POS system using Node MCU, Node-Red, and Firebase

P.D.Mahawela [180378M], Y.C.W.Arachchi [180042E], and K.R.H.M.D.M.Kularatne [180330K]

***Abstract*—The system presented in this report is focused on developing a Real-time end-to-end IoT, Point of Sale (PoS) system which can be installed on local restaurants with less effort. The presented POS system is consisted of three main features; GUIs for customers and to the kitchen and a Real-time database to handle orders and maintain stores' inventory. Key technologies such as Node MCU, Node Red , Firebase has been used to deliver a reliable as well as user friendly experience to the clients.**

I. Introduction

UNDER the new normal concept people are trying to avoid human contacts as much as possible. But still local restaurants and food stores are unable to follow these covid regulations where waiters deliver food and handle the invoices. As a solution to this problem, we are introducing this new POS platform where people can go to the food stores and enjoy their meals under a safe environment.
Also, we had to give our attention to the following parameters beforehand.

- A simple Interface and even non-technical personals can work freely.
- Simple installation guides.
- Attractive UI for the customers to place orders.
- Keeping a real time database to keep record on everything.
- Easily scalable platform as number of tables gets vary depending on the restaurant.
- All in one platform to handle and monitor everything happens inside the shop.

We designed a Realtime end-to-end IoT POS system addressing above parameters as well as covering the minimum technical scope that is expected to reach through this project. In this report we present all the technical as well as practical implementation related to this project. A basic overview will be provided under section II. Following 5 sections will discuss the technical and working principles in a concise manner.

II. Overview

Our proposed POS system resolves the problem mentioned in Introduction, through IoT devices fixed at each table of a restaurant. When a customers arrive at the restaurant they can sit at any vacant table inside the restaurant. An IoT device [Fig. 1] is placed at the centre of the table with the respective table number. Customers need to connect to the WiFi router of our restaurant and scan the QR code given to their table which connects them to the web server we have created.

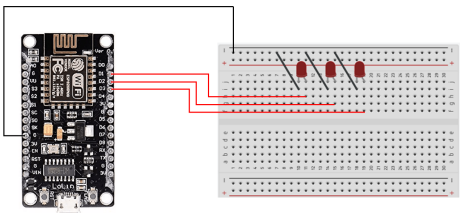

Fig. 1: The IoT device circuitry

After they are connected to our web server, they will be displayed a web page with all the items of the menu. They must be connected to our web server until they check-out. First, customers need to select the table number which they are seated at from the drop down menu. Then, they must add quantity of each item they like to order and click the *ADD* button to add that item to the order.

After the customer have entered all the items they need to the order, they can view their order and the total amount at the end of the web page. There, they will have two options, whether to re-enter the order or to confirm the order. If they click *Re enter* button the web page will refresh and they will have to follow the process from the beginning. If they click *Confirm order* button their order will be sent

to the Kitchen. Also, they can view the final order description, which they need to show at the cashier.

When the order is sent to the Kitchen, the chef will be notified the arrival of the order with the table number. Then, the chef can view that order and send a estimated time to complete the order from the options given. An audio with the estimated time will be played when the chef notifies the customer while a popup message will also show up.

After the order is completed, chef will click the complete button. Then, the customer will be notified with a audio message as well as a pop up message, to come to the counter and collect their order. After enjoying their meal, the customer must go to the cashier to pay their bill. Fig. 2 shows the block diagram of our POS system.

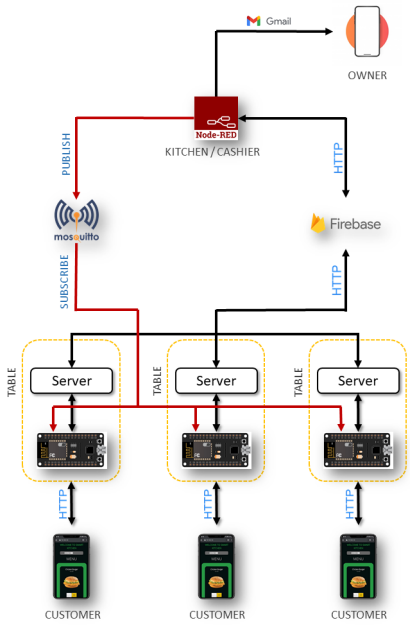

Fig. 2: POS block diagram

III. Firebase

Firebase is a database service provided by Google. It is free to use, and users do not need SQL knowledge to use Firebase. Firebase consists of many functionalities. In this project we have used Firestore database, Realtime database, and the Storage functionalities. We use Javascript in our webpage to read and write data from the customers smartphone and *node-red-contrib-firebase-admin* palette to read and write data from Node-Red to Firebase database.

Firestore database is used to save all the items of our restaurant menu including their attributes such as the item name, item unit price, ingredients and their quantities, and an image of the item. Firestore is used get relevant data to display in the webpage. By using Firestore, we have significantly reduced the size of the web page code, resulting in saving memory in Node MCU. We do not use Firestore to set any data from our web page.

Firebase Realtime database is used to write data we send out from our web page and to read data for the Node-Red dashboard. The main advantage when using Realtime database is that we can get results whenever the data in the relevant reference is changed or updated. It has increased the dynamic nature of our application to a great extent.

Firebase Storage is used to save all the multi-media files we have used in our POS system. It has helped us to provide a better user experience for the customers. Fig. 3 demonstrates the use of Firebase in our project.

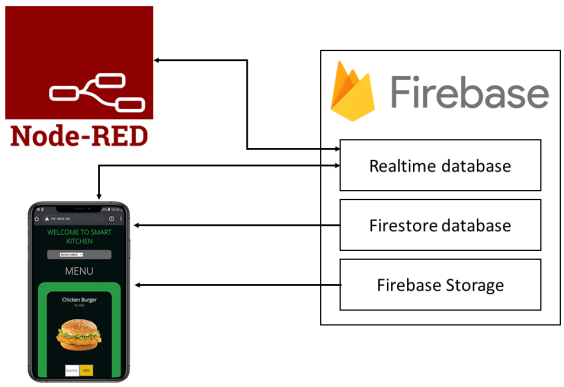

Fig. 3: The menu item details are read from the *Firestore* database to the web page. Data is read and written to the *Realtime* database from both the web page and Node-Red dashboard. *Firebase storage* is used to save all the multimedia files in a cloud storage.

IV. ESP8266 Web Server

We have used a Node MCU ESP8266 module to connect the customer's device to our POS system through http protocols. We first configured our Node MCU as a WiFi AP. In addition to that, we have also configured the Node MCU as a ESP8266 web server which hosts our web application [Fig. 4]. In this project we heavily relied on JavaScript to read and write data from Firebase as well as make our web page code simple and dynamic. Therefore, we did not display our web page using client.println method. Since we are using MQTT also, using client.println will make confusions in the code with the client.subscribe method which is used in MQTT.

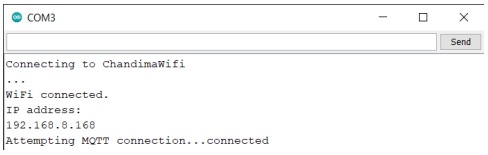

Fig. 4: Creating the web server.

Therefore, we created a group15NodeMCUcode.h file and included into our Arduino code. In this group15NodeMCUcode.h file, it consists of the HTML, CSS, and JavaScript code pieces of our web page. By following this method we were able to make a dynamic web page.

A. HTML

The HTML code snippet defines all the basic components in the web page. The components of our web page are,

1) The heading.
2) Selecting the table.
3) The menu.
   - Names of the items.
   - Prices of the items.
   - Images of the items.
   - Text fields to enter the quantity of each item.
   - Buttons to add the items to the order.
4) The order and the total amount.
5) Buttons to re-enter the order or confirm the order.
6) Audio components to be played when a notification arrives.

B. CSS

We use CSS to add aesthetics and styling to our web page. We have used classes to style categories of components and ids to style special components.

C. JavaScript

JavaScript code snippet is the most important pieces of code of our web page. We used JavaScript to increase the efficiency of the code, reduce the size of the code, add dynamic features to our web page, and read and write data from/to our Firebase database. We have defined and used several JavaScript functions to gain the above advantages. We defined JavaScript functions to,

1) Configure the Firebase databse.
2) Select the table and display notifications if any arrives.
   - Window popups with the relevant message.
   - An audio file is played according to the notification.
3) Add items to the order.
4) View the items of the menu.
   - Reads data from the Firestore database.
   - Appends items to the Menu list.
5) Confirm the order.
   - Writes data to the Realtime database.
   - Popups a confirmation to make sure.
6) Re-enter the order.

V. Node Red

Node red and it's dashboard provides following features: data prepossessing, GUI Interfaces for kitchen, cashier, and Inventory , sends a record of daily sales to the owner at the end of the day [Fig. 5]. It is important to know that this proposing dashboard is compatible only for three tables and it can be adjusted as per user requirement.

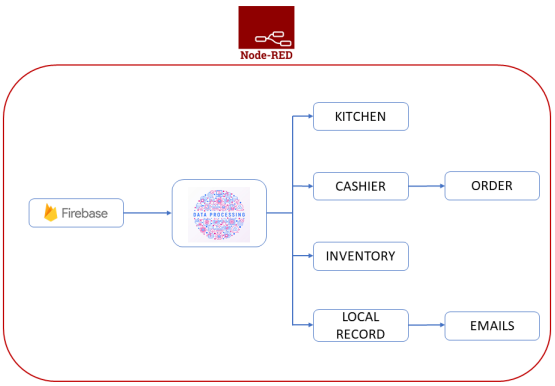

Fig. 5: node red block diagram

A. Kitchen

This tab consisted with three groups, each for the respective table. When an order is placed it appears on the respective tab with a pop-up notification. Chef can view the order and should select an estimation time for order to be prepared. Depending on the order it could be 5, 10, 15 or 20 minutes and will be informed to the customer. When the order is complete chef should press the complete button to

inform the customers about it and by pressing the clear will clear out the respective order from this tab.[Fig. 6]

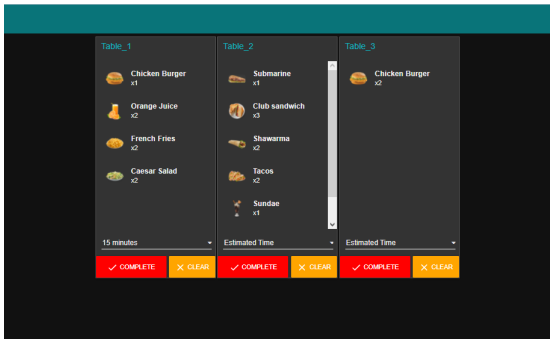

Fig. 6: node red block diagram

B. Cashier

This tab has two groups.

- Order list
- Daily sales

Ongoing orders may appear in the order list.[Fig. 7] The proposing platform is capable of maintaining a local record of daily sales inside the hosting machine and it can be sent to the owners' mobile phone as an e-mail or a SMS by pressing daily sales button at the end of the day. We use gmail and twillio open APIs to achieve this. Once you select an order you will be directed to the Billing tab. It will display order details such as item name, unit price, quantity, item price and gross amount.[Fig. 8] Balance will automatically appear when the amount paid by the customer is entered. After is payment is finished dashboard will be redirected to the CASHIER tab by pressing the paid button.

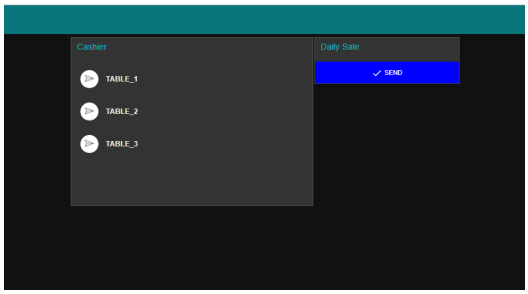

Fig. 7: Cashier Dashboard

C. Inventory

This tab gives a bar graph visual representation[Fig. 9] on current inventory(available ingredient levels) which is maintained on firebase and get

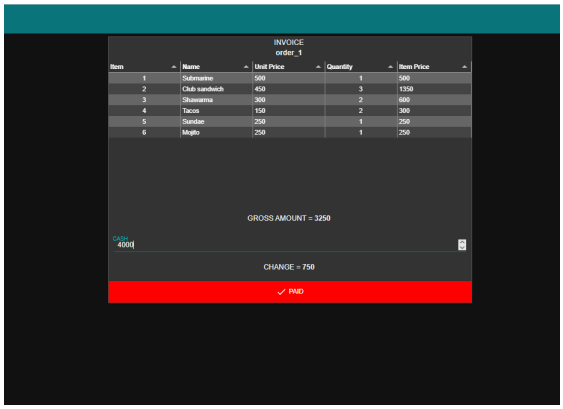

Fig. 8: Bill Dashboard

updated every time when a new order is placed. Also, there is a short list including ingredients which are running short.

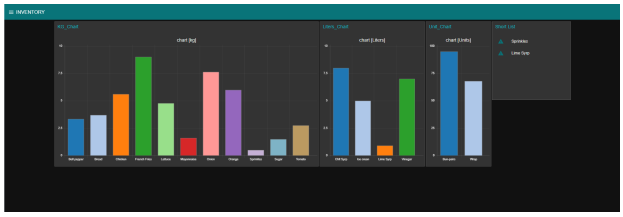

Fig. 9: Inventory Dashboard

VI. Firebase - Node Red connection

Here in our implementation , firebase Realtime database is in the middle of the flow of the system. So, we have installed a palette to get data from firebase which is **node-red-contrib-firebase-admin**. Credentials of the firebase must be added to the palettes therefore it can access the Firebase Realtime database

Node	Feature
rtdb - set	Set data at a path in the Realtime database. Use "on" snapshot so will fire every time the data at the path changes and so drive flow execution from that point
rtdb - get	Get data from a path in the Realtime database
rtdb - on	Get data from a path when the Realtime database path triggers
rtdb - push	Pushes the new object onto an array under the path

Fig. 10: Nodes used to Firebase-NodeRed Connection

To connect the Firebase console or the database we must add its credentials to the node palettes.

Once we connect through it, we can easily add, delete, modify, read in the database through the node-red. Here the overview of the flow ,

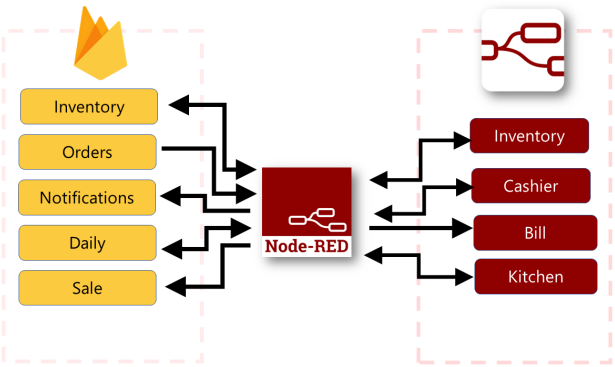

Fig. 11: flow of Firebase-NodeRed Connection

Node MCU updates the order section in the firebase database with the pre-defined structure in it. Once the Orders section is update through the node MCU, Node-red reads the relevant data through the rtbd-on which only trigger when the relevant path changes. Cashier, Kitchen, Bills tabs are only depending on the order section of the database.

Then the data send to dashboard to data visualize and another to the inventory by updating the inventory by redacting the ingredients from the inventory. Here we use another rtbd-on node to visualize the current inventory and the inventory is always set as a flow object in node-red. As in the Kitchen dashboard the customer can be notified the estimated time and the order completed notification. Basically, the node-red writes in the notifications section in the Realtime database with use of rtdb-set node.

From the Cashier tab we can send the daily sales to the owner or anywhere via email and as a message(twilio). With every order, the order details are updated in the daily section in the database. By clicking the send button in the daily sale table, node-red access the database, send the relevant data and clear the daily sale list.

VII. MQTT

We used MQTT to send messages to our Node MCU when an order was completed. We used the mosquitto MQTT broker. The Node MCU is subscribed to "POS/EN2560/order" topic. When an order is completed, we publish a value of '1' to the above-mentioned topic. When the Node MCU receives a message of '1' from the above topic [Fig. 12], it will light up LEDs to inform the customer. The callback function is executed when a message is arrived. Inside the call back function we execute the LED function if the message we received is equal to 1.

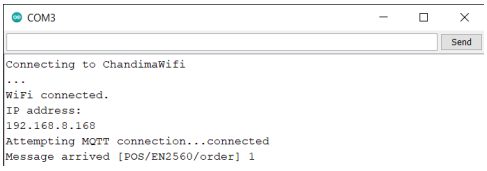

Fig. 12: Recieving the MQTT message

VIII. Conclusion

In our implementation, we were trying to set the all the data in a Cloud to quick access ,security of data and well as to prevent crashing the system with power-outs. User-friendly in Customer, Kitchen, cashier and well as the owner side. Google Firebase realtime database with along the node-red palettes processes data in less than 1 second which makes the system more relable and accurate. For the further development of Our implementation we are thinking of adding a customer profile to each customer via to improve the service and Quality.

References

1) https://flows.nodered.org
2) List of videos in a youtube playlist which we used for our implementation https://youtube.com/playlist?list=PLrT4C4Ykd-kKXpCvbwVkC9vqdVh-0aYgl
3) https://firebase.google.com/docs/database/web/start
4) https://www.twilio.com/console