#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <WiFi.h>

#include <NTPClient.h>

#include <WiFiUdp.h>

#include <TimeLib.h>

#include <FirebaseESP32.h>

// LCD I2C address and dimensions

LiquidCrystal\_I2C lcd(0x27, 16, 2);

// WiFi credentials

const char\* ssid = "Xperia";

const char\* password = "11111111";

// Firebase credentials

#define FIREBASE\_HOST "https://l1project-681ca-default-rtdb.firebaseio.com"

#define FIREBASE\_API\_KEY "AIzaSyDPwA\_y2ZVycvIoek0bmrobve6BT1G-FZ0"

// NTP settings

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800, 60000); // NTP server, time offset in seconds (5 hours 30 minutes), update interval in milliseconds

// Pin definitions

#define LDRPIN 34

const int ledPin = 26;

const int ledPin1 = 27;

const int b1 = 25;

unsigned long startTime = 0;

bool timerRunning = false;

bool b1State = false;

unsigned long displayTimeStart = 0;

bool displayingTime = false;

String currentDate;

String currentTime;

String lastTermStartTime;

String lastTermRtime;

FirebaseData firebaseData;

FirebaseConfig firebaseConfig;

FirebaseAuth firebaseAuth;

bool b1low = false;

bool b1high = false;

int recordCounter = 0;

void setup() {

  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);

  pinMode(ledPin1, OUTPUT);

  pinMode(b1, OUTPUT);

  digitalWrite(b1, LOW); // Ensure b1 is initially low

  delay(500);

  // Initialize the LCD

  Wire.begin(21, 22); // Initialize I2C communication with custom pins

  lcd.init();

  lcd.backlight();

  // Display initial message

  lcd.setCursor(0, 0);

  lcd.print("Do Your Best..");

  // Connect to Wi-Fi

  WiFi.begin(ssid, password);

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

    delay(1000);

    Serial.println("Connecting to WiFi...");

  }

  Serial.println("Connected to WiFi");

  // Initialize the NTP client

  timeClient.begin();

  // Initialize Firebase

  firebaseConfig.host = FIREBASE\_HOST;

  firebaseConfig.api\_key = FIREBASE\_API\_KEY;

  firebaseAuth.user.email = "224094@example.com";

  firebaseAuth.user.password = "amma1234";

  Firebase.begin(&firebaseConfig, &firebaseAuth);

  Firebase.reconnectWiFi(true);

}

unsigned long parseTime(String timeStr) {

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

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

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

  int milliseconds = timeStr.substring(9, 12).toInt();

  return (hours \* 3600000UL) + (minutes \* 60000UL) + (seconds \* 1000UL) + milliseconds;

}

String displayTimeDifference(String startTime, String endTime) {

  unsigned long startMillis = parseTime(startTime);

  unsigned long endMillis = parseTime(endTime);

  unsigned long diffMillis = endMillis - startMillis;

  int hours = diffMillis / 3600000UL;

  diffMillis %= 3600000UL;

  int minutes = diffMillis / 60000UL;

  diffMillis %= 60000UL;

  int seconds = diffMillis / 1000UL;

  diffMillis %= 1000UL;

  int milliseconds = diffMillis;

  char buffer[16];

  sprintf(buffer, "%02d:%02d:%02d:%03d", hours, minutes, seconds, milliseconds);

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Timing:");

  lcd.setCursor(0, 1);

  lcd.print(buffer);

  return String(buffer);

}

void updateLaserState(bool state) {

  String path = "/laserstate";

  if (Firebase.setInt(firebaseData, path, state ? 1 : 0)) {

    Serial.println("Laser state updated successfully.");

  } else {

    Serial.print("Error updating laser state: ");

    Serial.println(firebaseData.errorReason());

  }

}

void loop() {

  timeClient.update();

  int val = analogRead(LDRPIN);

  Serial.print("Value: ");

  Serial.println(val);

  delay(5);

  if (val < 600) { // If the LDR value is less than 600

    digitalWrite(ledPin, HIGH); // Turn on green LED

    digitalWrite(ledPin1, LOW); // Turn off red LED

    if (!timerRunning) {

      startTime = millis(); // Start the timer

      timerRunning = true;

    }

    if (millis() - startTime >= 10000) { // If 10 seconds have passed

      digitalWrite(b1, HIGH); // Turn on b1

      b1State = true;

      // Update the state of b1

      if(!b1high){

        updateLaserState(true); // Update laser state in Firebase

        b1high = true;

      }

    }

  } else {

    digitalWrite(ledPin1, HIGH); // Turn on red LED

    digitalWrite(ledPin, LOW);   // Turn off green LED

    timerRunning = false; // Reset the timer

    digitalWrite(b1, LOW); // Turn off b1

    // if(!b1low || (b1 == LOW)){

    //   updateLaserState(false);

    //   b1low = true;

    // }

     // Update laser state in Firebase

    if (b1State) { // If b1 was previously high

      if(!b1low){

        updateLaserState(false);

        b1low = true;

      }

      Serial.println("Value exceeded 600, turning off b1");

      b1State = false; // Reset the state of b1

      // Get the current time from the NTP server

      time\_t rawTime = timeClient.getEpochTime();

      struct tm\* timeInfo = localtime(&rawTime);

      // Format and store date and time separately

      unsigned long currentMillis = millis() % 1000; // Get the milliseconds

      char dateBuffer[11]; // Buffer for date string (YYYY-MM-DD)

      char timeBuffer[14]; // Buffer for time string (HH:MM:SS:MMM)

      sprintf(dateBuffer, "%04d-%02d-%02d",

              timeInfo->tm\_year + 1900, timeInfo->tm\_mon + 1, timeInfo->tm\_mday);

      sprintf(timeBuffer, "%02d:%02d:%02d:%03lu",

              timeInfo->tm\_hour, timeInfo->tm\_min, timeInfo->tm\_sec, currentMillis);

      currentDate = String(dateBuffer);

      currentTime = String(timeBuffer);

      // Print current date and time to Serial

      Serial.print("Current date: ");

      Serial.println(currentDate);

      Serial.print("Current time: ");

      Serial.println(currentTime);

      // Initialize the record counter if it doesn't exist

      if (Firebase.getInt(firebaseData, "/runner/recordCounter")) {

        recordCounter = firebaseData.intData();

      } else {

        Firebase.setInt(firebaseData, "/runner/recordCounter", 0);

      }

      // Fetch the last term start time from Firebase

      if (Firebase.getString(firebaseData, "/runner/" + String(recordCounter) + "/Starttime")) {

        lastTermStartTime = firebaseData.stringData();

      } else {

        Serial.print("Error getting last term start time: ");

        Serial.println(firebaseData.errorReason());

      }

      // Fetch the last term Rtime from Firebase

      if (Firebase.getString(firebaseData, "/runner/" + String(recordCounter) + "/Rtime")) {

        lastTermRtime = firebaseData.stringData();

        Serial.print("Last term Rtime: ");

        Serial.println(lastTermRtime); // Display the last term Rtime on Serial monitor

      } else {

        Serial.print("Error getting last term Rtime: ");

        Serial.println(firebaseData.errorReason());

      }

      // Convert lastTermRtime and lastTermStartTime to milliseconds for comparison

      unsigned long lastTermRtimeMillis = parseTime(lastTermRtime);

      unsigned long lastTermStartTimeMillis = parseTime(lastTermStartTime);

      if (lastTermRtimeMillis < lastTermStartTimeMillis) {

        Serial.println("Faul");

        lcd.clear();

        lcd.setCursor(0, 0);

        lcd.print("Timing:");

        lcd.setCursor(0, 1);

        lcd.print("Faul Run");

        String buffer = "Faul Run";

        // Send timing buffer to Firebase

        if (Firebase.setString(firebaseData, "/runner/" + String(recordCounter ) + "/Timing", buffer)) {

          Serial.println("Timing buffer stored successfully.");

        } else {

          Serial.print("Error storing timing buffer: ");

          Serial.println(firebaseData.errorReason());

        }

      } else {

        // Display and store time difference

        String buffer = displayTimeDifference(lastTermStartTime, currentTime);

        // Send timing buffer to Firebase

        if (Firebase.setString(firebaseData, "/runner/" + String(recordCounter) + "/Timing", buffer)) {

          Serial.println("Timing buffer stored successfully.");

        } else {

          Serial.print("Error storing timing buffer: ");

          Serial.println(firebaseData.errorReason());

        }

      }

      // Read the current value of the record counter

      if (Firebase.getInt(firebaseData, "/runner/recordCounter")) {

        recordCounter = firebaseData.intData();

      } else {

        Serial.print("Error getting recordCounter: ");

        Serial.println(firebaseData.errorReason());

      }

      // Send date and time to Firebase under the new term branch

      String termPath = "/runner/" + String(recordCounter);

      if (Firebase.setString(firebaseData, termPath + "/Date", currentDate)) {

        Serial.println("Date stored successfully.");

      } else {

        Serial.print("Error storing date: ");

        Serial.println(firebaseData.errorReason());

      }

      if (Firebase.setString(firebaseData, termPath + "/EndTime", currentTime)) {

        Serial.println("Time stored successfully.");

      } else {

        Serial.print("Error storing time: ");

        Serial.println(firebaseData.errorReason());

      }

      // Start the time display timer

      displayTimeStart = millis();

      displayingTime = true;

      // Update the last term start time for the next term calculation

      // lastTermStartTime = currentTime;

    }

  }

  // Check if we need to revert back to "Do Your Best"

  if (displayingTime && millis() - displayTimeStart >= 40000) { // 20 seconds

    // Reset the board after stopping the player

    ESP.restart();

  }

}