```
/**
 * Advisory System v2.3
 * Christopher Hove 2024 (edited)
 * (Email: cahove16@gmail.com if questions / issues)
 *
 * What it does:
 * 1. Every time the magnetic sensor detects a rotation: a. Increments counter by 1,
b. Measure and updates humidity and temperature readings, c. Updates last update time
 * 2. Every PING (set to 3x per day): a. Read and updates humidity and temperature
readings (if not already on array), b. Updates last update time
 * Why PING? To ensure that the device is functioning properly and at a good
temperature + humidity even when there is no rain.
 *
 *
 * External Libraries Used:
 * 1. FirebaseClient (for Arduino devices), link:
https://github.com/mobizt/FirebaseClient
 * Specifics: Async functions USED and NO callback enabled.
 * 2. DHT11 (temperature and humidity sensor), link:
https://github.com/dhrubasaha08/DHT11
 *
 *
 * Device Used: Arduino Nano ESP32
 *
 *
 *
 *
 SYNTAX FROM FirebaseClient GITHUB:
 *
 * Firestore::Documents::commit(<AsyncClient>, <Firestore::Parent>, <Writes>,
<AsyncResultCallback>, <uid>);
 *
 * <AsyncClient> - The async client.
 * <Firestore::Parent> - The Firestore::Parent object included project Id and database
Id in its constructor.
 * <Writes> - The writes to apply.
 * <AsyncResultCallback> - The async result callback (AsyncResultCallback).
 * <uid> - The user specified UID of async result (optional).
 *
 * The Firebase project Id should be only the name without the firebaseio.com.
 * The Firestore database id should be (default) or empty "".
 *
 * The complete usage guidelines, please visit
https://github.com/mobizt/FirebaseClient
 */


// Set up
```

```
48
49  #include <Arduino.h>
50  #include <DHT11.h> // Include temperature / humidity sensor
51  #if defined(ESP32)
52  #include <WiFi.h>
53  #endif
54
55  // Definitions for boot count (variable saved when Arduino resets)
56  #define uS_TO_S_FACTOR 1000000ULL  /* Conversion factor for micro seconds to seconds
57  */
58
59  // VERY IMPORTANT: TIME BETWEEN PINGS IN SECONDS
60  #define TIME_TO_SLEEP  28800  /* Time ESP32 will go to sleep (in seconds) */
61
62  DHT11 dht11(2); // Digital I/O Pin 2 = for DHT11. Also note: Default delay of 500ms
63  between each sensor reading
64
65  #include <FirebaseClient.h>
66
67  // WiFi
68  #define WIFI_SSID ""
69  #define WIFI_PASSWORD ""
70
71  // Web API Key: The API key can be obtained from Firebase console > Project Overview >
72  Project settings.
73  #define API_KEY "AIzaSyDkSJ6dK6EgJG7kT0iWVXwztzmoYt2fqew"
74
75  // Project Specific
76  #define USER_EMAIL "testin7583cec@gmail.com" // Authentication Email (does not have to
77  be real)
78  #define USER_PASSWORD "it984465" // Authentication Password to Email
79  #define DATABASE_URL "cec-app-a569e.firebaseio.com"
80  #define FIREBASE_PROJECT_ID "cec-app-a569e" // project id
81
82  // FirebaseClient specific setups + WiFiClient
83  void authHandler();
84  void printResult(AsyncResult &aResult);
85  void printError(int code, const String &msg);
86  DefaultNetwork network; // initilize with boolean parameter to enable/disable network
87  reconnection
88  UserAuth user_auth(API_KEY, USER_EMAIL, USER_PASSWORD);
89  FirebaseApp app;
90
91  #if defined(ESP32)
92  #include <WiFiClientSecure.h>
93  WiFiClientSecure ssl_client;
94  #endif
95
```

```
using AsyncClient = AsyncClientClass;
AsyncClient aClient(ssl_client, getNetwork(network));
Firestore::Documents Docs;

AsyncResult aResult_no_callback;


// Variables

int temperature; // Temperature, Celcius - to report temp inside.
int humidity; // Humidity, Relative Humidity (%) - to report humidity inside.
RTC_DATA_ATTR int magnet_count = 0; // Variable to be saved over boots. In case it
fails to upload, it can upload detection later.
int magnet_pin = 5; // I/O pin for hall effect sensor
int state = 0; // digitalRead of I/O hall effect sensor pin
bool has_turned_off = false; // If magnet has turned off since last detection
bool magnet_wakeup = false; // Reason for wakeup
int failed_attempts = 0; // Number of attempts to upload to server >> max = 6

// Document Paths
String postPath = "arduino/post";
String measurementPath = "arduino/measurements";

// Time running
unsigned long time_running = millis();


void setup()
{
    // Serial Output + WiFi setup
    Serial.begin(115200);
    delay(1000); // Prepare Serial output
    pinMode(magnet_pin, INPUT);


    // DEEP SLEEP SETUP
    print_wakeup_reason();

    // Setup timer and external sensor to awake Arduino from deep sleep
    esp_sleep_enable_ext0_wakeup(GPIO_NUM_8,0); // Pin D5 = GPIO NUM 8 for Arduino
// Nano ESP32 (magnetic sensor detection) 0 = Magnet detected
    esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
    Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) + "
Seconds");



    // Connect to WiFi
```

```
140
141         WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
142
143         Serial.print("Connecting to Wi-Fi");
144         while (WiFi.status() != WL_CONNECTED)
145         {
146             Serial.print(".");
147             delay(300);
148         }
149         Serial.println();
150         Serial.print("Connected with IP: ");
151         Serial.println(WiFi.localIP());
152         Serial.println();
153
154         Firebase.printf("Firebase Client v%s\n", FIREBASE_CLIENT_VERSION);
155
156         Serial.println("Initializing app...");
157
158     #if defined(ESP32)
159         ssl_client.setInsecure();
160     #if defined(ESP8266)
161         ssl_client.setBufferSizes(4096, 1024);
162     #endif
163     #endif
164
165         initializeApp(aClient, app, getAuth(user_auth), aResult_no_callback);
166
167         authHandler();
168
169         // Binding the FirebaseApp for authentication handler.
170         // To unbind, use Docs.resetApp();
171
172         app.getApp<Firestore::Documents>(Docs);
173         aClient.setAsyncResult(aResult_no_callback);
174
175         Serial.println("End of setup()");
176     }
177
178     /*
179       Function to be called after each sleep
180
181       What it does:
182       1. IF magnet_wakeup == true: Updates Firebase temperature and humidity variables,
183     updateTime, and counter
```

```
  2. IF magnet_wakeup == false: Updates ONLY Firebase temperature and humidity variables, updateTime

*/
void loop()
{

    counter();

    authHandler(); // Some delay

    Docs.loop();

    state = digitalRead(magnet_pin);

    counter(); // Count if magnet pressed and wasn't pressed before

    // Magnetic sensor detects magnet signalling rain tilts bucket
    if (app.ready())
    {
      // Timestamp update
      // Writes addTimestamp_first(String fieldPath, String documentPath)
      Writes writes = addTimestamp_first("lastUpdate", postPath);


      // Counter increment

      if (magnet_wakeup == true) // woken up by magnet = increase counter
      {
      //void incr_counter(String fieldPath, String documentPath)
      writes.add(Write(incr_counter("counter", postPath), Precondition() /* currentDocument precondition */));
      }


      Document<Values::Value> updateDoc;
      updateDoc.setName(measurementPath);

      updateTempHumid(); // Update temp and humidity values
      Values::IntegerValue temp(temperature);
      Values::IntegerValue humid(humidity);
      updateDoc.add("lastTemperature", Values::Value(temp));
      updateDoc.add("lastHumidity", Values::Value(humid));

      writes.add(Write(DocumentMask(), updateDoc, Precondition()));

      // Commit Updates to Database
```

```
      // All Writes, DocumentTransform and Values::xxxx objects can be printed on
Serial port
      // You can set the content of write and writes objects directly with
write.setContent("your content") and writes.setContent("your content")

      String payload = Docs.commit(aClient, Firestore::Parent(FIREBASE_PROJECT_ID),
writes);

        if (aClient.lastError().code() == 0)
        {
          Serial.println(payload);
          magnet_count = 0;
          sleep();
        }
        else
        {
          printError(aClient.lastError().code(), aClient.lastError().message());
        }
    }

    printResult(aResult_no_callback); // Print Results of database changes

    quit_long();
}


void authHandler()
{
    // Blocking authentication handler with timeout
    unsigned long ms = millis();
    while (app.isInitialized() && !app.ready() && millis() - ms < 120 * 1000)
    {
        // The JWT token processor required for ServiceAuth and CustomAuth
authentications.
        JWT.loop(app.getAuth());
        printResult(aResult_no_callback);
        counter(); // Increment magnet counter if it was detected
        quit_long();
    }
}

// Check if it has been running too long
void quit_long() {
      if (millis() - time_running > 60 * 1000) // quit if trying for over 1 min
      {
        Serial.println("Error: Running for too long. Going to sleep.");
        sleep();
      }
```

```
}

// Go to sleep
void sleep()
{
  Serial.println("Sleep in 3 seconds");
  delay(3000);
  esp_deep_sleep_start();
}

void counter()
{
  state = digitalRead(magnet_pin);

  if (state == LOW && has_turned_off == true) // magnet pressed, incremenet counter
   {
    has_turned_off = false;
     ++magnet_count;
    Serial.print("Magnet count: ");
    Serial.println(magnet_count);
  }
  else if (state == HIGH && has_turned_off == false)
  {
    has_turned_off = true;
  }
}

// FUNCTIONS

// Increment Counter for "Writes" - Returns DocumentTransform
DocumentTransform incr_counter(String fieldPath, String documentPath)
{
  FieldTransform::Increment incr(Values::IntegerValue( (int) magnet_count )); //
Increment counter by magnet_counter
  FieldTransform::FieldTransform fieldTransforms(fieldPath, incr); // TESTING
  DocumentTransform transform(documentPath, fieldTransforms);

  return transform;
}

// Add Timestamp for "Writes" + Creates new "Write" element - Returns Writes
Writes addTimestamp_first(String fieldPath, String documentPath)
{
  FieldTransform::SetToServerValue setTime(FieldTransform::REQUEST_TIME); //
FieldTransform::REQUEST_TIME
  FieldTransform::FieldTransform fieldTransform(fieldPath, setTime);
   DocumentTransform transform(documentPath, fieldTransform);
   Writes write(Write(transform, Precondition() /* currentDocument precondition */));
```

```
   return write;
}

// Updates int temperature and int humidity Values
void updateTempHumid(){
    int result = dht11.readTemperatureHumidity(temperature, humidity);

    // Check the results of the readings.
    // If the reading is successful, print the temperature and humidity values. If
there are errors, print the appropriate error messages.
    if (result == 0) {
        Serial.print("Temperature: ");
        Serial.print(temperature);
        Serial.print(" °C\tHumidity: ");
        Serial.print(humidity);
        Serial.println(" %");
    } else {
        // Print error message based on the error code.
        Serial.println(DHT11::getErrorString(result));
    }
}

// Method that prints reason for ESP32 wakeup
void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;
  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal
using RTC_IO"); magnet_wakeup = true; ++magnet_count; break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal
using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer");
magnet_wakeup = false; break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad");
break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep:
%d\n",wakeup_reason); break;
  }
  Serial.print("Magnet wakeup: ");
  Serial.println(magnet_wakeup);
}

// Print Results of Data Linkage
void printResult(AsyncResult &aResult)
```

```
372 {
373     if (aResult.isEvent())
374     {
375         Firebase.printf("Event task: %s, msg: %s, code: %d\n", aResult.uid().c_str(),
376 aResult.appEvent().message().c_str(), aResult.appEvent().code());
377     }
378
379     if (aResult.isDebug())
380     {
381         Firebase.printf("Debug task: %s, msg: %s\n", aResult.uid().c_str(),
382 aResult.debug().c_str());
383     }
384
385     if (aResult.isError())
386     {
387         Firebase.printf("Error task: %s, msg: %s, code: %d\n", aResult.uid().c_str(),
388 aResult.error().message().c_str(), aResult.error().code());
389     }
390
391     if (aResult.available())
392     {
393         Firebase.printf("task: %s, payload: %s\n", aResult.uid().c_str(),
394 aResult.c_str());
395     }
396 }
397
398 void printError(int code, const String &msg)
399 {
400     Firebase.printf("Error, msg: %s, code: %d\n", msg.c_str(), code);
401 }
```