extract of the **callback** function is given below.

```
void callback(char* topic, byte* payload, unsigned int length) {
  if (String(topic) == "IOT_6B/G05/BuzzerNotification") {
   process_notification(payload, length, 50, 5);
  }
  if (String(topic) == "IOT_6B/G05/CommonData") {
   process_data(payload, length, 70, 8);
  }
  :
  :
}
```

The **process_notification** function given below, splits the byte array into character arrays and then saves them in the variables treating them as separate cases, at every **callback** call.

```
void process_notification(byte* payload, unsigned int length, int charlen, int
    numitem) {

     int digit;
     payloadstr = "";
     Serial.println();
     for (int i = 0; i < length; i++) {
       payloadstr += (char)payload[i];
     }

     char payloadstr_array[charlen];
     payloadstr.toCharArray(payloadstr_array, charlen);

     char * token = strtok(payloadstr_array, "$");

     for (int i = 1; i < numitem+1; i++) {
        switch (i) {
         case 1:
```

```
            timestamp = atol(token);
            Serial.print(timestamp);
            Serial.println();
            break;
  :
  :
        }
        token = strtok(NULL, "$");
    }
}
```

2.2.6 Sensing alerts and notifying the user

When an alert is sent through the **"IOT_6B/G05/BuzzerNotification"** topic, the callback function instantly identifies a crossing in ceiling or floor prices, if any, and activates the **buzzer**. To identify notifications which have expired is essential, to ensure that the user is alerted based on timely information only. The NodeMCU was configured to be updated with the time and date so that it could be compared with the timestamp of received notification messages in unix epoch time format. A sample code is given below.

```
if (timestamp > unix_epoch - 19820) {
    current_user = String(token);
}
```

2.2.7 Supporting the LCD 16x2 display

Figure 6: LCD display showing currency values time and date

LCD display is used to show the live currency prices updated every 30 seconds, along with the date and time. These values are displayed one after the other, each being displayed for 2 seconds. The time and date was obtained using an available library (the NTPClient library) for NodeMCU, which enables it to communicate with an NTP (Network Time Protocol) Server.

2.2.8 Power Saving Mechanisms of the NodeMCU

The NodeMCU, along with the LCD display and the buzzer are power constrained devices. Hence power saving methods must be implemented to improve energy efficiency.

The main power saving method is by implementing the sleep function in the NodeMCU. When the Foreign exchange markets are closed during the weekends, the Node MCU enters the deep sleep mode to conserve power. Meanwhile, the dashboards from Node-RED are available for the user to monitor the exchange rate fluctuations. However, the maximum NodeMCU deep sleep time is 3 hours and 46 minutes [3] and if the NodeMCU is kept in deep sleep mode for too long, it may stay in the sleep mode forever. The deep sleep mode is scheduled to begin 15 minutes after the time at which the Forex market is closed and then enters a series of sleep-wake-sleep cycles until 15 minutes before the market is set to open.

15 minutes after the market closes, the NodeMCU is set to sleep for an hour. Then it wakes up, and checks whether the market is open or closed using the current time and date. If it is closed, the NodeMCU goes back to sleep for another hour. This is continued till the market opens. NodeMCU is set to keep sleeping every week from Saturday 1.45 a.m. to Monday 2.15 a.m [4].

The second power saving mechanism used in this application is the use of a simplified message encoding scheme in the form of a string of messages separated by "$" signs. The same can be done using messages in JSON format, but this would require an additional library to be installed in the ESP8266, as well as a much larger overhead for the messages which are transmitted. The reduction of the message size in the encoding scheme suggested in this project actively reduces the power consumed in data transmission.

2.3 Functionalities of the Node-RED Application

Node-RED is a flow-based development tool which was used in this project to implement the backend functionalities, database connectivity, and live dashboards for this project.

2.3.1 Extracting Data from the Exchange Rate API

The Exchangeratesapi.io API service has been used to obtain the exchange rates of the selected 6 currency types. The API service used for this application is updated every second with the exchange rates of many types of currencies. For our application we proposed that exchange rates are needed to be sampled at 30 second intervals but a paid account is required with a fee up to USD 80.00 per month to obtain exchange rates at such a high frequency, as shown in the figure 7.

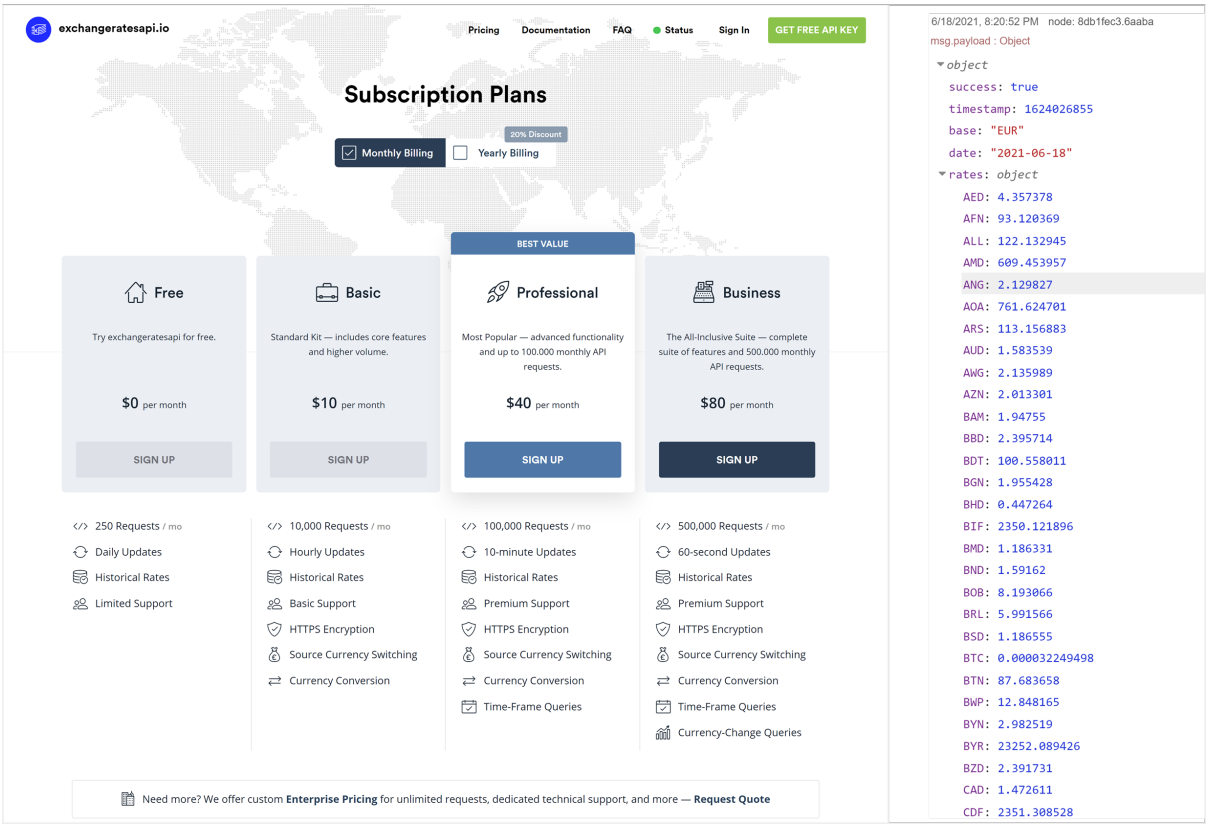


Figure 7: The subscription plans of the API and the Response array containing Exchange rates

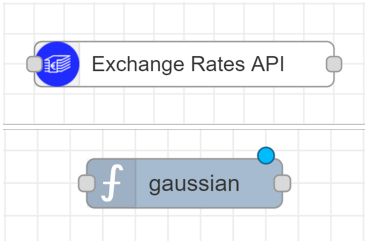


Figure 8: Exchange rate API node and Gaussian value generating node

Hence, for the purpose of designing a prototype for this application, we have used the free package which updates the exchange rates hourly. Further, we used a random number generator to vary the hourly rate at 30 second intervals so that only its 5th decimal place varies (which is similar to the real variations of exchange rates every 30 seconds) to simulate the actual variation of exchange rates. These were simulated using Node-RED, and the following packages were used.

a. node-red-contrib-exchangeratesapi [5]

The base currency and the currency of which the exchange rate is required needs to be specified to this node and it sends an API request to retrieve the required exchange rates. The response from this API containing a large collection of exchange rates is shown in figure 7, from which 6 currency types are selected for our application.

A timer was created using the “Inject” node with repeat enabled at hourly intervals to simulate this function.

b. node-red-contrib-acoustics [6]

The gaussian node from this package (check figure) takes the mean and standard deviation values as inputs and gives a random number as the output. This value was multiplied by the exchange rate in the previous hour and divided by 105to simulate the actual variations of exchange rates every 30 seconds.

A timer was created using the “Inject” node with repeat enabled at 30 second intervals to simulate this function.

2.3.2 Integrating Firebase

Figure 9: Firebase data structure

The data obtained hourly through API requests, is stored in a Firebase realtime database [7] for future use, as there are no free sources of historical exchange rate data for Day Traders. This functionality was implemented via Node-RED as well using the node-red-contrib-firebase [8] package.

Firebase is able to synchronize data to all its connected clients within milliseconds i.e., any change to the database would be reflected in its applications without the need for HTTP requests. Further, the applications hosted using firebase remain interactive, even when the devices are offline, and synchronizes with the database as soon as connectivity is established.

The Firebase realtime database is a cloud-hosted database, and one of the many products and services of Google. This is a NoSQL database and hence the data is stored in JSON format. Figure 9 shows the structure which was used to store the user data, user ceiling / floor values, and the exchange rates in the Firebase realtime database.

Three nodes from the node-red-contrib-firebase package have been used to connect the Firebase Real-time database to the Node-RED application, and they are as follows.

a. firebase.on

This node listens for changes in the database actively and in realtime and takes any changes to the Node-RED application. This was used to synchronize the ceiling and floor values in the database to the Node-RED application in this project.

b. firebase.once

This node allows us to read data from the database. This was used to extract the exchange rates over 40 hours when plotting the charts of the Node-RED live dashboard.

c. firebase.modify

This node is used to write or modify the data in the database using the Node-RED application. This was used when storing the authentication data and exchange rate data in the database.

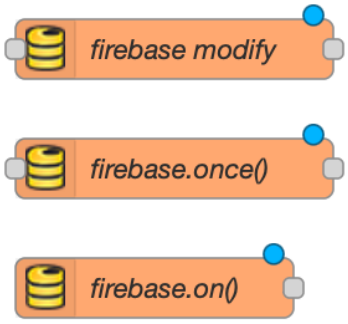


Figure 10: Firebase nodes used in the project

Following items are saved in Firebase,

- User credentials
- User inputs : selected currency, ceil, floor
- Fetched hourly foreign exchange rates

This is important because the stored data must be available in case of the following situations,

- In case the NodeMCU resets
- In case the Node-RED server restarts
- To store formatted data into database and fetch later

2.3.3 Obtaining technical analysis charts from prices

The hourly price data collected and stored in the real-time database can be sent through popular technical analysis functions to generate useful charts. These charts can be used to make valuable inferences. To process the price data available, an external library named TechnicalIndicators was used. Using the library, following indicators are calculated for hourly historical currency rates. Each currency will have four technical analysis charts.

1. **Moving average lines** are frequently used to smooth the fluctuations in a price chart (or a chart of any time series). A moving average is simply the mean of the last n closing prices.
2. **Rate of Change oscillator** (ROC) or momentum oscillator is calculated as 100 times the difference between the latest closing price and the closing price n periods earlier. Thus, it oscillates around zero.
3. **Moving average convergence/divergence** (MACD) oscillators are drawn using exponentially smoothed moving averages, which place greater weight on more recent observations. The "MACD line" is the difference between two exponentially smoothed moving averages of the price.
4. **Relative Strength Index** (RSI) is based on the ratio of total price increases to total price decreases over a selected number of periods. This ratio is then scaled to oscillate between 0 and 100.

2.3.4 Creating the Node-RED dashboard

The exchange rate data obtained from the API were analyzed using Node-RED functions to yield the above values of technical indicators over a period of 40 hours and plotted using a Node-RED dashboard as shown in the figure 11,12,13 and 14.

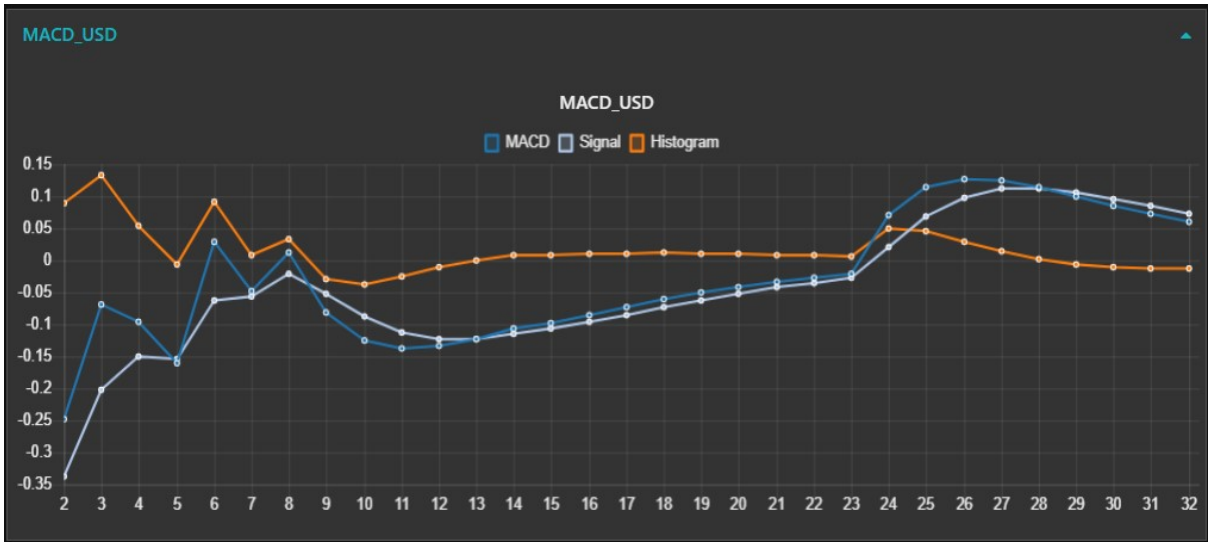


Figure 11: MACD indicator in Node-RED dashboard

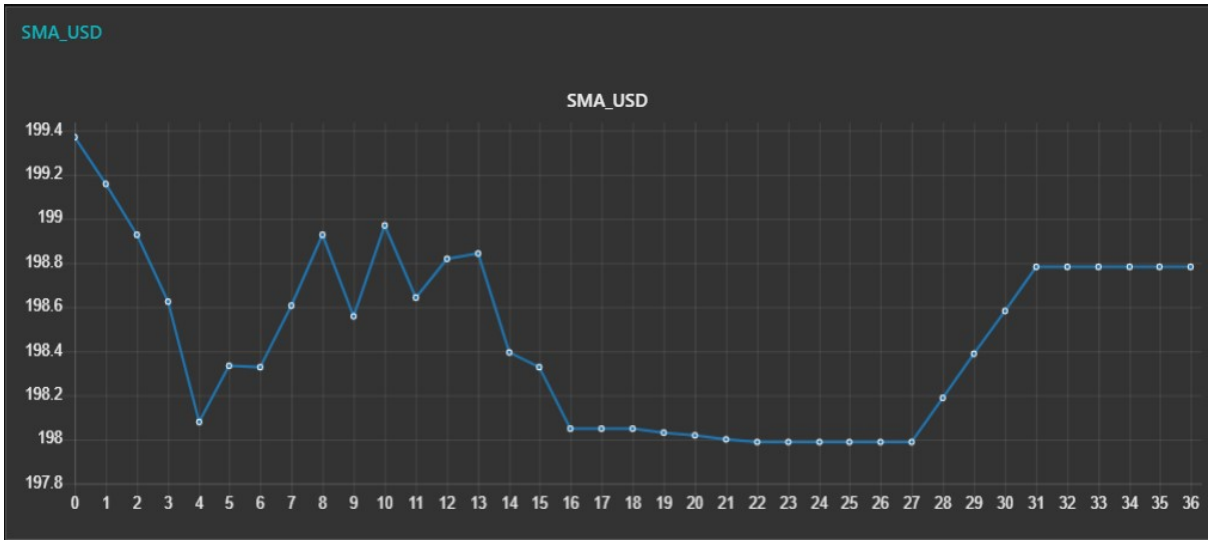


Figure 12: SMA indicator in Node-RED dashboard

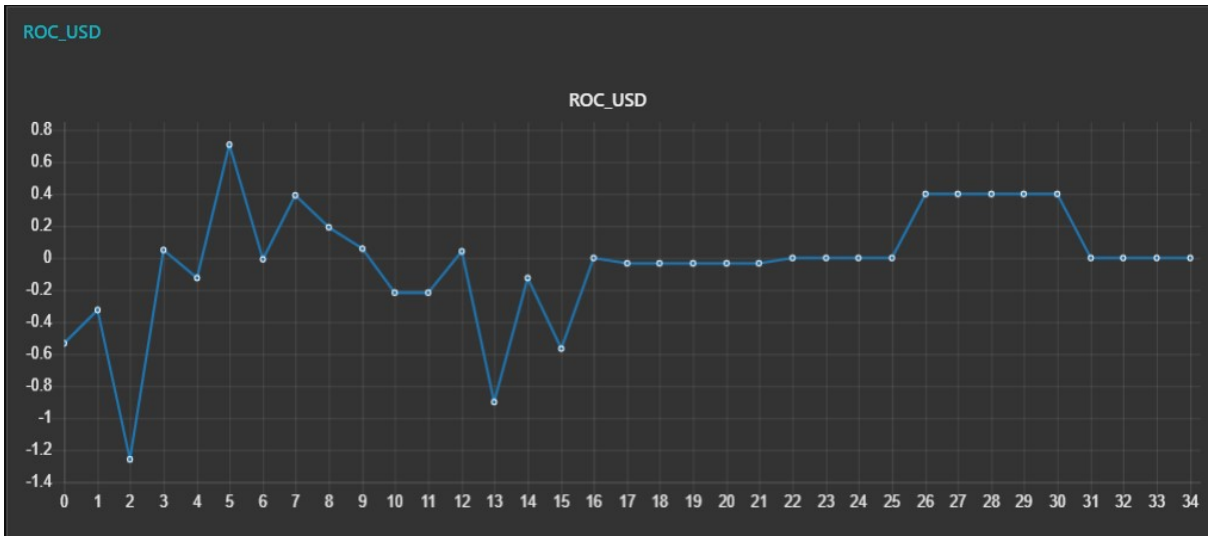


Figure 13: ROC indicator in Node-RED dashboard

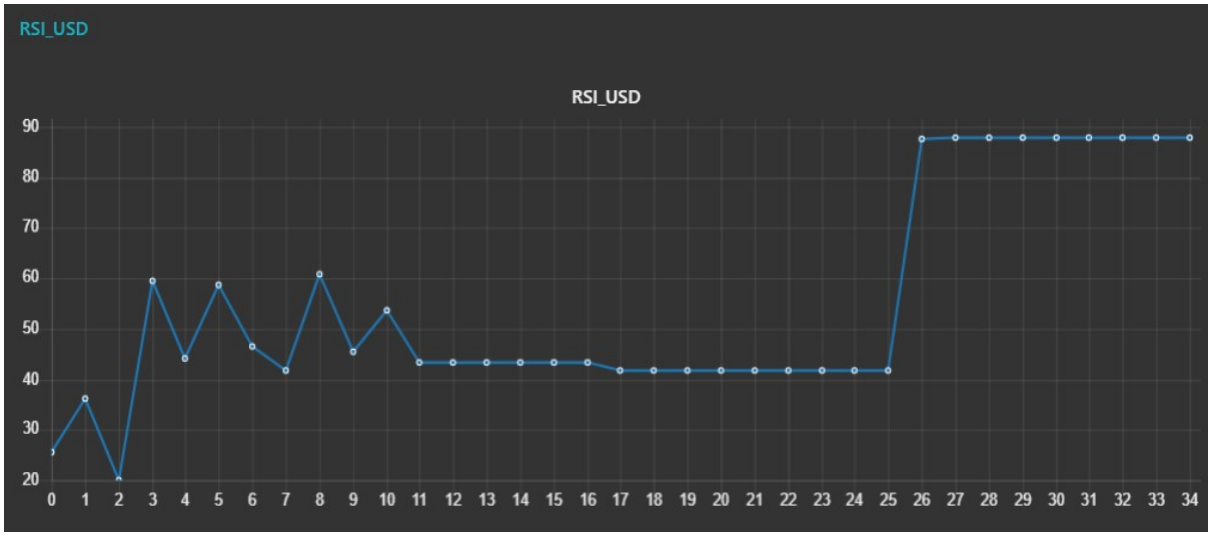


Figure 14: RSI indicator in Node-RED dashboard

The dashboard contains 6 Tabs, one for each currency type and 4 groups per tab. Each group contains the plot of a single technical indicator. The Day Trader can predict the future fluctuations of exchange rates by viewing these charts and making decisions on his future investments. If the exchange rates are going to increase in time to come, he would purchase more currency, and if they are to decrease, he would short sell the currency in his possession.

3 Results and Discussion

This prototype was successfully implemented to include all the functionalities mentioned in the Methodology for a single user, who can subscribe to a single currency type. However, this system could be extended to accommodate multiple users as well as to include the functionality for a single user to be able to subscribe to multiple currency types for notifications.

Since the exchange rates are available to Day-Traders on a daily basis only, and this application succeeds at providing exchange rate information at a higher frequency (hourly) even with a free API subscription plan, this could be seen as a useful service for Day-Traders. This could however be extended to provide data every 30 seconds with a paid subscription, or even to be updated every second with the same API.

There were several challenges encountered during the design and development of this application and one of which would be the implementation of a sleep mode for the NodeMCU. As the exchange rate information is required at high rates (30 seconds or less), it was not possible to implement a sleep mode energy saving algorithm for the NodeMCU and the constrained devices connected to it, at times when the Forex market is active.

4 Conclusion

Day Trading is an important activity to maintain the value of a currency type. The value of currency in the Forex market is determined by the supply and demand of currency and Day-trading supports this

process. This project included the implementation of a prototype of a service to provide Day-Traders with exchange rate information and several Technical indicators over a period of time information which they require to conduct their Day-Trading activities successfully. The ability to view this information through live dashboards and to receive notifications regarding their investments automate the process of having to monitor the exchange rates manually.

5 Annexes

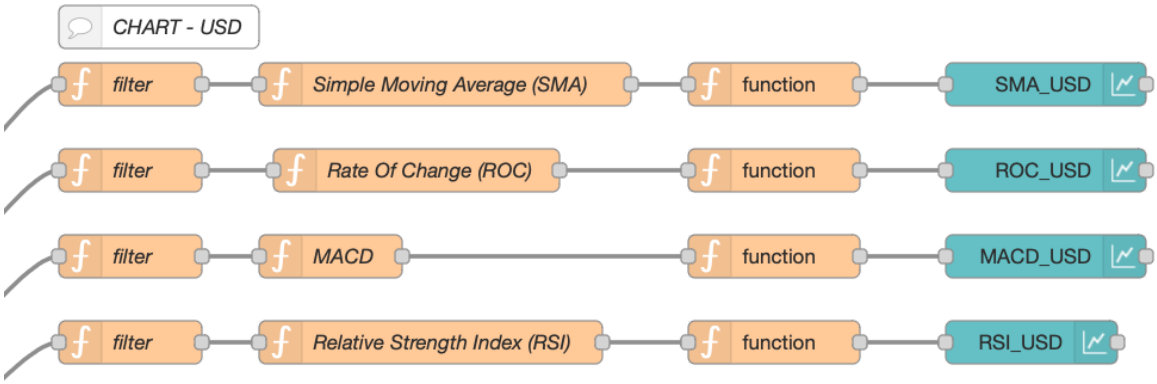


Figure 15: Nodes used to create technical indicator charts

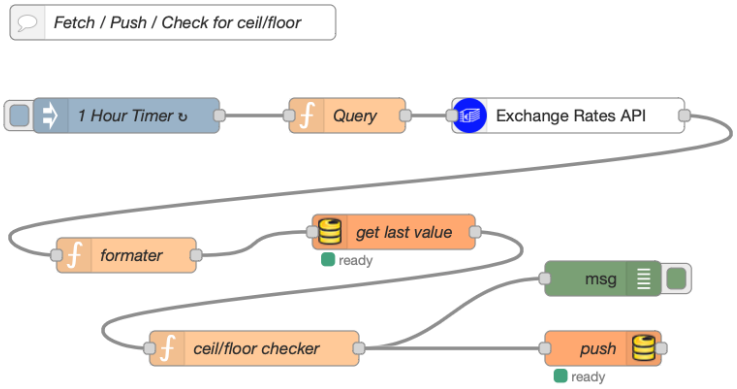


Figure 16: Nodes used to fetch values and then send to firebase

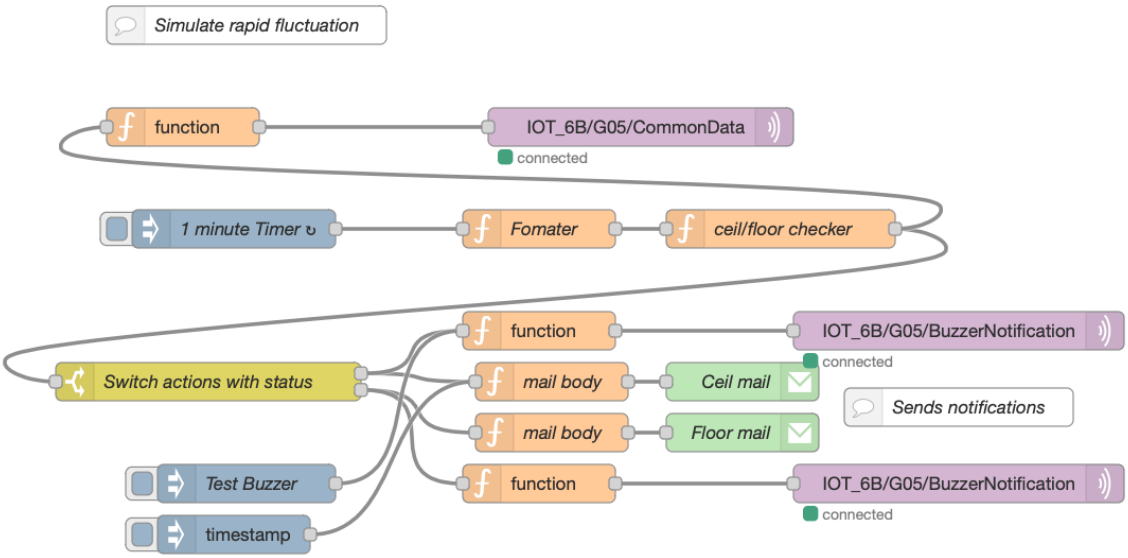


Figure 17: Nodes used to simulate rapid fluctuations

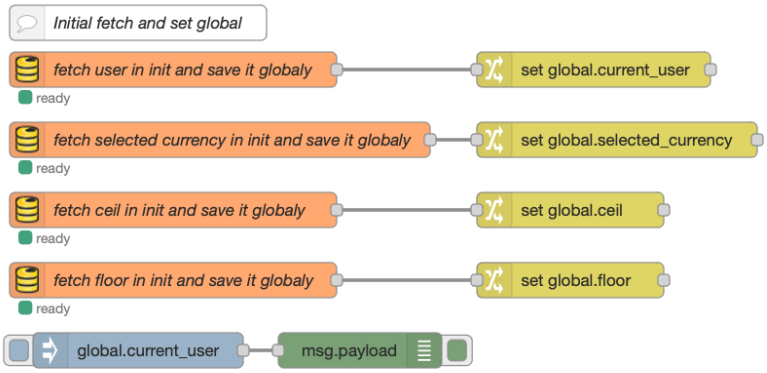


Figure 18: Nodes used to initiate set up

References

[1] "test.mosquitto.org," . *Available at:* https://test.mosquitto.org/ (*accessed* 2021-06-20).

[2] "What Is Token-Based Authentication? | Okta," . *Available at:* https://www.okta.com/identity-101/what-is-token-based-authentication/ (*accessed* 2021-06-20).

[3] "Max deep sleep for ESP8266, a deep-dive analysis • ThingPulse," . *Available at:* https://thingpulse.com/max-deep-sleep-for-esp8266/ (*accessed* 2021-06-20).

[4] "Forex Market Hours," . *Available at:* https://www.markethours.net/forex-market-hours (*accessed* 2021-06-20).

[5] "node-red-contrib-exchangeratesapi (node) - Node-RED," . *Available at:* https://flows.nodered.org/node/node-red-contrib-exchangeratesapi (*accessed* 2021-06-20).

[6] "node-red-contrib-acoustics (node) - Node-RED," . *Available at:* https://flows.nodered.org/node/node-red-contrib-acoustics (*accessed* 2021-06-20).

[7] "Firebase Realtime Database," . *Available at:* https://firebase.google.com/docs/database (*accessed* 2021-06-20).

[8] "node-red-contrib-firebase (node) - Node-RED," . *Available at:* https://flows.nodered.org/node/node-red-contrib-firebase (*accessed* 2021-06-20).