
**Department of Electronic & Telecommunication Engineering**

**University of Moratuwa**

# EN2560 - Internet of Things Design and Competition

## Course Project Report (Group 04)



| Name | Index number |
|---|---|
| Watawana H.S. | 180677E |
| V.Y.N.Lokugama | 180359G |
| K.L.G.J.Chandula | 180092F |



This report is submitted in partial fulfillment of the requirements for the

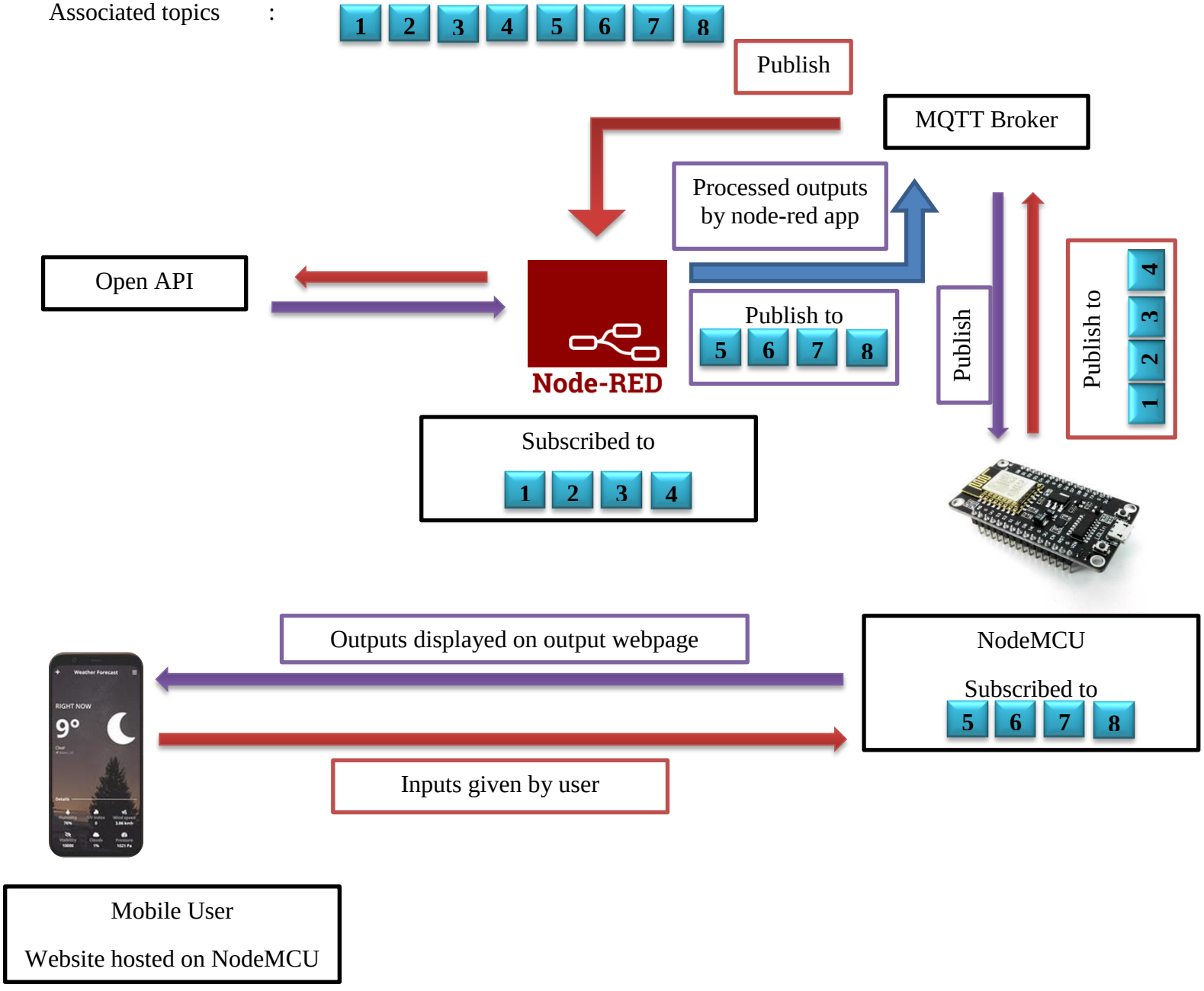
EN2560 - Internet of Things Design and Competition

## Problem Statement

Each year a considerable number of Sri Lankans migrate to China for various reasons such as higher studies, occupations, tourism etc. They encounter struggles in currency conversions during various activities. Being aware of the updated Covid-19 details and statistics, weather updates and air quality levels are other huge concerns they have. A single and a user friendly solution to overcome all these issues doesn't exist.

## System Overview and Operation

We have created an IoT based system which can solve all of the above problems.



## Implementation

### Hardware

A NodeMCU development kit with ESP8266 WiFi SoC is used to implement this system.

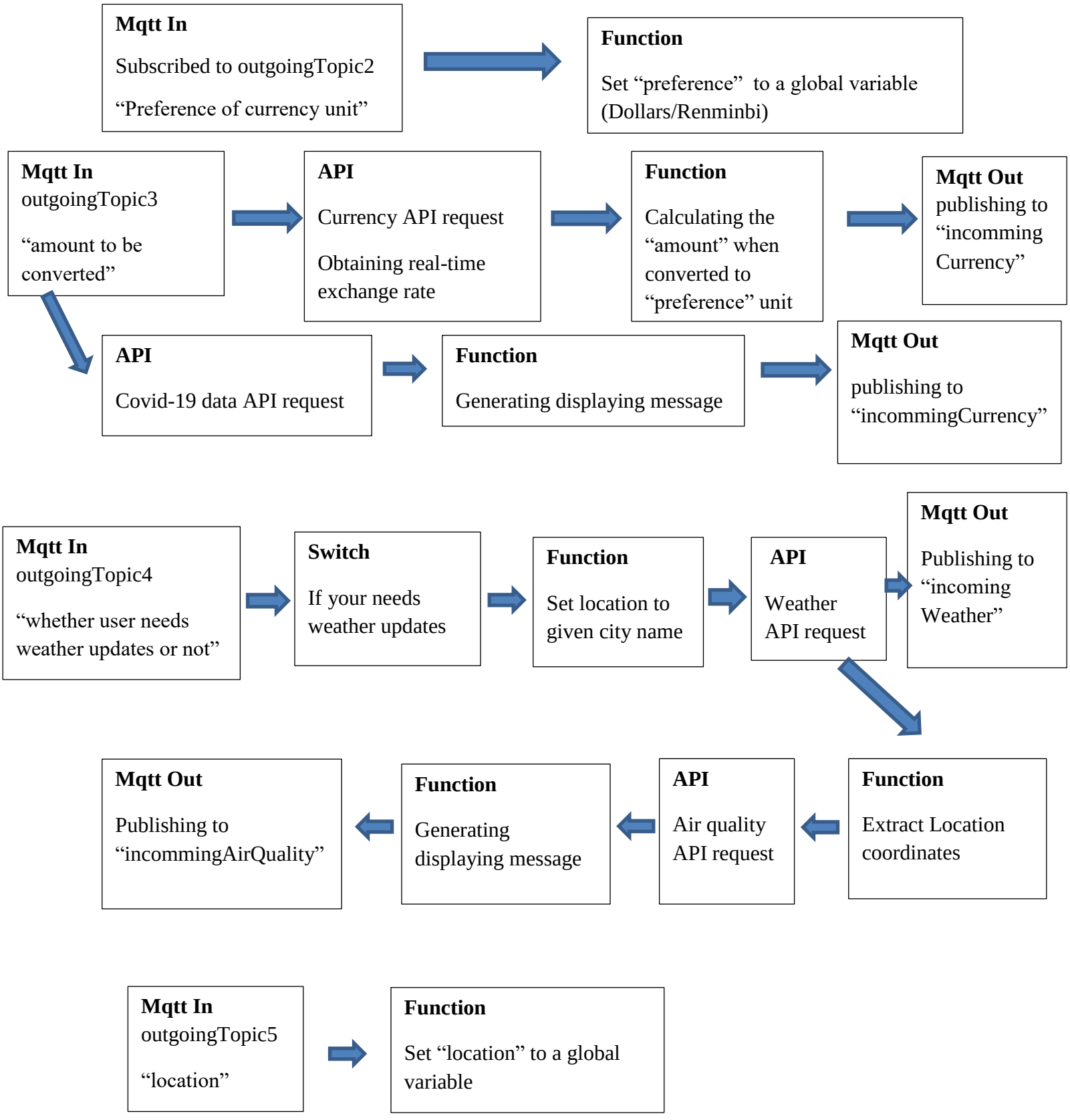
### Software Flow

Software functionality of ESP8266 as a WiFi server and an MQTT client,

- Additionally to Arduino, ESP8266 WiFi libraries and PubSubClient mqtt client library, two other special libraries are used. They are ESPAsyncTCP and ESPAsyncWebServer. ESPAsyncTCP is an asynchronous TCP library which allows multi connection network environment for ESP8266 based systems. ESPAsyncWebServer is a library based on ESPAsyncTCP and it can create an asynchronous server instance running on ESP8266. In this application scenario, the system needs to handle multiple concurrent connections (ex: dealing with user inputs and outputs), therefore it is important to use an asynchronous approach which can easily integrate with the user interface. On ESP8266, an asynchronous server listening to http traffic is created instead of a normal WiFi server.
- Using PubSubClient library, an mqtt client instance is created on ESP82266 which is connected to a mqtt broker ("Broker.mqttdashboard.com").
- ESP8266 is connected to the WiFi LAN in station mode. Mqtt client connection drives on top of this.
- An HTML code segment (with CSS) on ESP8266 web server hosts the input webpage to take user inputs.
- Any user input filled by the user under any field will be collected to a string variable, converted to a character array and published to the mqtt broker under relevant topic.
- Input fields and corresponding topics that they are published to (these same topics are subscribed by node-red application),
    - outgoingTopic2 : Preference of currency unit (to which currency unit the user needs to convert the inserted amount to, it is either Dollars or Renminbi)
    - outgoingTopic3 : Currency amount to convert
    - outgoingTopic4 : Preference on obtaining weather updates or not
    - outgoingTopic5 : Location (if user wishes to obtain weather updates, can enter the city name here)
- A reconnect function is used to reconnect the ESP8266 mqtt client if it gets disconnected from the broker. Every time it reconnects a randomly generated client ID is used to ensure that it connects with a unique client ID for a secured connection. Under this function, ESP8266 mqtt client will subscribe to four topics on the same mqtt broker
- Topics to which ESP8266 mqtt client has subscribed to and corresponding data published under that topic by the node-red application,
    - incommingCurrency : Inserted currency amount converted to the requested currency unit
    - incommingCovid : Covid-19 updates (real time data)
    - incommingWeather : Weather updates of the location provided
    - incommingAirQuality : Air quality of the location provided
- A callback function is used to collect data that comes to ESP8266 mqtt client via the mqtt broker, to the above topics that ESP8266 has subscribed to. These data will then be embedded inside an HTML code

segment which holds the output webpage of the user interface along with the results. Using the asynchronous server mode running on ESP8266, this will be displayed to the user via http.

Software functionality of cloud hosted node-red application ,



- OutgoingTopic3 ( “amount” to be converted) triggers the Covid-19 API request
- Air quality API needs coordinates of the city to provide data. But it is not user friendly to ask coordinates as an input. Therefore, we get city name as an input. Weather data is taken from user input city name. The coordinates are included in weather API response. Therefore we extract location coordinates from weather API response and use it in air quality API request .

**Protocols**

HTTP : Used to host the user interface webpage on the internet (code on ESP8266)

Used to communicate between node-red application and the Open API data bases to obtain necessary data

MQTT : The entire system is implemented in a publish-subscribe architecture via an MQTT broker

Other than these application layer protocols, in lower layers normal internet protocol suite is used (TCP/IP). Here we use an asynchronous TCP library to create an asynchronous web server (on ESP8266) that can handle up to multiple connections at a time.

**Online Resources**

- Currencyscoop API for Currency Data

Currencyscoop provides a simple REST API with real-time and historical exchange rates for 168 world currencies

Request: https://api.currencyscoop.com/v1/latest?api_key=53e6a9b8f099686afaef8c3057eebdc3

Documentation: Currencyscoop Currency APi Code Samples

- IQAIr AirVisual API for Air Quality Data

Request:
http://api.airvisual.com/v2/nearest_city?lat={{LATITUDE}}&lon={{LONGITUDE}}&key=80b7dfaf-7852-4792-98d3-d63dba9b0734

Documentation: AirVisual API (iqair.com)

We generate advice based on US AQI (Air Quality Index)

- Open Weather Map API for Weather Data

Covid19 data: using palette node-red-contrib-covid19 (node) - Node-RED (nodered.org)

**References**

https://github.com/me-no-dev/ESPAsyncTCP

https://github.com/me-no-dev/ESPAsyncWebServer

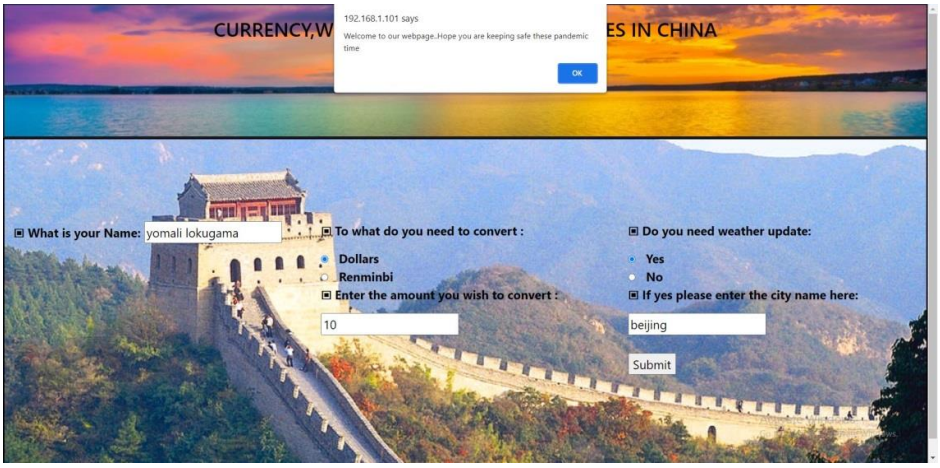
AirVisual API (iqair.com)

node-red-contrib-covid19 (node) - Node-RED (nodered.org)

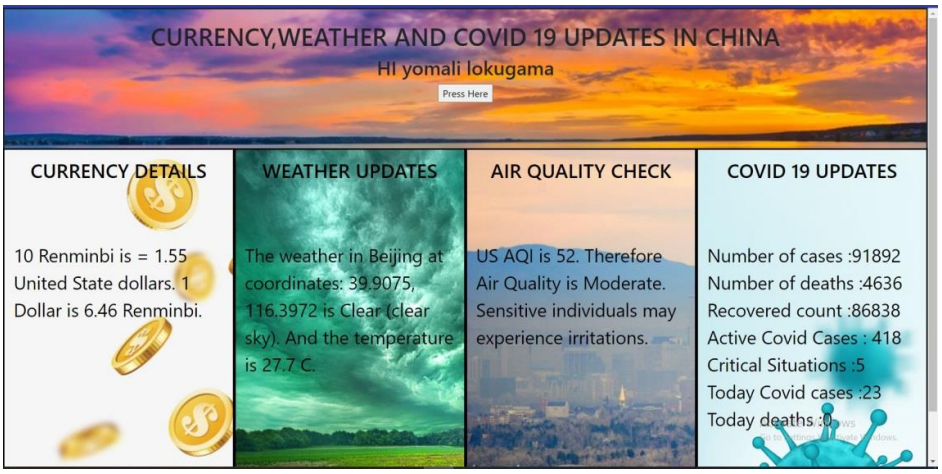
## Achieved System Functionality

The entire system functionality as mentioned above is achieved

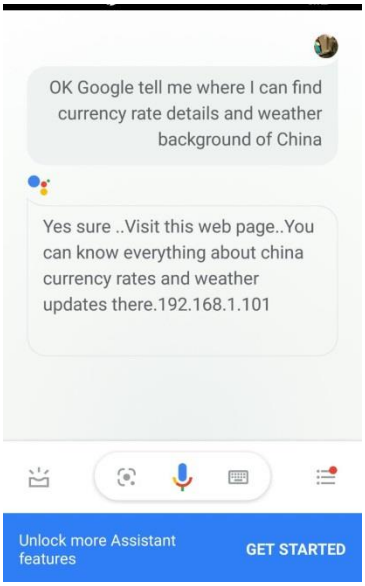
Input window,




Output window,




Integrating the system with Google Assistant,




- Implemented using Google Assistant and IFTTT