Implementation of Wireless Quick Response Code Using MCU ESP8266

Dr.V.Vijaya Kumari,Professor

Department of Electronics and Communication Engineering,

The Oxford College of Engineering,Bangalore-560 068

Mail.id:ebinviji@rediffmail.com

**Abstract**

QR scanners are seen in almost all places and the aim of this paper is to revolutionize information accessibility by providing a seamless Wi-Fi- based solution by eliminating the need for QR scanners. To achieve this we use server-client communication method, where the NODE MCU (ESP8266) functions as the server, while a mobile application serves as the client. By leveraging this technology, users will be able to effortlessly retrieve data within their proximity without the need for QR codes, enhancing convenience and improving overall accessibility.

Keywords: Scanner, server, client, communication, node MCU

1.Introduction

In today's fast-paced world, people are constantly on the go, and time is of the essence. One common issue that people face in busy places such as airports, train stations, and shopping malls is the difficulty of scanning QR codes quickly and efficiently. Traditional QR codes require users to scan a physical code within a small distance to read the information, which can be inconvenient and time-consuming in crowded or busy environments.

To solve this problem, wireless QR technology can be implemented, which allows users to read information stored on a local server without the need for physical scanning. With wireless QR, the user's device (such as a smartphone or tablet) can read the information remotely, eliminating the need for physical scanning. This technology uses server-client communication through Wi-Fi to send information from the server to the client when a request is made within a specific range. This range can be customized to suit the needs of the application.

Client-Server Based Applications examines exploring their architecture, functionalities, and potential research opportunities. The author discusses various aspects such as communication protocols, data storage, security, and scalability, highlighting the importance of client-server models in modern computing systems. The paper offers insights into current trends and identifies areas for further investigation in this field(Santosh Kumar,2020). This study provides a comprehensive analysis of client-server architecture, focusing on the utilization of the HTTP protocol for efficient data transmission. The paper explores the functionalities and interactions between clients and servers in this model, highlighting the role of HTTP in facilitating communication and data exchange(Natesan, Abirami N, 2019. A novel approach for controlling a prosthetic hand using an ESP8266 Wi-Fi module and an Android application describes the design and implementation of the interface, which enables users to manipulate the prosthetic hand wirelessly through their mobile devices. The study showcases the potential of integrating Wi-Fi technology and mobile applications in enhancing the functionality and accessibility of prosthetic devices, opening up new possibilities for improving the quality of life for individuals with limb impairments (Pakalapati, S.et.al,2017) The system for controlling electric switches using an Android application via Wi-Fi. The authors describe the design and implementation of the system, which allows users to remotely switch on or off electrical devices through their mobile devices (JA Dandge, R. Shirwadkar et.al., 2016). By providing a user-friendly interface and wireless connectivity, the proposed system offers an innovative solution for remote switch control, contributing to improved energy management and automation in residential settings (SP Makhanya et.al,2019). By leveraging Wi-Fi technology, the proposed system offers convenient and widespread access to information, contributing to improved connectivity and accessibility in various environments (Singh, S & Vijayvargiya. A, 2019).

**2.METHODS**

In order to enable communication between an ESP8266 NodeMCU board and a mobile application, it is necessary to establish a Wi-Fi connection that facilitates the exchange of data without requiring a connection to the internet. This can be accomplished by configuring sever client communication between the devices where ESP8266 acts as an access point or server and the mobile application as a station or client.

Once the Wi-Fi connection has been established and the appropriate configurations have been made, the client can begin making HTTP requests to the server in order to retrieve the desired information. This information may include items such as UPI ID, restaurant menus, or pass verification keys, depending on the specific application.

Multiple clients can connect to the ESP8266 Access Point simultaneously, making it easy for large groups of people to access the same information. Additionally, clients no longer need to physically approach a QR code to scan it, simplifying the process and making it more convenient for users. Overall, the use of wireless communication technology can greatly enhance the speed, efficiency, and accessibility of information for the user who had requested.

Figure 1:Process flow of the wireless QR

To begin with, the user needs to connect their mobile device to the hotspot created by the ESP8266 NodeMCU board via Wi-Fi. Once the connection is established, the Mobile Application will retrieve the IP address of the ESP8266 board. Using the IP address, the Mobile Application will then establish a connection with the ESP8266 board using the HTTP protocol. Upon successful connection, the Mobile Application can send a request to the ESP8266 board to retrieve the required information, such as the UPI ID, Pass Verification Key or Restaurant menu. The ESP8266 board will then promptly responds to the request and provide the quick information, which will be displayed on the Mobile Application. Multiple clients can connect to the ESP8266 Access Point and obtain the quick information seamlessly, eliminating the need for physical QR code scanning and making the process of obtaining information effortlessely.

2.1 NodeMCU ESP8266

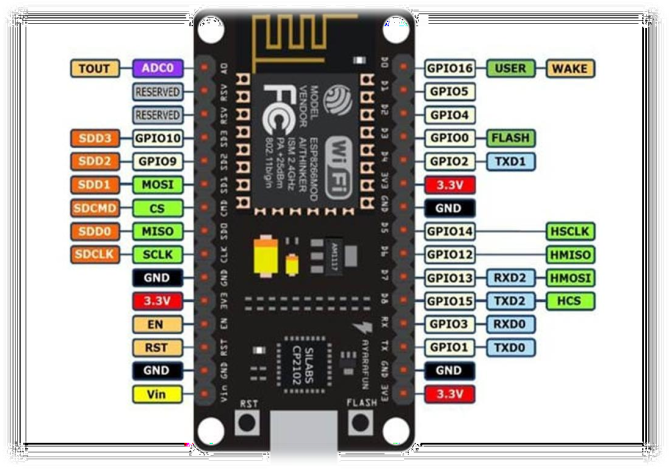
Figure 2: NodeMCU ESP8266 pin diagram

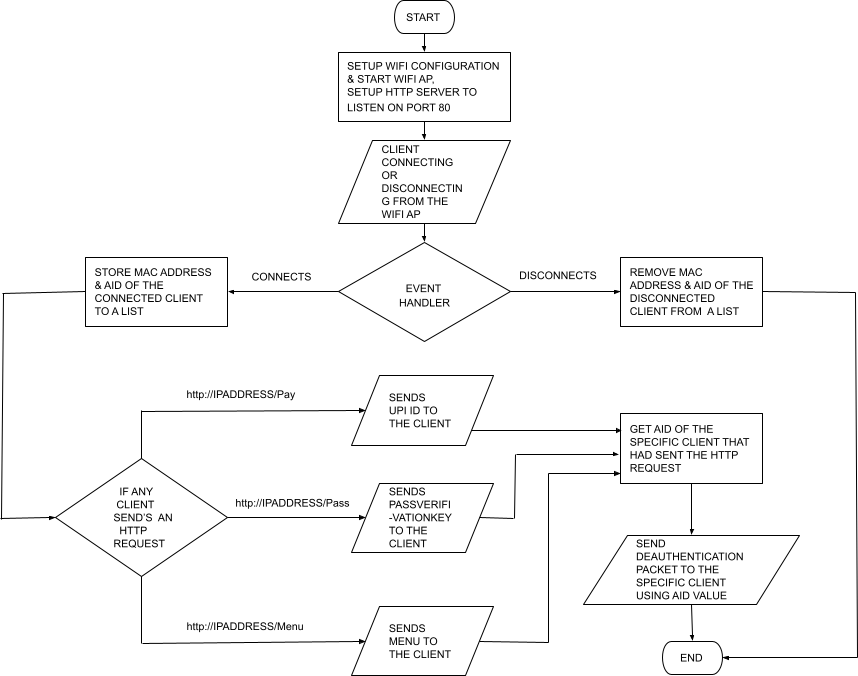
ESP8266\_RTOS\_SDK is a software development kit for the ESP8266 chip, which is a low-cost Wi-Fi microchip designed for Internet of Things (IoT) applications. This software development kit is based on FreeRTOS, a popular open-source real-time operating system, and provides a robust platform for developing Wi-Fi-based IoT applications.ESP8266\_RTOS\_SDK provides a set of libraries and tools for building applications for the ESP8266 chip, including support for Wi-Fi networking, TCP/IP communication, and GPIO interfaces. The SDK also provides a range of sample applications and documentation to help developers get started quickly.

#### 2. 2 MIT App Inventor

MIT App Inventor is a web application integrated development environment. It allows a newcomer to computer programming to create application software (apps).This application is open Source software which can be access through web browsers. Applications are created by dragging and dropping components into a design view and using a visual blocks language to obtaining the information effortlessly. program application behaviour that can run on Android devices.

* 1. **3.FLOW CHART**





* The program starts at the top of the flowchart.
* The program initializes the WiFi configuration and starts the WiFi access point.
* The program sets up an HTTP server to listen on port 80.
* The program waits for input from clients connecting or disconnecting from the access point.
* If a client disconnects, the program removes the MAC address and AID of the disconnected client from a list and ends.
* If a client connects, the program adds the MAC address and AID of the connected client to a list.
* The program checks if the client has sent an HTTP request.
* If the client has sent an HTTP request, the program sends a corresponding response to the client.
* The program gets the AID of the client that sent the HTTP request from the list.
* The program than sends a de-authentication packet using the AID.
* The program ends.

## 2.3 APP DESCRIPTION

Figure 3: WiFi Connection User Interface

The WiFi Connection Screen is the first screen that appears when the app is launched. It contains a button that initiates the WiFi connection process. When the button is clicked, the app checks whether the location is enabled or not. Location is required because the app uses the Tiffin extension to fetch the available WiFi SSIDs. If location is not enabled, the app will prompt the user to go to the device's settings and enable it. This is done using the "Activity Starter" block that opens the device's location settings. Once the user enables location, they can return to the app by pressing the back button.

Next, the app uses the Tiffin extension to get the available WiFi SSIDs and stores them in a list picker. The Tiffin extension is a third-party extension that provides functionality to fetch and connect to WiFi networks. In the block code, the "AvailableSSIDs" block is used to fetch the available SSIDs. The "for each" loop is used to iterate through the list of available SSIDs and add them to the list picker.

When a user selects a particular WiFi network from the list picker, the app displays the SSID and prompts the user to enter the password. However,to connect to the ESP8266 board, the password is not required. Therefore, the app proceeds to connect to the WiFi network using the Tiffin extension. The "ConnectSSID" block is used to connect to the selected WiFi network.



Once the WiFi network is connected, the app redirects to the next screen. In the block code, the "SignalStrength" block is used to check whether the WiFi network is connected or not. If the network is connected, the app navigates to the next screen using the "Screen1.Open another screen" block.

password is not required. Therefore, the app proceeds to connect to the WiFi network using the Tiffin extension. The "ConnectSSID" block is used to connect to the selected WiFi network.

Once the WiFi network is connected, the app redirects to the next screen. In the block code, the "SignalStrength" block is used to check whether the WiFi network is connected or not. If the network is connected, the app navigates to the next screen using the "Screen1.Open another screen" block.

Overall, this screen's block code uses various components such as the Location Sensor, List Picker, Button, and Tiffin extension to implement the WiFi connection process. The logic behind the block code is that the app first checks the location status and then fetches the available SSIDs using the Tiffin extension. When the user selects a specific network from the list picker, the app uses the Tiffin extension to connect to the WiFi network. Finally, the app navigates to the next screen once the WiFi connection is established.

3.1.Service Selection Screen

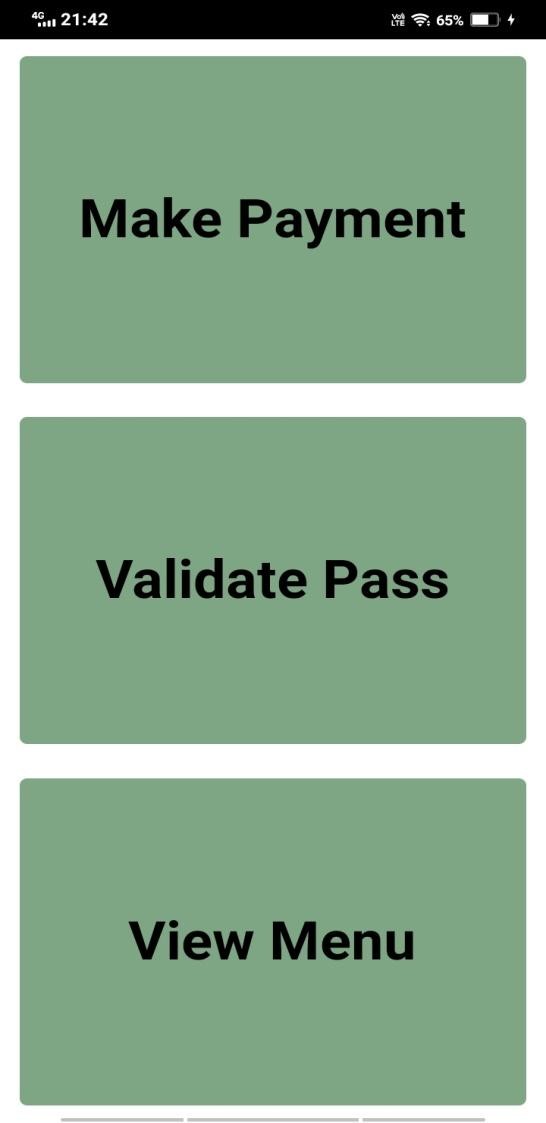
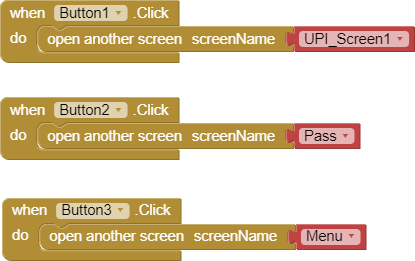


Figure 4: Service Selection User Interface

The Service Selection Screen is the second screen that appears in the app after the user successfully connects to the WiFi network. This screen demonstrates three different services that the app can provide, namely making a payment using UPI, validating a BMTC bus pass, and viewing a restaurant menu.

The screen displays three buttons, each representing a service. When a user clicks on any of these buttons, the app redirects the user to a different screen depending on the selected service. The button for making a payment redirects the user to the UPI\_Screen1 screen. Similarly, the button for validating a BMTC bus pass redirects the user to the Pass screen, while the button for viewing a restaurant menu redirects the user to the Menu screen.

In the block code for this screen, the "when Screen2.Initialize" block is used to initialize the screen after a successful WiFi connection. The three buttons are implemented using the "Button.Click" block. When a user clicks on any button, the app uses the "Screen1.Open another screen" block to redirect the user to the corresponding screen.

Overall, this screen demonstrates the app's ability to provide multiple services using a single app interface. The logic behind the block code is that the app initializes the screen after a successful WiFi connection and displays three buttons that represent different services. When a user clicks on any button, the app redirects the user to the corresponding screen to perform the selected service.

* 1. 3.2UPI Payment Screen

The UPI Payment Screen is the third screen in the app, and it is accessed by clicking the "Make Payment" button on the Service Selection Screen. This screen allows the user to make a payment using the Unified Payment Interface (UPI) service.

Figure 5: Service Selection Block code

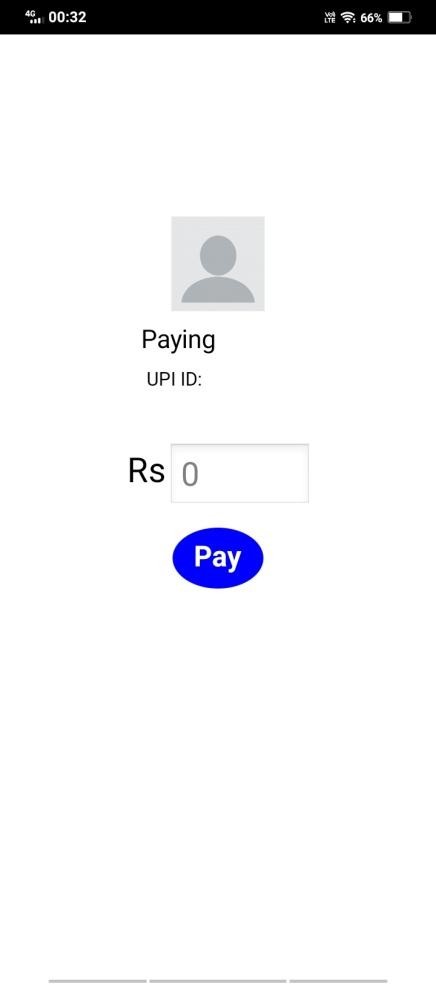


Figure 5: UPI Payment User Interface

When the screen initializes, it gets the

IP address of the connected WiFi network using the Tiffin extension and stores it in a variable. The app then makes an HTTP request to the IP address using the Web component. The request is in the form of a URL, [http://IPADDRESS/Pay,](http://IPADDRESS/Pay) and it expects a response in the form of a UPI ID. The UPI ID is displayed on the screen, and the user can enter the amount to be paid. The minimum amount that can be paid is 1 Rs.

Once the user enters the amount to be paid, they can click the "Pay" button. If the UPI ID is received and the amount entered is greater than or equal to 1 Rs, the app redirects the user to the UPI\_Screen2 to show the success message of the payment transaction.

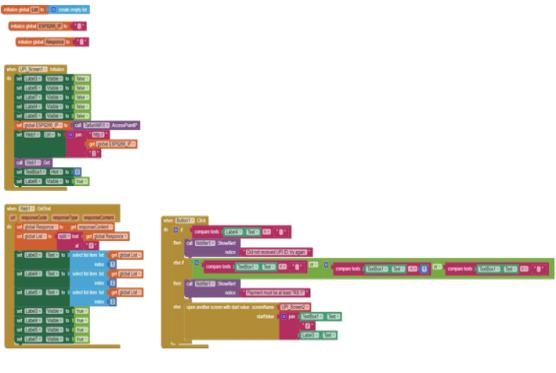


Figure 6: UPI Payment Block code

In the block code for this screen, the "when Screen3.Initialize" block is used to initialize the screen and retrieve the IP address. The "Web.GotText" block is used to receive the UPI ID from the HTTP response. The "Button.Click" block is used to handle the Pay button click event and redirect the user to the UPI\_Screen2 screen.

Overall, this screen demonstrates the app's ability to make UPI payments using WiFi connectivity. The app retrieves the IP address of the connected WiFi network and makes an HTTP request to obtain the UPI ID. The user can enter the amount to be paid and complete the transaction using the Pay button.

* 1. 3.3 UPI Payment Success Screen

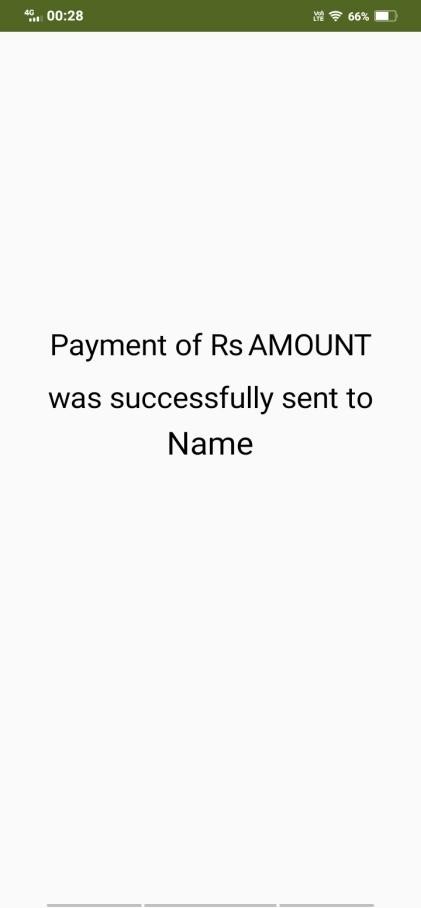


Figure7: UPI Payment Success User Interface

The UPI Payment Success Screen, also known as UPI\_Screen2, is the fourth screen in the app. It is accessed after the user successfully completes a payment transaction by clicking the "Pay" button on the UPI Payment Screen (Screen 3). This screen displays a success message to confirm that the payment was successful. When the screen initializes, it shows a message on the screen indicating the success of payment

transaction. The message is displayed as "Payment of Rs 'Amount' was sent to

'Name'". Here, 'Amount' represents the amount of money that was sent, and 'Name' represents the name of the person or entity to whom the payment was made.

In the block code for this screen, the "when Screen4.Initialize" block is used to initialize the screen and display the success message. The message is typically set by retrieving the 'Amount' and 'Name' values from the previous screen using global variables or stored data. Overall, this screen provides confirmation to the user that the payment transaction was successful. It displays the amount sent and the recipient's name, allowing the user to verify the details of the transaction. This serves as a reassuring message that the payment process has been completed successfully.

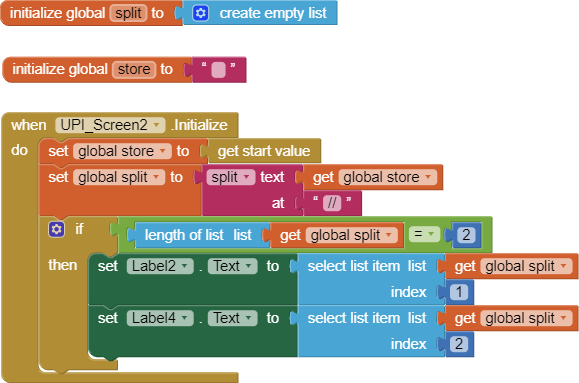


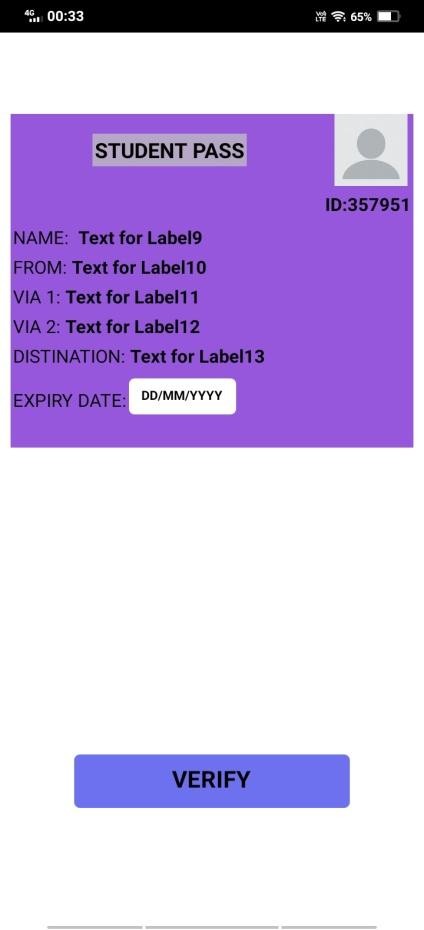
Figure 8: UPI Payment Success Block code

3.4 Pass Validation Screen

The Pass Validation Screen, also known as the Pass screen, is the fifth screen in the app. It is accessed when the user clicks the "Validate Pass" button on the Service Selection Screen (Screen 2). This screen allows the user to validate a BMTC bus pass by verifying its expiration date.

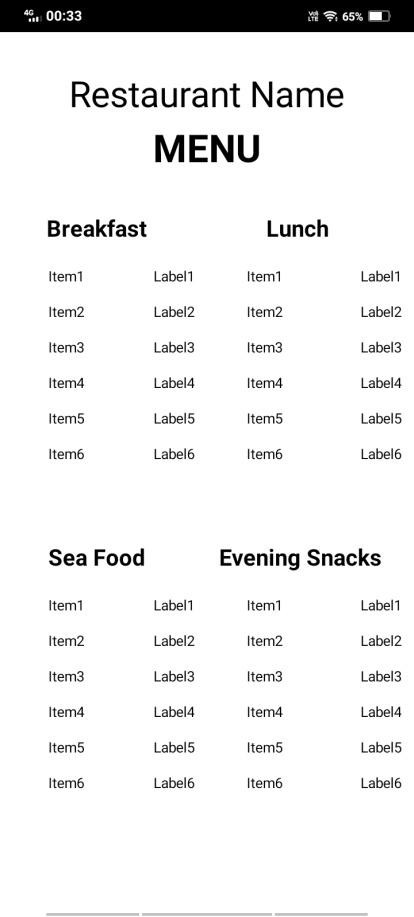
When the screen initializes, the app first sets a demonstration expiry date to showcase the functionality. In practice, the actual expiry date is set when the pass is created and stored in the pass's data.

Figure 9: Pass Validation User Interface



On this screen, there is a "Verify" button. When the user clicks the "Verify" button, the app retrieves the IP address of the connected WiFi network using the Tiffin extension and stores it in a variable. The app then makes an HTTP request to the IP address using the Web component. The request is in the form of a URL, [http://IPADDRESS/Pass,](http://IPADDRESS/Pass) and it expects a response in the form of a Pass verification key. The Pass verification key is stored in a variable.

Next, the app compares the received Pass verification key with the verification key generated when the pass was created. If the two keys match, the app proceeds to check the expiry date of the pass with the current date. If the pass has not expired, the app displays a message on the same screen indicating that the pass has been verified. However, if the pass has expired, the app displays a message indicating that the pass has expired. If the Pass verification key does not match the generated key, the app displays a message stating "Verification ID didn't match".



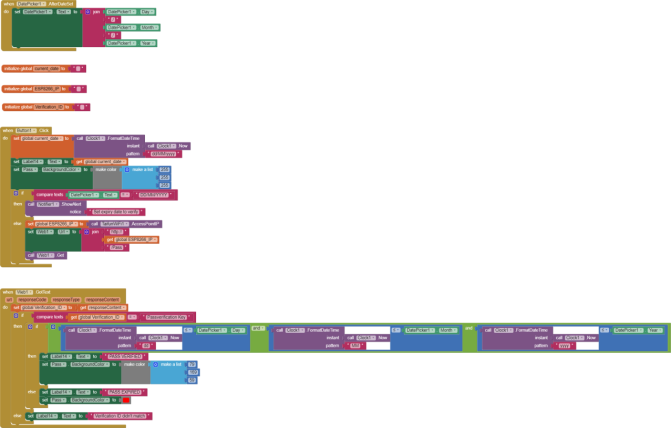


Figure 10: Pass Validation Block code

In the block code for this screen, the "when Screen5.Initialize" block is used to initialize the screen and set the demonstration expiry date. The "Web.GotText" block is used to receive the Pass verification key from the HTTP response. The verification process, including the comparison of the verification keys and checking the expiry date, is implemented using conditional statements.

Overall, this screen provides a means to validate a BMTC bus pass by comparing the Pass verification key and checking the pass's expiry date. The user can click the "Verify" button to initiate the validation process, and the app provides appropriate feedback based on the verification result and pass expiration status.

3.5Restaurant Menu Screen

Figure 11: Restaurant Menu User Interface

The Restaurant Menu Screen, also known as Screen 6, is accessed when the user clicks the "View Menu" button on the Service Selection Screen (Screen 2). This screen allows the user to view the menu of a restaurant.

When the screen initializes, the app retrieves the IP address of the connected WiFi network using the Tiffin extension and stores it in a variable. The app then makes an HTTP request to the IP address using the Web component. The request is in the form of a URL, [http://IPADDRESS/Menu,](http://IPADDRESS/Menu) and it expects a response in the form of the menu items.

The response received from the HTTP request is displayed on the screen as the restaurant menu. The menu is divided into sections, each having a heading. Under each heading, there are rows that display the item name and its corresponding price. The number of rows displayed depends on the items received in the response. This ensures that only the available items are displayed in the menu.

In addition to displaying the menu, the app also receives the restaurant name before displaying the menu. This allows the user to identify the restaurant associated with the menu they are viewing.

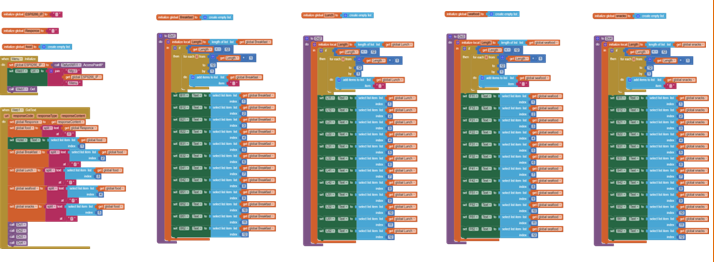


Figure 12: Restaurant Menu Block code

In the block code for this screen, the "when Screen6.Initialize" block is used to initialize the screen and retrieve the IP address. The "Web.GotText" block is used to receive the menu response from the HTTP request. The menu is dynamically generated by iterating over the received items and displaying them in the appropriate sections and rows on the screen.

Overall, this screen demonstrates the app's ability to retrieve and display a restaurant's menu using WiFi connectivity. The app retrieves the IP address of the connected WiFi network and makes an HTTP request to obtain the menu items. The menu is then displayed on the screen, organized into sections and rows, with the available items shown to the user.

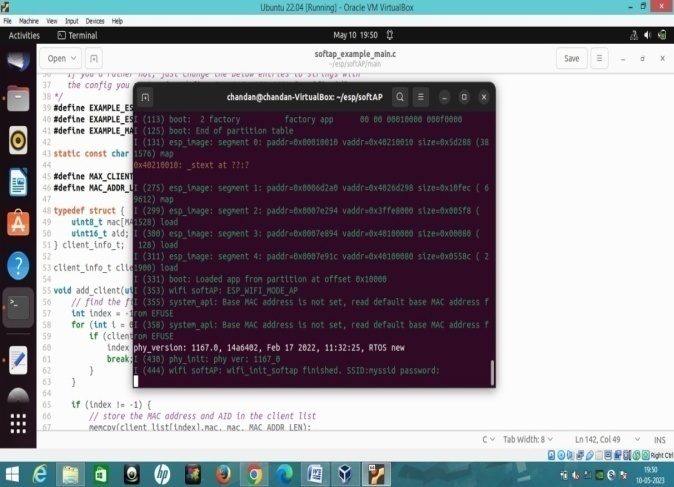


Figure 13: Restaurant Menu Block code



Figure 14: Wi-Fi-based Information Retrieval System

4.CONCLUSION

This paper proposes a new method to get quick information on a digital screen via Wi-Fi, when a client makes a request to the server. Wireless QR (Server) respond’s for long range than QR code and it helps the client to read the information when the client is within that specified range. Nowadays we make use of QR code in hotels to get menu for ordering the food instead we can make use of this wireless QR which provides digital menus to customer and make menus available digitally, so that customers can view on personal electronic devices. In QR Code we cannot change the information once the QR Code is generated where as in Wireless QR information stored can be changed.

The proposed paper is focused on improving information accessibility by using Wi-Fi instead of QR codes. As a future scope, instead of using the NodeMCU (ESP8266) as a server, we can develop a mobile application that can perform the same functions. This application can act as a server, allowing users to access information. With the rise of mobile technology, this approach can provide a more user- friendly and convenient solution for modifying information stored in the server. This paper suggests that digitally we can place orders in wireless mode. This can be extended to many applications.

Acknowledgement

The author thank chandan kumar P,Deepak.S, Harshith K.N, Arunthathi Nath Choudhary for the support rendered in this work.

Declaration of Competing Interest

The author declares that the work is hers and no one can have any interest to influence this work reported in this paper

References

JA Dandge, R. Shirwadkar et.al 2016,Electric Switch On/Off System Using Android App Via Wi-Fi”, International Research Journal of Engineering and Technology (IRJET),03(03).1278-1282.

SP Makhanya et.al,2019, A Smart switch Control System Using ESP8266 Wi-Fi Module Integrated with an Android Application,IEEE 7th International Conference on Smart Energy Grid Engineering (SEGE),125-128. **DOI:**[10.1109/SEGE.2019.8859904](https://doi.org/10.1109/SEGE.2019.8859904)

Natesan and Abirami N., 2019. A Detailed Study of Client-Server and its Architecture using HTTP Protocol for data transmission,International journal of Advanced Research in Computer Science, 10(1), 1-6.

Santosh Kumar,2020 A Review on

Client - Server Based Applications

and Research Opportunity.

International Journal on Recent

Scientific Research., 10(07),33857-

33862.

DOI: <http://dx.doi.org/10.24327/ijrsr.2019.1007.3768>.

Singh, S & Vijayvargiya. A,2019, Implementation of Wi-Fi based information access system using ESP8266,Proceedings of 3rd International Conference on Trends in Electronics and Informatics (ICOEI),pp.475-478. IEEE. DOI: 10.1109/ICOEI.2019.8862501.

Pakalapati, S.et.al,2017,A prosthetic hand control interface using ESP8266 Wi-Fi module and Android application, proceedings of the International Conference on Innovations in Information, Embedded and Communication Systems(ICIIECS). DOI:[10.1109/ICIIECS.2017.8275978](http://dx.doi.org/10.1109/ICIIECS.2017.8275978)

https://randomnerdtutorials.com/esp8266-nodemcu-client-server-wi-fi/ Grigorik, 2018 ,Making the web faster with HTTP 2.0, Communications of the ACM, 56(2)12, 42–49, 2018.

Wikipedia. (2018). Client-Server:

https://simple.wikipedia.org/wiki/Client-server

P. E. Renaud, Introduction to Client-server systems : A practical guide for systems professionals, Wiley & Sons,2017.

ESP8266 Client-Server Wi-Fi

Communication Between Two

Boards (NodeMCU). 2021.