Universidad de Guadalajara

centro universitario de ciencias exactas e ingenierías

Google App Script

Omar Sánchez Gudiño.

Mtra en computo aplicado | Internet de las cosas.

13 de marzo de 2025

Resumen

Este artículo presenta el desarrollo de una Web App con Google Apps Script para recibir datos de un dispositivo IoT (ESP32) mediante solicitudes HTTP GET y almacenarlos en Google Sheets. Se detallan los pasos para crear la hoja de cálculo, configurar el script y probar la API mediante una URL personalizada. Esta solución permite almacenar y visualizar datos en tiempo real sin necesidad de infraestructura adicional. Se sugieren mejoras como autenticación, validación de datos e integración con herramientas de análisis para optimizar su funcionalidad.

Índice

1. Objetivo.

El objetivo de este artículo es desarrollar una Web App utilizando Google Apps Script que permita la recepción, procesamiento y almacenamiento de datos enviados por un dispositivo IoT (ESP32) a través de solicitudes HTTP GET, integrando estos datos de manera automática en una hoja de cálculo de Google Sheets. Este proyecto busca proporcionar una solución accesible y eficiente para la recopilación y gestión de datos en tiempo real, eliminando la necesidad de infraestructura adicional y aprovechando los servicios en la nube de Google. Se pretende ofrecer una metodología clara para la implementación de este sistema, detallando la configuración de Google Sheets, la creación del script en Google Apps Script y el proceso de prueba mediante solicitudes web. Además, se plantea la posibilidad de ampliar la funcionalidad del sistema mediante mejoras futuras, como la implementación de mecanismos de autenticación para restringir el acceso, la validación de datos para garantizar su integridad y la integración con herramientas de análisis para facilitar la interpretación y visualización de la información recopilada.

2. Introducción.

El avance de la tecnología IoT (Internet de las Cosas) ha permitido la interconexión de dispositivos para la recopilación, transmisión y análisis de datos en tiempo real. Sin embargo, gestionar y almacenar esta información de manera eficiente puede representar un desafío, especialmente para proyectos que buscan soluciones accesibles y de bajo costo. En este contexto, los servicios en la nube ofrecen alternativas viables para el almacenamiento y procesamiento de datos sin la necesidad de infraestructura adicional. Google Apps Script es una plataforma basada en JavaScript que permite la automatización e integración de servicios de Google, como Google Sheets, Gmail y Drive. En este artículo, se describe el desarrollo de una Web App utilizando Google Apps Script para recibir datos de un dispositivo IoT, específicamente una ESP32, a través de solicitudes HTTP GET y almacenarlos automáticamente en una hoja de cálculo de Google Sheets. La ESP32 es un microcontrolador ampliamente utilizado en proyectos de IoT debido a su conectividad WiFi y Bluetooth, su bajo consumo energético y su facilidad de programación. En este proyecto, la ESP32 se encarga de recopilar datos de sensores ambientales, como temperatura, humedad, y enviarlos mediante una solicitud HTTP a la Web App, que los procesa y almacena en Google Sheets. El artículo cubre el proceso completo de implementación, incluyendo la configuración de la hoja de cálculo, la creación del script en Google Apps Script y la prueba de la aplicación mediante el envío de datos simulados desde una URL. Además, se exploran posibles mejoras y ampliaciones del sistema, como la autenticación de usuarios, la validación de datos y la integración con herramientas de análisis. Con esta solución, se ofrece una alternativa sencilla y eficiente para la gestión de datos IoT en tiempo real, facilitando su almacenamiento, consulta y análisis sin necesidad de infraestructura adicional ni conocimientos avanzados en servidores o bases de datos.

3. Desarrollo.

I. App script.

El código utilizado se puede revisar en la sección de Anexo.

I.1. Recepción de parámetros desde una solicitud HTTP GET.

La función doGet(e) es el punto de entrada del script y se ejecuta automáticamente cuando se recibe una solicitud HTTP GET. La variable e contiene todos los parámetros enviados en la URL de la solicitud. En esta parte del código, se extraen los valores de los parámetros enviados por el dispositivo IoT:

- iddevice: Identificador único del dispositivo que envía los datos.
- temp: Valor de la temperatura medida.
- humidity: Nivel de humedad registrado.
- gas: Nivel de gas detectado (como CO2 u otros gases).
- pressure: Valor de la presión atmosférica registrada.

Estos valores son recuperados usando e.parameter."nombre_del_parametroz se almacenan en variables para ser usados posteriormente.

I.2. Acceso a la hoja de cálculo de Google Sheets.

Para almacenar los datos recibidos, el script necesita acceder a una hoja de cálculo específica en Google Sheets. Esto se hace utilizando la función SpreadsheetApp.openById(id), donde id es el identificador único del documento. Luego, se accede a la primera hoja de la hoja de cálculo con getSheets()[0], asumiendo que los datos deben almacenarse en la primera pestaña del archivo. Además, la función ReadSensor() lee los valores del sensor y almacena los datos en un arreglo de 6 bytes llamado SensorValueCatch[].

I.3. Determinación de la siguiente fila vacía.

En una hoja de cálculo, cada fila representa un conjunto de datos de un solo envío desde el dispositivo IoT. Para insertar los nuevos datos en la fila correcta, el script primero obtiene la última fila con datos utilizando sheet.getLastRow(), y luego le suma 1. Esto permite que los datos se agreguen en la primera fila vacía disponible.

I.4. Inserción de los datos en la hoja de cálculo.

Cada valor obtenido de la solicitud GET es almacenado en la hoja de cálculo en su respectiva columna. Las columnas están organizadas de la siguiente manera:

Columna	Dato
A (1)	`iddevice`
B (2)	`temp`
C (3)	`humidity`
D (4)	`gas`
E (5)	`pressure`

Tabla 1: Asignación de datos en la hoja de cálculo

El script usa getRange(row, column).setValue(value) para colocar los valores en las celdas correspondientes. row es la fila determinada previamente (lastRow), y column indica en qué columna se debe insertar cada dato.

I.5. Respuesta a la solicitud

Después de insertar los datos en la hoja de cálculo, el script genera una respuesta utilizando ContentService.createTextOutput(). Esta respuesta será devuelta al cliente (el navegador o el dispositivo IoT que hizo la solicitud) e indicará si los datos fueron almacenados correctamente o si ocurrió un error. Si la operación es exitosa, se devuelve un mensaje confirmando el número de fila en la que se guardaron los datos. Si ocurre un error en cualquier parte del proceso, el script captura el error y devuelve un mensaje con la descripción del problema.

II. Código para la ESP32

II.1. Inclusión de Librerías

```
#include <WiFi.h>
#include <HTTPClient.h>
```

WiFi.h: Permite la conexión del ESP32 a una red WiFi. HTTPClient.h: Facilita el envío de solicitudes HTTP al servidor (Google Apps Script en este caso).

III. Definición de Credenciales WiFi y URL del Script

```
const char* ssid = "NETWORK_NAME";
const char* password = "PASSWORD";
String scriptURL = "SCRIPT_URL"
```

ssid: Nombre de la red WiFi a la que se conectará la ESP32. password: Contraseña de la red WiFi. scriptURL: URL del Google Apps Script, que recibe los datos y los almacena en Google Sheets.

III.1. Configuración inicial (setup())

```
void setup()
{
    Serial.begin(115200);
    WiFi.begin(ssid, password);
```

```
    Serial.print("Conectando a WiFi");
    while (WiFi.status() != WL_CONNECTED)
    {
      delay(500);
      Serial.print(".");
    }
    Serial.println("\nConectado a WiFi");
}
```

En este bloque de código se inicia la comunicación serial a 115200 baudios para depuración, además se inicia la conexión WiFi con las credenciales proporcionadas. El código mantiene a la ESP32 intentando conectarse hasta que se establezca la conexión y, una vez conectada, entonces muestra en el serial un mensaje informando do que la conexión se realizo de manera exitosa.

III.2. Void Loop()

III.3. Verificar conexión wiFi

```
if (WiFi.status() == WL_CONNECTED)
```

Si la ESP32 está conectada a WiFi, se ejecuta el envío de datos, en caso de desconexión, intentará reconectarse más adelante.

III.4. Crear objeto HTTP y datos simulados

```
HTTPClient http;
String iddevice = "ESP32-01";
float temp = random(20, 30);
float humidity = random(40, 60);
float gas = random(100, 300);
float pressure = random(900, 1100);
```

HTTPClient http, Crea un objeto para realizar solicitudes HTTP. iddevice, Identificador del dispositivo (ESP32-01). Valores Simulados (random()), Se generan datos de sensores en un rango específico:

- Temperatura: 20 - 30 °C
- Humedad: 40 - 60 %
- Gas: 100 - 300 ppm
- Presión: 900 - 1100 hPa

III.5. Construcción de la URL con los datos

```
String url = scriptURL + "?iddevice=" +
             iddevice +
             "&temp=" +
             String(temp) +
             "&humidity=" +
             String(humidity) +
             "&gas=" +
             String(gas) +
             "&pressure=" +
             String(pressure);
```

Se concatena la URL del script con los datos generados, dando como resultado una URL como la siguiente: https://script.google.com/macros/s/IDDELAPPSCRIPT/exec?iddevice=ESP32-01&temp=25.6&humidity=50&gas=150&pressure=1000. Esto permite que el Google Apps Script reciba los datos como parámetros en la solicitud GET.

III.6. Envío de datos a Google Sheets

```
if (WiFi.status() == WL_CONNECTED)
```

III.7. Envío de datos a Google Sheets

```
int httpResponseCode = http.GET();
```

http.begin(url), inicia la solicitud HTTP con la URL creada. http.GET(), envía la solicitud GET al servidor y almacena la respuesta.

III.8. Validación de la respuesta del servidor

```
if (httpResponseCode > 0)
{
    String response = http.getString();
    Serial.println("Respuesta del servidor:"
    + response);
}
else
{
```

```
    Serial.print("Error en solicitud:");
    Serial.println(httpResponseCode);
}
http.end();
```

Si el código de respuesta HTTP es mayor que 0, se imprime la respuesta del servidor, esto quiere decir que la comunicación con google sheets fue exitosa. Si hay un error, se imprime el código de error en la consola serial. http.end(), finaliza la conexión HTTP para liberar memoria.

III.9. Manejo de reconexión wiFi

```
else
{
    Serial.println("WiFi desconectado,
    reconectando...");
    WiFi.begin(ssid, password);
}
```

Si la ESP32 pierde la conexión WiFi, intentará reconectarse.

III.10. Espera antes de la próxima transmisión

```
http.begin(url);
delay(60000);
```

Espera de 60 segundos (60000 ms) antes de enviar nuevos datos, esto evita sobrecargar el servidor con demasiadas solicitudes y ademas evita el enviar mucho datos de forma irrelevante para la actividad.

4. Resultados.

iddevice	temp	humidity	gas	pressure
ESP32-01	24.00	49.00	157.00	960.00
ESP32-01	25.00	53.00	163.00	870.00

Figura 1: Imagen mostrando como se guardaban los datos en el google sheet.

Como se puede apreciar los datos se guardaban de manera adecuada en el google sheet cada que este enviaba información nueva.

5. Conclusión.

En este artículo, se ha desarrollado una Web App utilizando Google Apps Script para recibir y almacenar datos de un dispositivo IoT ESP32 en una hoja de cálculo de Google Sheets mediante solicitudes HTTP GET. Se ha detallado el proceso paso a paso, desde la configuración de la hoja de cálculo hasta la implementación del script y las pruebas del sistema, demostrando cómo es posible gestionar datos en tiempo real sin la necesidad de servidores dedicados ni infraestructura compleja. El uso de la ESP32 en este proyecto resalta su versatilidad como microcontrolador para aplicaciones IoT, gracias a su conectividad WiFi integrada y su capacidad para interactuar con sensores externos. En este caso, se ha utilizado para recopilar y enviar datos de temperatura, humedad, calidad del aire y presión, permitiendo su almacenamiento automatizado en Google Sheets. Este enfoque facilita la visualización y análisis de la información, ofreciendo una solución práctica para el monitoreo de parámetros ambientales. Entre las principales ventajas de esta implementación destacan su bajo costo, la facilidad de integración con los servicios en la nube de Google y la posibilidad de escalar el sistema sin requerir grandes modificaciones. No obstante, la solución puede mejorarse con diversas optimizaciones, como:

- Autenticación y seguridad: Implementar medidas de autenticación para restringir el acceso y evitar la manipulación de datos no autorizada.
- Validación de datos: Incorporar mecanismos de verificación para asegurar la integridad y coherencia de la información recibida.
- Notificaciones y alertas: Integrar el sistema con servicios como Gmail o Telegram para enviar alertas automáticas en caso de valores críticos en los sensores.
- Análisis avanzado de datos: Conectar Google Sheets con herramientas como Google Data Studio o lenguajes como Python para análisis más detallados y predicciones basadas en aprendizaje automático.

En conclusión, el uso de Google Apps Script en combinación con la ESP32 y Google Sheets proporciona una alternativa eficiente y flexible para la gestión de datos IoT. Su implementación es sencilla, escalable y permite el monitoreo en tiempo real, lo que la convierte en una opción ideal para proyectos de automatización, control ambiental y otras aplicaciones IoT que requieran almacenamiento y análisis de datos sin necesidad de infraestructura adicional.

6. Recursos.

I. Repositorio en Github.

En el siguiente enlace pueden encontrar el script generado para este experimento: https://github.com/OmarGudi/Google-App-Script

7. Anexo.

I. Codigo usado para la función get.

```
function doGet(e)
{
  try
  {
    var iddevice = e.parameter.iddevice;
    var temp = e.parameter.temp;
    var humidity = e.parameter.humidity;
    var gas = e.parameter.gas;
    var pressure = e.parameter.pressure;
    var ss = SpreadsheetApp.openById('S_ID');
    var sheet = ss.getSheets()[0];
    var lastRow = sheet.getLastRow() + 1;
    sheet.getRange(lastRow, 1).setValue(iddevice);
    sheet.getRange(lastRow, 2).setValue(temp);
    sheet.getRange(lastRow, 3).setValue(humidity);
    sheet.getRange(lastRow, 4).setValue(gas);
    sheet.getRange(lastRow, 5).setValue(pressure);
    return ContentService.createTextOutput("Datos guardados correctamente en la fila "
    + lastRow);
  }
  catch (error)
  {
    return ContentService.createTextOutput("Error: " + error.message);
  }
}
```