// Finish Line

#define BLYNK\_TEMPLATE\_ID "TMPL3qTE7ANmU"

#define BLYNK\_TEMPLATE\_NAME "unknown runners"

#define BLYNK\_AUTH\_TOKEN "WL5R-YxmO3c7Z1xjXse48NT0lRHusZ0p"

#include <WiFi.h>

#include <SPI.h>

#include <MFRC522.h>

#include <BlynkSimpleEsp32.h>

#include <NTPClient.h>

#include <WiFiUdp.h>

#define SS\_PIN 5

#define RST\_PIN 22

WidgetTerminal terminal(V2);

bool positionsDisplayed = false;

MFRC522 rfid(SS\_PIN, RST\_PIN);

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800, 60000); // IST offset

const char\* ssid = "Abbb";

const char\* password = "Princi@9";

BlynkTimer timer;

// Struct for each runner

struct Runner {

String id;

String startTime;

String endTime;

String totalTime;

bool dataUpdated; // Track if the runner's data has been updated

bool dataPrinted;

};

Runner runners[10];

int runnerCount = 0;

void syncStartTimes() {

Blynk.syncVirtual(V10);

}

void resetRunners() {

runnerCount = 0;

for (int i = 0; i < 10; i++) {

runners[i] = Runner(); // Reset to default

}

}

void displayPositions() {

// Simple bubble sort based on total time (in seconds)

for (int i = 0; i < runnerCount - 1; i++) {

for (int j = 0; j < runnerCount - i - 1; j++) {

int timeJ = convertToSeconds(runners[j].totalTime);

int timeJ1 = convertToSeconds(runners[j+1].totalTime);

if (timeJ > timeJ1) {

Runner temp = runners[j];

runners[j] = runners[j+1];

runners[j+1] = temp;

}

}

}

// Clear terminal

terminal.clear();

// Display positions

terminal.println("🏁 Final Positions:");

for (int i = 0; i < runnerCount; i++) {

terminal.print(String(i + 1) + ". Runner ID: ");

terminal.print(runners[i].id);

terminal.print(" | Time: ");

terminal.println(runners[i].totalTime);

}

terminal.flush();

}

bool allRunnersScanned() {

for (int i = 0; i < runnerCount; i++) {

if (!runners[i].dataUpdated) {

return false;

}

}

return true;

}

void setup() {

Serial.begin(9600);

SPI.begin();

rfid.PCD\_Init();

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\nWiFi connected");

Blynk.begin(BLYNK\_AUTH\_TOKEN, ssid, password);

timeClient.begin();

timeClient.update();

// Sync all start times from Blynk

resetRunners();

Blynk.syncVirtual(V10);

timer.setInterval(1000L, syncStartTimes);

Blynk.virtualWrite(V10, "");

Serial.println("RFID Finish Line Ready!");

}

String lastReceivedData = "";

void processData(String receivedData) {

if (receivedData.length() == 0) return; // Ignore empty

if (receivedData == lastReceivedData) return; // Same as last, ignore

lastReceivedData = receivedData;

int commaIndex = receivedData.indexOf(',');

if (commaIndex == -1) return; // Invalid format

if (runnerCount >= 10) {

Serial.println("Runner limit reached!");

return;

}

String runnerID = receivedData.substring(0, commaIndex);

String startTime = receivedData.substring(commaIndex + 1);

runners[runnerCount].id = runnerID;

runners[runnerCount].startTime = startTime;

runners[runnerCount].dataUpdated = false;

runners[runnerCount].dataPrinted = false;

runnerCount++;

Serial.println("Runner added: " + runnerID);

}

BLYNK\_WRITE(V10) {

String receivedData = param.asString();

processData(receivedData);

}

void loop() {

Blynk.run();

timer.run();

timeClient.update();

// Process and update runner data

for (int i = 0; i < runnerCount; i++) {

if (runners[i].dataUpdated && !runners[i].dataPrinted) {

// If the runner's data has been updated, print it

Serial.println("\nCurrent Runner Data:");

Serial.print("Runner ID: ");

Serial.println(runners[i].id);

Serial.print("Start Time: ");

Serial.println(runners[i].startTime);

Serial.print("End Time: ");

Serial.println(runners[i].endTime);

Serial.print("Total Time: ");

Serial.println(runners[i].totalTime);

Serial.println("--------------");

// Reset the dataUpdated flag after printing

runners[i].dataPrinted =true;

}

}

if (!rfid.PICC\_IsNewCardPresent() || !rfid.PICC\_ReadCardSerial()) return;

// Read UID

String scannedID = "";

for (byte i = 0; i < rfid.uid.size; i++) {

scannedID += String(rfid.uid.uidByte[i], HEX);

}

scannedID.toLowerCase();

Serial.println("Runner " + scannedID + " scanned at finish line");

bool matched = false;

for (int i = 0; i < runnerCount; i++) {

if (scannedID == runners[i].id) {

if (runners[i].dataUpdated) { // Check if data has already been updated

Serial.println("\nRunner " + scannedID + " has already been scanned.");

Blynk.virtualWrite(V1, "Runner " + scannedID + " has already been scanned.\n");

matched = true; // Indicate that the RFID matched but data is already updated

break;

}

matched = true;

String endTime = timeClient.getFormattedTime();

int startSec = convertToSeconds(runners[i].startTime);

int endSec = convertToSeconds(endTime);

int totalSec = endSec - startSec;

String totalTime = formatTime(totalSec);

// Update the runner's end time and total time

runners[i].endTime = endTime;

runners[i].totalTime = totalTime;

runners[i].dataUpdated = true; // Mark data as updated

String message = "\nRunner: " + scannedID +

"\nStart: " + runners[i].startTime +

"\nEnd: " + endTime +

"\nTotal Time: " + totalTime +

"\n-------------------------------\n";

Blynk.virtualWrite(V1, message);

Serial.println(message);

break;

}

}

if (!matched) {

Serial.println("Unrecognized RFID tag.");

Blynk.virtualWrite(V1, "Runner " + scannedID + " Unrecognized RFID tag.\n");

}

if (allRunnersScanned() && !positionsDisplayed) {

displayPositions();

positionsDisplayed = true;

}

rfid.PICC\_HaltA();

delay(3000); // Simple debounce

}

// Convert "HH:MM:SS" to seconds

int convertToSeconds(String timeStr) {

int h = timeStr.substring(0, 2).toInt();

int m = timeStr.substring(3, 5).toInt();

int s = timeStr.substring(6, 8).toInt();

return h \* 3600 + m \* 60 + s;

}

// Convert seconds to "HH:MM:SS"

String formatTime(int totalSeconds) {

int h = totalSeconds / 3600;

int m = (totalSeconds % 3600) / 60;

int s = totalSeconds % 60;

char buffer[9];

sprintf(buffer, "%02d:%02d:%02d", h, m, s);

return String(buffer);

}

//……,,...........

//..,.........

//Start Line

#define BLYNK\_TEMPLATE\_ID "TMPL3qTE7ANmU"

#define BLYNK\_TEMPLATE\_NAME "unknown runners"

#define BLYNK\_AUTH\_TOKEN "WL5R-YxmO3c7Z1xjXse48NT0lRHusZ0p"

#include <WiFi.h>

#include <SPI.h>

#include <MFRC522.h>

#include <BlynkSimpleEsp32.h>

#include <NTPClient.h>

#include <WiFiUdp.h>

#define SS\_PIN 5

#define RST\_PIN 22

MFRC522 rfid(SS\_PIN, RST\_PIN);

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800, 60000); // IST Timezone

const char\* ssid = "Abbb";

const char\* password = "Princi@9";

struct Runner {

String id;

String startTime;

};

Runner runners[10]; // Array to store data of up to 10 runners

int runnerCount = 0;

void setup() {

Serial.begin(9600);

SPI.begin();

rfid.PCD\_Init();

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\nWiFi connected");

Blynk.begin(BLYNK\_AUTH\_TOKEN, ssid, password);

timeClient.begin();

Serial.println("RFID Start Line Ready!");

}

void loop() {

Blynk.run();

timeClient.update();

if (!rfid.PICC\_IsNewCardPresent() || !rfid.PICC\_ReadCardSerial()) return;

String runnerID = "";

for (byte i = 0; i < rfid.uid.size; i++) {

runnerID += String(rfid.uid.uidByte[i], HEX);

}

bool runnerExist = false;

for (int i = 0; i < runnerCount; i++) {

// Check each runner in the array (example: print the IDs)

if(runners[i].id == runnerID){

Serial.print("\nRunner Id: "+ runnerID +" is already scanned.");

Blynk.virtualWrite(V0, "\nRunner " + runnerID + " is already scanned.");

runnerExist = true;

break;

}

}

if (!Blynk.connected()) {

Serial.println("Blynk not connected, retrying...");

}

if(!runnerExist && Blynk.connected()){

String startTime = timeClient.getFormattedTime();

runners[runnerCount].id = runnerID;

runners[runnerCount].startTime = startTime;

runnerCount++;

String data = runnerID + "," + String(startTime);

Blynk.virtualWrite(V10, data); // Save runner ID for finish line

Blynk.virtualWrite(V0, "\nRunner: " + runnerID + " | Start Time: " + startTime); // Show in app

Serial.print("\n Runner " + runnerID + " Start Time: " + startTime);

}

rfid.PICC\_HaltA();

delay(3000);

}