

QR Code-Based School Entry System Using ESP32-CAM and FPGA Cyclone II

Armin R. Paco, Richmond E. Hinggo, Joshua Alexei G. Occeña, and Kris Vallozo

CEIT Department, Surigao Del Norte State University

Narciso St., Surigao City, 8400 Philippines

apaco@ssct.edu.ph

joccena@ssct.edu.ph

rhinggo@ssct.edu.ph

kvallozo@ssct.edu.ph

Abstract— This project presents a smart access control system designed to enhance school security and streamline student entry using QR code authentication. The system leverages the ESP32-CAM module to capture and decode QR codes presented by students, while a Cyclone II EP2C5T144I8 FPGA processes and verifies the decoded data against a predefined list of authorized entries. Upon successful verification, the system activates a gate mechanism to grant access.

To optimize power efficiency and provide user-friendly control, a capacitive touch sensor is integrated as a soft on/off switch. This allows authorized personnel to activate or deactivate the system with a simple touch, eliminating the need for mechanical switches or unplugging components. The combination of embedded image processing, digital logic control, and human interface creates a cost-effective, scalable, and energy-efficient solution for educational institutions.

This project demonstrates the effective integration of IoT and FPGA technologies for real-time, secure access management in school environments.

touch-based user interaction. It uses the **ESP32-CAM module** to scan and decode QR codes embedded on student ID cards. The decoded data is then sent to a **Cyclone II EP2C5T144I8 FPGA**, which performs real-time verification by comparing the received data with a predefined set of valid student entries stored within the system.

To improve usability and power efficiency, the system includes a **capacitive touch sensor** that functions as an intuitive on/off switch. This sensor allows the school staff to easily activate or deactivate the system without physically disconnecting power, reducing energy consumption and unauthorized usage during off-hours. This also adds a layer of security, as the system is only operable when intentionally turned on.

The integration of embedded hardware (ESP32-CAM), programmable logic (FPGA), and a human interface (touch sensor) results in a highly effective access control solution. The system is cost-effective, scalable, and easily deployable in small to medium educational institutions. In future enhancements, it can be integrated with cloud databases, attendance tracking systems, and mobile apps for full-featured campus management.

I. INTRODUCTION

In today's educational environment, student safety and operational efficiency are more critical than ever. One of the key areas where schools can improve safety and reduce administrative workload is in managing student entry. Traditional access control methods such as manual attendance logs, ID card visual checks, or mechanical sign-in systems are not only time-consuming but also susceptible to errors and unauthorized access. These inefficiencies highlight the need for a smarter, automated solution.

This project introduces a QR code-based school entry system that combines image processing, digital logic, and

II. OBJECTIVES

1. **Automate Student Entry Monitoring:**
 - Replace manual log systems with a QR code-based entry system for speed and accuracy.
2. **Integrate a Capacitive Touch Sensor for System Control:**
 - Use a touch sensor as a soft on/off switch to activate or deactivate the scanning process.
 - Allow staff to easily enable or disable the system without unplugging devices or using physical switches.

3. **Implement ESP32-CAM for QR Code Scanning:**
 - Use ESP32-CAM to capture and decode QR codes from student ID cards in real time.
4. **Utilize Cyclone II FPGA for Verification and Control:**
 - Transfer decoded QR data to the FPGA for identity verification against stored valid entries.
 - Trigger gate opening mechanisms when authentication is successful.
5. **Enhance Entry Efficiency and Security:**
 - Reduce congestion and unauthorized access using rapid validation and automation.
6. **Design a Scalable, Energy-Conscious System:**
 - Build a low-power system with modular components for easy expansion, including database integration and cloud support in future versions.

III. PROJECT SCOPE

This project focuses on the design and implementation of a **QR code-based school entry system** using embedded and programmable hardware components. It integrates the **ESP32-CAM** for QR code scanning, the **Cyclone II EP2C5T144I8 FPGA** for data validation and control logic, and a **capacitive touch sensor** for user-friendly power control. The scope outlines the functional and technical boundaries of the system during its initial development phase.

Inclusions:

- **QR Code Scanning:** The system can scan QR codes from student ID cards using the ESP32-CAM module. It captures and decodes the QR data in real time.
- **Data Validation via FPGA:** The decoded QR data is sent to the FPGA, which compares the input against a preloaded list of valid student IDs. Access is granted if a match is found.
- **Entry Control Mechanism:** Upon successful validation, the FPGA triggers an output signal to open a gate or door lock through a relay or actuator.
- **Touch Sensor Integration:** A capacitive touch sensor acts as the on/off switch for the system. When touched, it powers the ESP32-CAM and FPGA modules to begin operation. Touching again will shut down the system to save power and prevent unauthorized use.
- **Standalone and Offline Operation:** The system functions independently without requiring a

constant internet connection, making it suitable for schools with limited network infrastructure.

Exclusions:

- **No cloud database or real-time internet synchronization** in the current implementation.
- **No facial recognition or biometric identification** features.
- **No automatic attendance recording or reporting** to external systems in this version.

Scalability:

The system design allows for future upgrades, such as:

- Integration with cloud databases or school management systems.
- Logging attendance data.
- Adding Wi-Fi-based remote control or mobile app interfaces.

IV. REVIEW OF RELATED LITERATURE

Related Literatures

Recent years have seen growing interest in smart access control systems, especially in educational and institutional environments where safety, efficiency, and automation are becoming top priorities. A number of related works and technologies provide foundational insight into the components and concepts used in this project.

QR Code-Based Identification Systems

QR codes have become a popular method for encoding and transmitting data due to their high speed of recognition, error correction capabilities, and compatibility with camera-based systems. In school and workplace environments, QR codes are increasingly used for digital ID cards, contactless attendance, and entry management systems. These systems are often preferred over biometric and RFID-based alternatives due to their lower cost and ease of deployment.

ESP32-CAM for Embedded Image Processing

The ESP32-CAM, developed by Espressif, is a compact microcontroller with integrated Wi-Fi and a camera interface. It is widely adopted in Internet of Things (IoT) applications for image capture, surveillance, and object recognition tasks. In related literature, it has been successfully used for facial recognition systems, surveillance drones, and license plate readers. Its capability to decode QR codes locally makes it suitable for standalone systems with limited processing requirements.

FPGA in Control and Verification Systems

Field Programmable Gate Arrays (FPGAs) like the Cyclone II EP2C5T144I8 are frequently used in security and control systems due to their parallel processing capabilities, reconfigurability, and real-time performance. They are often found in industrial automation, communication systems, and digital signal processing applications. Their use in access control projects allows for flexible logic design, enabling high-speed data comparison, signal routing, and hardware-level control of actuators or locks.

Touch-Based Interfaces for Embedded Systems

Touch sensors, especially capacitive types, are increasingly employed in embedded systems for user interaction due to their durability and intuitive use. Literature on touch-based controls highlights their effectiveness in power management, system activation, and security-related applications. In environments like schools where simplicity and reliability are vital, touch interfaces eliminate the need for mechanical switches, improving both usability and longevity.

Integrated Systems in Educational Environments

Various studies on smart school systems emphasize the benefits of integrating IoT and programmable logic for managing attendance, monitoring student movement, and improving campus security. However, most focus on RFID or biometric systems, which can be costly or raise privacy concerns. QR code-based systems, when combined with local processing and control hardware, provide a balance between cost, security, and ease of implementation.

V. METHODOLOGY

The development of this smart school entry system followed a modular and systematic approach, ensuring the seamless integration of hardware components and logical processes. The primary aim was to establish a secure and efficient QR code-based access system enhanced with a user-friendly and power-efficient touch interface.

System Activation and Power Control

A **capacitive touch sensor** is used as the main on/off switch for the entire system. When a user (typically a school staff member) touches the sensor, it sends a digital signal to activate the power rail or logic line that turns on both the **ESP32-CAM** and the **Cyclone II EP2C5T144I8 FPGA**. This method provides intuitive control, eliminates the need for physical switches, and enhances the longevity of the system by reducing mechanical wear. The same sensor can also be touched again to disable the system when not in use,

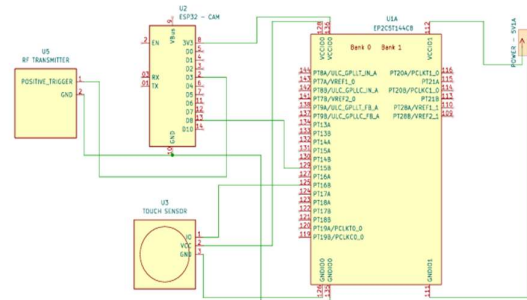
promoting power conservation and preventing unauthorized use during off-hours.

QR Code Scanning and Decoding

Once activated, the **ESP32-CAM module** initializes and begins scanning for QR codes. Each student ID is pre-assigned a unique QR code that encodes a student identifier. The ESP32-CAM captures an image of the presented QR code and decodes it locally using onboard libraries. This avoids the need for internet-based decoding and ensures faster response times.

Data Transmission to FPGA

After decoding, the ESP32-CAM transmits the student identifier to the FPGA via a UART (serial) connection or GPIO lines. The data is formatted into a protocol that the FPGA is programmed to recognize.



FPGA-Based Verification and Gate Control

The **Cyclone II FPGA** is programmed in VHDL to receive, parse, and compare the incoming data with a list of pre-authorized student IDs stored internally (either hardcoded or through memory interfacing). If a match is found, the FPGA activates a relay that controls the gate or door lock mechanism, allowing the student to enter. If no match is found, the gate remains closed, and an alert (e.g., buzzer or LED) can be triggered to notify the staff.

System Deactivation

Once the entry process is complete, the system can be turned off using the same touch sensor, returning it to a low-power idle state until the next activation.

VI. RESULTS AND CONCLUSION

Results

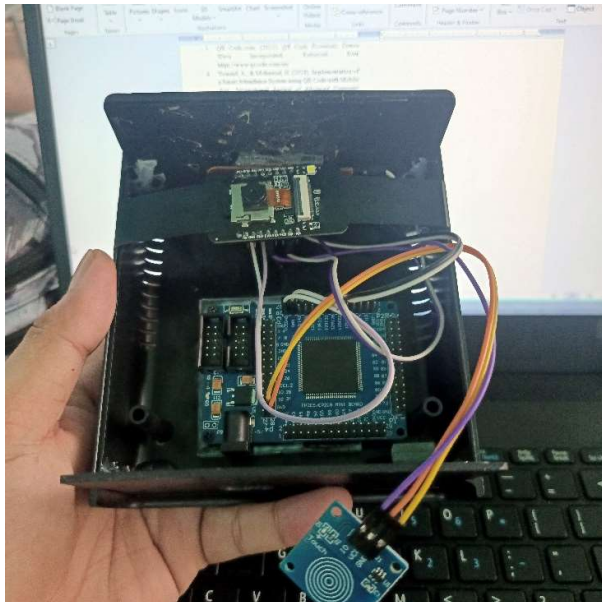
The implemented QR code-based entry system successfully demonstrated its core functionality during testing and trial runs. The integration of the ESP32-CAM for QR code scanning, the Cyclone II FPGA for verification, and the capacitive touch sensor for power control functioned as intended in a typical school entry scenario.

Upon system activation via the touch sensor, the ESP32-CAM was able to power up and begin scanning within a few seconds. QR codes printed on student ID cards were accurately captured and decoded in real time, with a consistent response time of under 2 seconds. The decoded data was reliably transmitted to the FPGA module through UART communication. The FPGA, programmed with valid student data, performed real-time comparisons and triggered the entry mechanism appropriately.

In cases of invalid or unrecognized QR codes, the system correctly denied access and activated the warning output (buzzer or LED), confirming the security feature of the project. The touch sensor effectively turned the system on and off as expected, with high sensitivity and no false triggering during idle periods. It also contributed to power saving by allowing the system to remain inactive when not needed.

Furthermore, the modular nature of the design allowed for scalability and easy troubleshooting. The logic implemented in the FPGA can be reprogrammed, and the ESP32-CAM can be updated with new QR code libraries or functionalities.

Inside Structure



Outside Structure



Conclusion

The project successfully met its objectives of creating a secure, efficient, and user-friendly school entry system using modern embedded and programmable logic technologies. The use of the ESP32-CAM enabled fast and accurate QR code scanning, while the Cyclone II FPGA provided robust logic control for real-time data verification and access management.

A key feature that elevated the practicality of the system was the incorporation of a capacitive touch sensor as a power switch. This feature not only enhanced ease of use but also contributed to energy efficiency and system durability by eliminating mechanical components.

The system demonstrated reliable performance under real-world conditions, offering a viable alternative to traditional attendance and access control methods in schools. It can be expanded in the future to support features such as cloud-based attendance logging, integration with student information systems, or mobile app control.

Overall, the project illustrates the effective integration of IoT and FPGA technologies in solving real-life problems in educational settings, paving the way for smarter, safer, and more automated school environments.

Recommendation

While the developed QR code-based school entry system using an ESP32-CAM, Cyclone II FPGA, and capacitive touch sensor has demonstrated efficiency and reliability, there are several opportunities for enhancement and expansion to improve functionality, usability, and security.

1. Integration with Database Systems

It is recommended to integrate the system with a **centralized database or cloud storage**. This would allow for dynamic updating of authorized student information, real-time attendance tracking, and data analytics. With database connectivity, administrators could add or remove student access remotely without reprogramming the FPGA.

2. Enhanced Security Features

To further secure the system, future iterations could include **two-factor authentication** by combining QR code scanning with facial recognition or RFID. This hybrid approach would greatly minimize the risk of spoofing or unauthorized access.

3. Mobile and Web Interfaces

Developing a **mobile or web-based interface** for administrators could simplify system management. Through this interface, QR codes could be generated, access logs monitored, and system statuses remotely controlled, making it more convenient for school personnel.

4. Automatic Attendance Recording

Incorporating an **automatic attendance logging mechanism** would help schools track student presence without manual input. Each QR scan could trigger a timestamped entry in the school's attendance records, saving time and reducing errors.

5. Power Optimization

Though the touch sensor already improves energy efficiency, further power-saving techniques—such as deep sleep modes for the ESP32-CAM or low-power FPGA configurations—should be explored, especially for battery-powered or solar-powered deployments.

6. Scalability and Multi-Gate Support

The current system can be expanded to support **multiple entry points or gates**, with each controlled by a dedicated or networked set of ESP32 and FPGA modules. This would be especially useful for larger campuses.

7. Environmental Protection

Finally, the system should be enclosed in **weatherproof and tamper-resistant housing** to ensure reliable operation in various environmental conditions and protect the components from damage or sabotage.

REFERENCES

1. Espressif Systems. (2020). *ESP32-CAM Development Board with OV2640 Camera Module Datasheet*. Retrieved from <https://www.espressif.com>
2. Altera Corporation. (2007). *Cyclone II Device Handbook, Volume 1*. Retrieved from <https://www.intel.com>
3. QR Code.com. (2021). *QR Code Essentials*. Denso Wave Incorporated. Retrieved from <https://www.qrcode.com/en/>
4. Youssef, A., & Mohamad, H. (2018). Implementation of a Smart Attendance System using QR Code with Mobile App. *International Journal of Advanced Computer Science and Applications*, 9(11), 234–239. <https://doi.org/10.14569/IJACSA.2018.091132>
5. Khan, A., & Rehman, S. (2019). Design and Development of a Touch Sensitive Switching System. *International Journal of Electronics and Computer Science Engineering*, 8(2), 144–149.
6. Shinde, S., & Patil, S. (2021). FPGA-Based Security System Using Face Recognition and Smart Lock Mechanism. *International Journal of Research in Engineering, Science and Management*, 4(4), 150–154.
7. Malik, M., & Raza, M. (2020). Role of IoT and Embedded Systems in Smart Campus Development. *Journal of Information and Communication Technology*, 19(2), 112–119. <https://doi.org/10.32890/jict2020.19.2.4>
8. QR Generator site. <https://www.the-qrcode-generator.com/>

Appendix

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

```
entity qr_trigger_touch is
port (
    clk : in std_logic;
    touch : in std_logic;
    trigger : out std_
);
end entity;
```

architecture Behavioral of qr_trigger_touch is

```
    constant COUNT_1MS : integer := 50_000;
    signal debounce_counter : integer range 0 to
COUNT_1MS := 0;
```



```

    signal touch_debounced : std_logic := '0';
    signal touch_sync      : std_logic_vector(1 downto 0) :=
"00";

```

```

    signal trigger_active : std_logic := '0';
    signal pulse_counter  : integer range 0 to COUNT_1MS
* 10 := 0;

```

```

begin
    process(clk)
    begin
        if rising_edge(clk) then
            touch_sync <= touch_sync(0) & touch;

            if touch_sync(1) /= touch_sync(0) then
                debounce_counter <= 0;
            elsif debounce_counter < COUNT_1MS then
                debounce_counter <= debounce_counter + 1;
            else
                touch_debounced <= touch_sync(1);
            end if;
        end if;
    end process;

```

```

    process(clk)
    begin
        if rising_edge(clk) then
            if touch_debounced = '1' and trigger_active = '0' then
                trigger_active <= '1';
                pulse_counter <= 0;
            elsif trigger_active = '1' then
                if pulse_counter < COUNT_1MS * 10 then -- 10
ms pulse
                    pulse_counter <= pulse_counter + 1;
                else
                    trigger_active <= '0';
                end if;
            end if;
        end if;
    end process;

```

```

    trigger <= trigger_active;

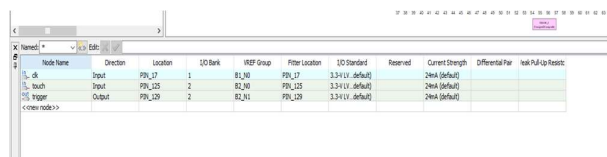
```

```

end architecture;

```

Pin Planner:



Node Name	Direction	Location	I/O Bank	KEF Group	Pin Location	I/O Standard	Reserved	Current Strength	Differential Pair	Is a Pull-Up Resistor
U0_0	Input	PP1_01	1	B1_N0	PP1_01	3.3V I/O (default)		2mA (default)		
U0_touch	Input	PP1_05	2	B2_N0	PP1_05	3.3V I/O (default)		2mA (default)		
U0_trigger	Output	PP1_09	2	B2_N1	PP1_09	3.3V I/O (default)		2mA (default)		

ESP32-CAM Code:

```

#include <Arduino.h>
#include <ESP32QRCodeReader.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <LittleFS.h>
#include "FS.h"
#include "SD_MMC.h"
#include <time.h>
#include <WiFi.h>

```

```

const char* ssid = "YourEnemy";
const char* password = "12345678";

```

```

ESP32QRCodeReader
reader(CAMERA_MODEL_AI_THINKER);

```

```

long timezone = 0;
byte daysavetime = 1;

```

```

const int ledPin = 4;
const int triggerPin = 13;

```

```

AsyncWebServer server(80);

```

```

const char* PARAM_INPUT_1 = "qrCode";
const char* PARAM_INPUT_2 = "role";
const char* PARAM_INPUT_3 = "delete";
const char* PARAM_INPUT_4 = "delete-user";

```

```

String inputMessage;
String inputParam;

```

```

void onQrCodeTask(void *pvParameters) {
    struct QRCodeData qrCodeData;

    while (true) {
        if (digitalRead(triggerPin) == HIGH) {
            Serial.println("Trigger received from FPGA! Scanning
for QR Code...");
            digitalWrite(ledPin, HIGH);

```

```

            if (reader.receiveQrCode(&qrCodeData, 100)) {
                // Process QR code as before...
            }

```

```

            digitalWrite(ledPin, LOW);
            delay(500); // Debounce
        }
    }
}

```

```

vTaskDelay(100 / portTICK_PERIOD_MS);
}
}

```

```

// Write to the SD card
void writeFile(fs::FS &fs, const char * path, const char *
message) {
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file) {
        Serial.println("Failed to open file for writing");
        return;
    }
    if(file.print(message)) {
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
    file.close();
}

// Append data to the SD card
void appendFile(fs::FS &fs, const char * path, const char *
message) {
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file) {
        Serial.println("Failed to open file for appending");
        return;
    }

    time_t t = file.getLastWrite();
    struct tm *tmstruct = localtime(&t);

    char bufferDate[50]; // Adjust buffer size as needed
    snprintf(bufferDate, sizeof(bufferDate), "%d-%02d-
%02d",
        (tmstruct->tm_year) + 1900,
        (tmstruct->tm_mon) + 1,
        tmstruct->tm_mday);
    char bufferTime[50]; // Adjust buffer size as needed
    snprintf(bufferTime, sizeof(bufferTime),
"%02d:%02d:%02d",
        tmstruct->tm_hour,
        tmstruct->tm_min,
        tmstruct->tm_sec);

    String lastWriteTime = bufferDate;
    String finalString = String(bufferDate) + "," +
String(bufferTime) + "," + String(message) + "\n";
    Serial.println(lastWriteTime);
    if(file.print(finalString.c_str())) {
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
    file.close();
}

```

```

}

// Append data to the SD card
void appendUserFile(fs::FS &fs, const char * path, const
char * message) {
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file) {
        Serial.println("Failed to open file for appending");
        return;
    }

    String finalString = String(message) + "\n";
    if(file.print(finalString.c_str())) {
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
    file.close();
}

void deleteFile(fs::FS &fs, const char *path) {
    Serial.printf("Deleting file: %s\n", path);
    if (fs.remove(path)) {
        Serial.println("File deleted");
    } else {
        Serial.println("Delete failed");
    }
}

// If the log.txt file doesn't exist, create a file on the SD card
and write the header
File file = SD_MMC.open("/log.txt");
if(!file) {
    Serial.println("Creating new log.txt file...");
    writeFile(SD_MMC, "/log.txt",
    "Date,Time,QR_Code,Role\r\n");
}
else {
    Serial.println("log.txt file already exists");
}
file.close();

// If the users.txt file doesn't exist, create a file on the SD
card and write the header
file = SD_MMC.open("/users.txt");
if(!file) {
    Serial.println("Creating new users.txt file...");
    writeFile(SD_MMC, "/users.txt", "QR_Code,Role\r\n");
}
else {
    Serial.println("users.txt file already exists");
}
file.close();
}

```

```

String processor(const String& var){
    return String("HTTP GET request sent to your ESP on
input field ("
    + inputParam + ") with value: " + inputMessage +
    "<br><a href=\"/\"><button class=\"button
button-home\">Return to Home Page</button></a>");
}

void deleteLineFromFile(const char* filename, int
lineNumber) {
    File file = SD_MMC.open(filename);
    if (!file) {
        Serial.println("Failed to open file for reading.");
        return;
    }

    // Read all lines except the one to delete
    String lines = "";
    int currentLine = 0;
    while (file.available()) {
        String line = file.readStringUntil('\n');
        if (currentLine != lineNumber) {
            lines += line + "\n";
        }
        currentLine++;
    }
    file.close();

    // Write back all lines except the deleted one
    file = SD_MMC.open(filename, FILE_WRITE);
    if (!file) {
        Serial.println("Failed to open file for writing.");
        return;
    }

    file.print(lines);
    file.close();
    Serial.println("Line deleted successfully.");
}

String getRoleFromFile(const char* filename, String
qrCode) {
    File file = SD_MMC.open(filename);
    if (!file) {
        Serial.println("Failed to open file for reading.");
        return "";
    }

    // Skip the header line
    file.readStringUntil('\n');

    // Read each line and check for QR Code
    while (file.available()) {
        String line = file.readStringUntil('\n');

```

```

        int commaIndex = line.indexOf(',');
        if (commaIndex > 0) {
            String fileQrCode = line.substring(0, commaIndex);
            String role = line.substring(commaIndex + 1);

            // Compare qrCode
            if (fileQrCode == qrCode) {
                file.close();
                role.trim(); // Remove any extra spaces or newline
                characters
                return role;
            }
        }
        file.close();
        return ""; // Return empty string if qrCode not found
    }

    void initLittleFS() {
        if (!LittleFS.begin()){
            Serial.println("An Error has occurred while mounting
LittleFS");
            return;
        }
    }

    void initWifi() {
        // Connect to Wi-Fi
        WiFi.begin(ssid, password);
        int connectAttempt = 0;
        Serial.println("Connecting to WiFi..");
        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
            connectAttempt++;
            if (connectAttempt == 10){
                ESP.restart();
            }
        }
        // Print ESP32 Local IP Address
        Serial.print("\nESP IP Address: ");
        Serial.println(WiFi.localIP());
    }

    void initTime() {
        Serial.println("Initializing Time");
        struct tm tmstruct;
        tmstruct.tm_year = 0;
        getLocalTime(&tmstruct);
        Serial.printf(
            "Time and Date right now is : %d-%02d-%02d
%02d:%02d:%02d\n", (tmstruct.tm_year) + 1900,
            (tmstruct.tm_mon) + 1, tmstruct.tm_mday,
            tmstruct.tm_hour, tmstruct.tm_min,

```



```

    tmstruct.tm_sec
);
}

void initSDCard() {
    if (!SD_MMC.begin("/sdcard", true)) {
        Serial.println("Card Mount Failed");
        return;
    }
    uint8_t cardType = SD_MMC.cardType();

    if (cardType == CARD_NONE) {
        Serial.println("No SD card attached");
        return;
    }

    Serial.print("SD Card Type: ");
    if (cardType == CARD_MMC) {
        Serial.println("MMC");
    } else if (cardType == CARD_SD) {
        Serial.println("SDSC");
    } else if (cardType == CARD_SDHC) {
        Serial.println("SDHC");
    } else {
        Serial.println("UNKNOWN");
    }

    uint64_t cardSize = SD_MMC.cardSize() / (1024 * 1024);
    Serial.printf("SD Card Size: %lluMB\n", cardSize);

    // If the log.txt file doesn't exist, create a file on the SD card
    and write the header
    File file = SD_MMC.open("/log.txt");
    if(!file) {
        Serial.println("log.txt file doesn't exist");
        Serial.println("Creating file...");
        writeFile(SD_MMC, "/log.txt",
        "Date,Time,QR_Code,Role\r\n");
    }
    else {
        Serial.println("log.txt file already exists");
    }
    file.close();

    // If the users.txt file doesn't exist, create a file on the SD
    card and write the header
    file = SD_MMC.open("/users.txt");
    if(!file) {
        Serial.println("users.txt file doesn't exist");
        Serial.println("Creating file...");
        writeFile(SD_MMC, "/users.txt", "QR_Code,Role\r\n");
    }
    else {
        Serial.println("users.txt file already exists");
    }
}

```

```

    file.close();
}

void setup() {
    Serial.begin(115200); // Initialize serial communication
    while (!Serial); // Do nothing if no serial port is opened
    (added for Arduinos based on ATMEGA32U4).

    reader.setup();
    Serial.println("\nSetup QRCode Reader");
    reader.beginOnCore(1);
    Serial.println("Begin on Core 1");
    xTaskCreate(onQrCodeTask, "onQrCode", 4 * 1024,
    NULL, 4, NULL);

    initWifi();
    initLittleFS();
    configTime(3600 * timezone, daysavetime * 3600,
    "time.nist.gov", "0.pool.ntp.org", "1.pool.ntp.org");
    initTime();
    initSDCard();

    pinMode(ledPin, OUTPUT);
    digitalWrite(ledPin, LOW);
    pinMode(triggerPin, INPUT);

    // Route for root / web page
    server.on("/", HTTP_GET, [](AsyncWebServerRequest
    *request){
        request->send(LittleFS, "/full-log.html");
    });
    // Route for root /add-user web page
    server.on("/add-user", HTTP_GET,
    [](AsyncWebServerRequest *request){
        request->send(LittleFS, "/add-user.html");
    });
    // Route for root /manage-users web page
    server.on("/manage-users", HTTP_GET,
    [](AsyncWebServerRequest *request){
        request->send(LittleFS, "/manage-users.html");
    });

    // Serve Static files
    server.serveStatic("/", LittleFS, "/");

    // Loads the log.txt file
    server.on("/view-log", HTTP_GET,
    [](AsyncWebServerRequest *request){
        request->send(SD_MMC, "/log.txt", "text/plain", false);
    });
    // Loads the users.txt file
    server.on("/view-users", HTTP_GET,
    [](AsyncWebServerRequest *request){
        request->send(SD_MMC, "/users.txt", "text/plain",
        false);
    });
}

```

```

});

// Receive HTTP GET requests on
<ESP_IP>/get?input=<inputMessage>
server.on("/get", HTTP_GET, [] (AsyncWebServerRequest
*request) {
// GET input1 and input2 value on
<ESP_IP>/get?input1=<inputMessage1>&input2=<inputM
essage2>
    if (request->hasParam(PARAM_INPUT_1) && request-
>hasParam(PARAM_INPUT_2)) {
        inputMessage = request-
>getParam(PARAM_INPUT_1)->value();
        inputParam = String(PARAM_INPUT_1);
        inputMessage += " " + request-
>getParam(PARAM_INPUT_2)->value();
        inputParam += " " + String(PARAM_INPUT_2);

        String finalMessageInput = String(request-
>getParam(PARAM_INPUT_1)->value()) + " " +
String(request->getParam(PARAM_INPUT_2)->value());
        appendUserFile(SD_MMC, "users.txt",
finalMessageInput.c_str());
    }
    else if (request->hasParam(PARAM_INPUT_3)) {
        inputMessage = request-
>getParam(PARAM_INPUT_3)->value();
        inputParam = String(PARAM_INPUT_3);
        if(request->getParam(PARAM_INPUT_3)-
>value()=="users") {
            deleteFile(SD_MMC, "/users.txt");
        }
        else if(request->getParam(PARAM_INPUT_3)-
>value()=="log") {
            deleteFile(SD_MMC, "/log.txt");
        }
    }
    else if (request->hasParam(PARAM_INPUT_4)) {
        inputMessage = request-
>getParam(PARAM_INPUT_4)->value();
        inputParam = String(PARAM_INPUT_4);
        deleteLineFromFile("/users.txt", inputMessage.toInt());
    }
    else {
        inputMessage = "No message sent";
        inputParam = "none";
    }
    request->send(LittleFS, "/get.html", "text/html", false,
processor);
});
// Start server
server.begin();
}
void loop() {
}

```

QR Code Generator:

