

Una solución IoT que basa en ESP32, Mosquito, Node-RED y recursos de Google

Problemática

Google ofrece la posibilidad de desarrollar soluciones serverless, es decir, sin necesidad de que el desarrollador tenga que implementar un servidor e involucrando muchas herramientas como almacenamiento, cómputo en la nube, inteligencia artificial. Debido a los costos que tiene el uso del almacenamiento en la nube, resulta atractivo para las soluciones IoT poder brindarle al usuario la posibilidad de elegir su propio método de almacenamiento, que bien puede ser en la base de datos de nuestro servidor privado, pero también en GoogleDrive, DropBox, OneDrive, etc. Lograr que los datos de sensores remotos IoT se registren en google sheet representaría un importante avance en esa dirección.

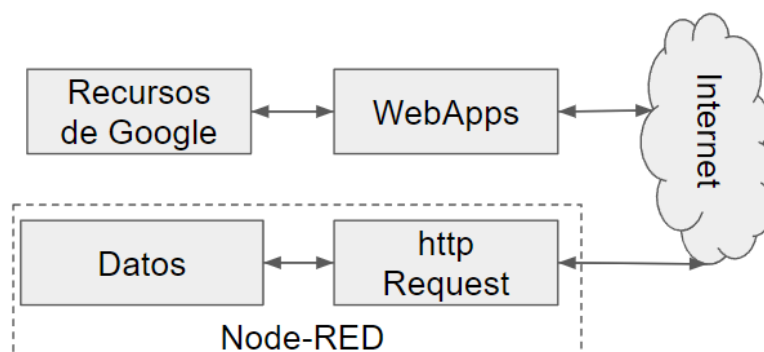
Problema específico a resolver

Se requiere una solución para que datos que se encuentran en Google puedan viajar hasta un servidor como Node-RED y viceversa. Es necesario deducir las condiciones en las que esto es posible.

Restricciones:

- Del lado de Google usaremos Google WebApps. Esto a su vez implica que el protocolo de comunicación es http
- Del otro lado, usaremos Node-RED como intermediario entre otras soluciones.
- Es necesario identificar las condiciones en que es posible lograr esta conexión.

Diseño previo



Sprint #1. Crear un hola mundo desde la WebApp a Node-RED

En este sprint buscaremos implementar la solución más simple posible que permite el envío de un saludo creado en una WebApp y visualizarlo en Node-RED

Sprint1.Tarea 1. Implementar una WebApp que devuelve un saludo

Paso1. Escribir el código de la WebApp

Código para una WebApp que devuelve un saludo



```
var saludo='Hola. Soy una WebApp hecha en Google Script. Como tal estoy representada en internet por un endpoint que es lo mismo que mi URL. Si la conoces, podrás usarla para ver mi saludo para lo cual debes usar herramientas con protocolo http, por ejemplo un navegador o comandos hechos para http request'

// doGet() es la función que responde a los request tipo GET
function doGet() {
  // Empaquetamiento de la información para que pueda viajar por http
  var miTextOutput=ContentService.createTextOutput(saludo);
  return miTextOutput;
}
```

Cómo se implementa la WebApp:

Crea un proyecto en Google Apps Script > Escribe el código anterior > prueba que no tenga errores con “Ejecutar” > Conviértelo en una WebApp así: Implementar > Nueva implementación > haz la siguiente configuración > cuando termines, copia la URL de la WebApp

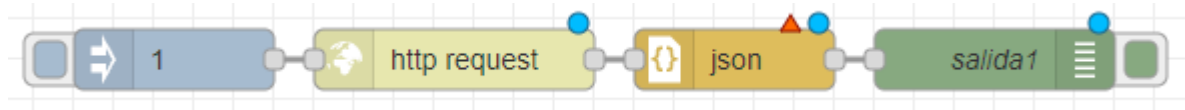
Nueva implementación

Seleccionar tipo 	Configuración 
Aplicación web	<div>Descripción <div>Nueva descripción</div><div>saludo2</div></div> <div>Aplicación web <div>Ejecutar como</div><div>Yo (abet.admin@e3t.uis.edu.co) ▼</div><div>La ejecución de la aplicación web se autorizará con los datos de tu cuenta.</div><div>Quién tiene acceso</div><div>Cualquier usuario ▼</div></div>

Análisis de la configuración realizada:

- Una incorrecta configuración puede no servir
- Nueva descripción. Aquí no hay problema con lo que se escriba
- Imaginemos que la WebApp se consume desde un navegador, si allí el usuario se loguea con una cuenta de google tiene sentido, seleccionar que la WebApp sea ejecutada por un usuario logueado. Pero ese no es nuestro caso, porque llamaremos la WebApp desde Node-RED que no tiene nada que ver con un navegador.
- Ejecutar como: La ventaja de escoger "Yo". Es la única opción que permite que más abajo, en "Quién tiene acceso" se escoja "Cualquier usuario", de manera que no sea necesario loguearse a una cuenta de google para acceder a la WebApp. Una pregunta lógica sería: si escojo "Yo", entonces solo yo puedo usar la WebApp, la respuesta es que, si más abajo escoge "Cualquier usuario" entonces la puede usar cualquier usuario. El "yo" lo que significa es que la WebApp correrá con mis permisos. Por ejemplo, si como consecuencia de la WebApp se escribe algo en un google sheet, eso será escrito con mi usuario. Es como si yo les dijera, pásenme datos por la WebApp que yo los escribo.

Paso2. Preparar el flujograma en Node-RED



Esos bloques de derecha a izquierda son tipo: Inject, http request, json, debug

Configuración del bloque tipo Inject

Edit inject node




Delete Cancel Done


Properties

Name


msg.payload = 1

Configuración del bloque tipo http request - lo más relevante es que en el campo URL se escriba la URL de la WebApp creada en el paso anterior.



Properties




 Method

GET


 URL

https://script.google.com/macros/s/AKfycbyjM5mV


 Payload

Ignore

☐ Enable secure (SSL/TLS) connection


☐ Use authentication

☐ Enable connection keep-alive

☐ Use proxy




☐ Only send non-2xx responses to Catch node


☐ Disable strict HTTP parsing


 Return


a UTF-8 string

Configuración del bloque tipo json



Properties




 Action

Always convert to JSON String


 Property

msg. payload

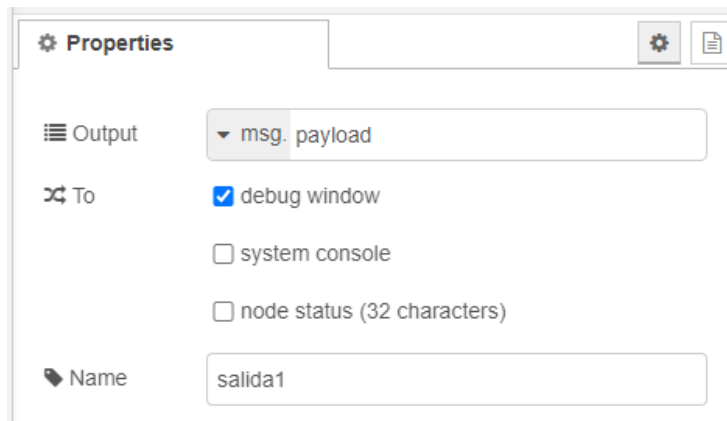

 Name

Name

 Object to JSON options

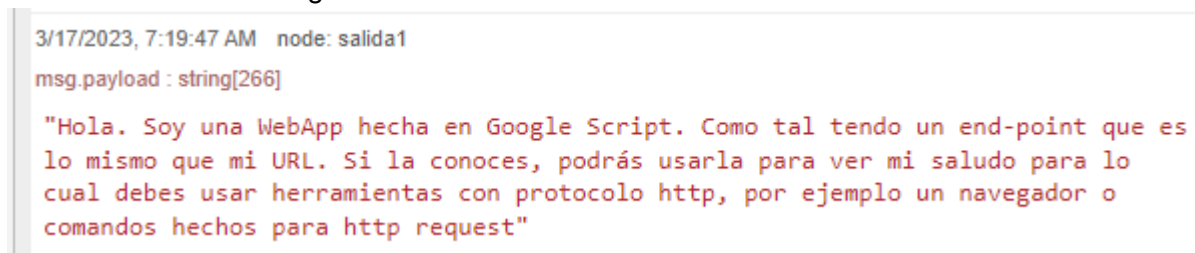
☐ Format JSON string

Configuración del bloque tipo debug



paso 3. Validación

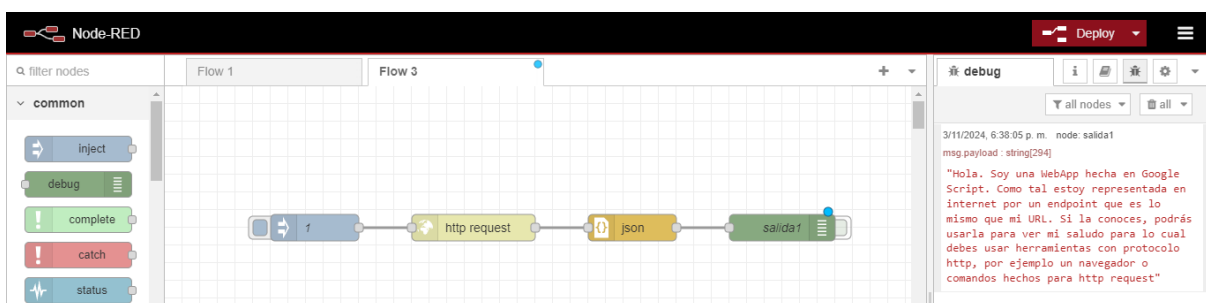
En la ventana del debug veremos el saludo



Paso 4. conclusiones de la experiencia

- si es posible consumir una WebApp desde Node-RED
- hacerlo con Node-RED significa poder hacerlo desde cualquier sistema de programación que tenga la posibilidad de implementar funciones de http request. Incluso quizá desde una ESP32 con las librerías adecuadas

Resultados de lo obtenido en mi usuario de Node-RED:

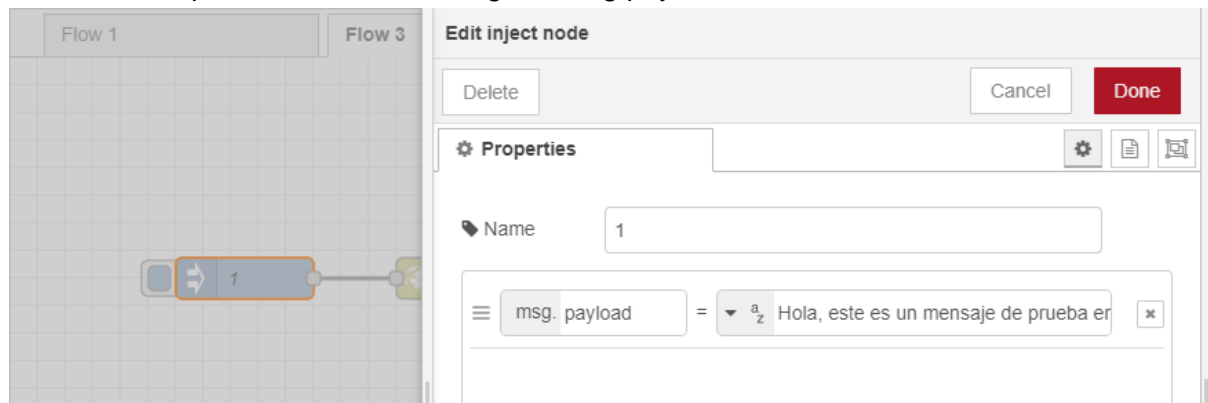


Sprint #2. Crear un hola mundo desde Node-RED a una WebApp

Para crea un “hola mundo desde node-RED” hacia una WebApp se deben tener en cuenta algunas consideraciones, sin embargo, se puede ver como el proceso inverso del sprint1. Para este caso deben configurarse los bloques de node-RED para que envié datos a la WebApp, se puede dejar el mismo diagrama de bloques usado para el sprint1, solamente que

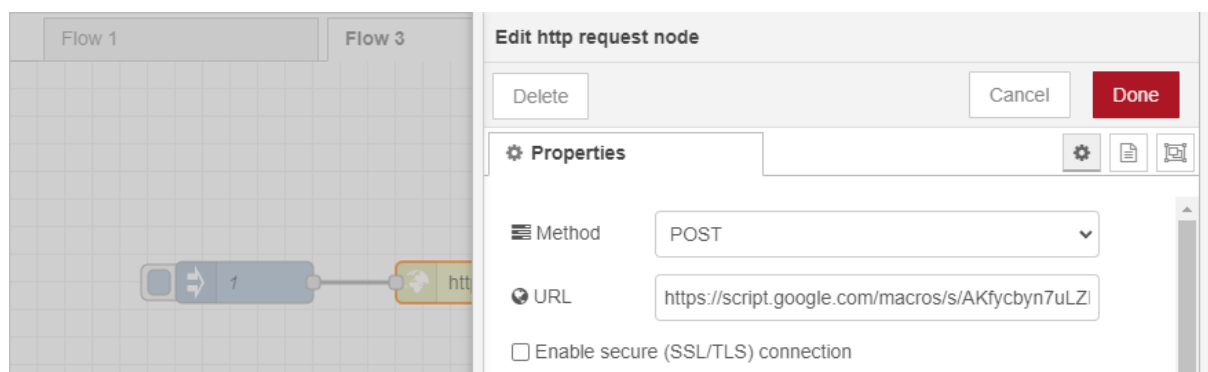
realizando algunas configuraciones para que se envíen datos en lugar de recibirlos, algunas de las configuraciones son las siguientes:

Bloque Inject: Se debe escribir el mensaje que se desea enviar, tal como se muestra a continuación, para ello se debe configurar msg.payload



Bloque http request: Este bloque es fundamental, ya que aquí debe ponerse la URL del webapp el cual va a recibir los mensajes que se envían desde Node-RED, para este caso el webapp se encuentra en el siguiente link:

https://script.google.com/macros/s/AKfycbyn7uLZHT_0XuLiDoj3vHeCQdA6bHgG9Gom8X1NI3ys4I9nWJXqP6A86ZtJJrUsCw00g/exec



Los bloques JSON y salida1 deben dejarse configurados como ya lo estaban en el sprint1, con estas configuraciones ya debería verse el mensaje que se quiere enviar cuando se oprime el botón Deploy, tal como se muestra a continuación:

```
3/11/2024, 8:16:47 p. m. node: salida1
msg.payload : string[3010]

"<!DOCTYPE html><html lang="en"><head><meta name="description" content="Web word processing, presentations and spreadsheets">
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=0">
<link rel="shortcut icon" href="//docs.google.com/favicon.ico"><title>Page Not Found</title><meta name="referrer"
content="origin"><link href="//fonts.googleapis.com/css?family=Product+Sans" rel="stylesheet" type="text/css"
nonce="mmvbvodiZUraKpvaKN8wF0w"><style nonce="mmvbvodiZUraKpvaKN8wF0w">.goog-inline-block{position:relative;display:-moz-inline-
box;display:inline-block}* html .goog-inline-block{display:inline}*:first-child+html .goog-inline-block{display:inline}#drive-
logo{margin:18px 0;position:absolute;white-space:nowrap}.docs-drive-logo-img{background-
image:url(//ssl.gstatic.com/images/branding/googlelogo/1x/googlelogo_color_116x41dp.png);-webkit-background-size:116px
41px;background-size:116px 41px;display:inline-block;height:41px;vert..."
```

Sin embargo este mensaje aparece debido a que existen errores que ya no dependen de la configuración de nodered sino de google app script, simplemente no permite la comunicación ni envío del mensaje, a pesar de verificar en múltiples ocasiones que todo estuviese en orden

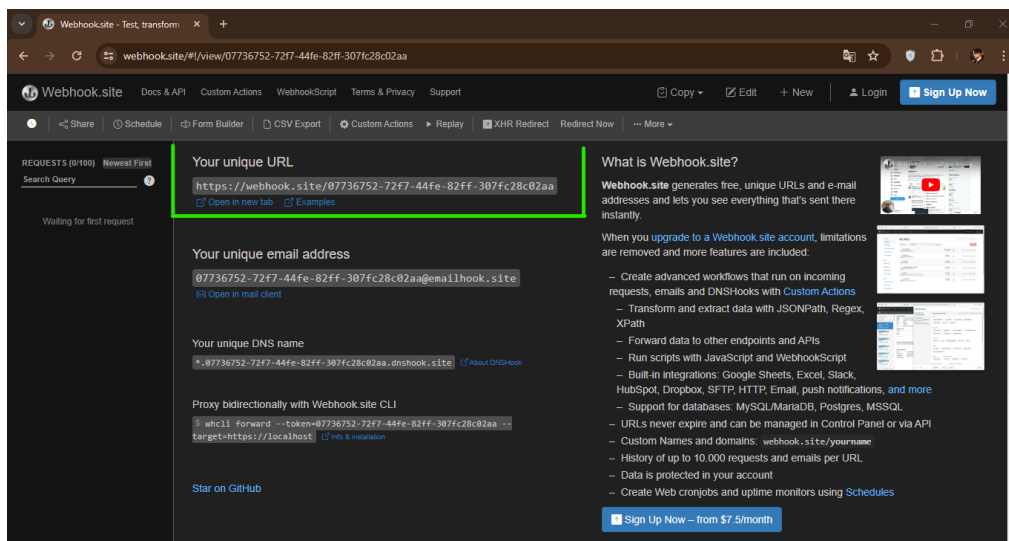
y funcionando correctamente, para evidenciar si el problema se encuentra en NodeRED o en Google App Script, podemos hacer uso de webhook, antes de seguir, vamos a ver un poco acerca de esta plataforma.

¿Qué es un Webhook?

Un **webhook** es una manera de permitir que una aplicación envíe información a otra aplicación en tiempo real, utilizando el protocolo HTTP. Los webhooks funcionan como "callbacks" o "event listeners", donde un sistema envía datos a una URL específica en respuesta a un evento (como una actualización o un cambio en los datos). En otras palabras, es un mecanismo que permite que una aplicación notifique automáticamente a otra cuando algo sucede.

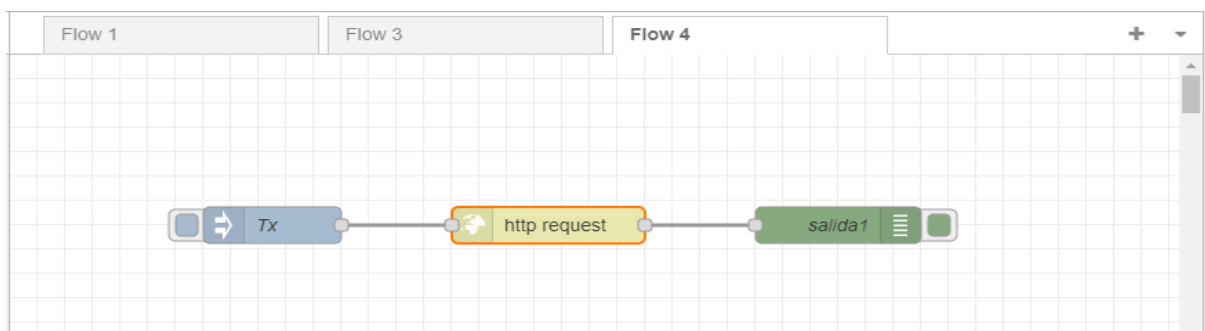
En lugar de que una aplicación tenga que consultar constantemente a otra para ver si hay actualizaciones, un webhook permite que la aplicación emisora envíe los datos de forma inmediata cuando ocurre un evento. ¿Pero cómo funciona para este caso?

Accedemos a webhook.site/#/, donde la pantalla de inicio debe verse así:

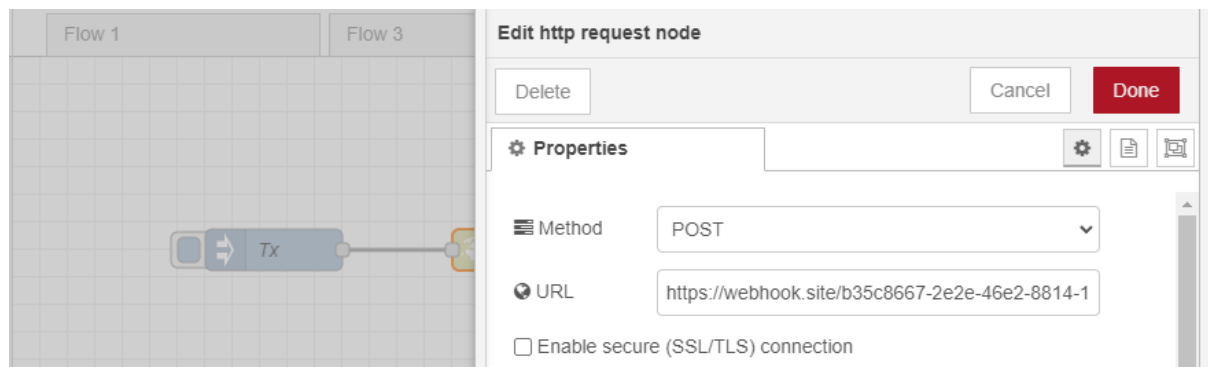


Automáticamente se generará una URL única en la página. Esta URL es un webhook temporal que puedes utilizar para recibir solicitudes HTTP.

La configuración de los bloques para usar weebhook es prácticamente la misma, simplemente se hizo cambio a un mensaje mucho mas sencillo como un “Hola mundo”, se cambió en http request la URL de GAS por la de webhook y adicional a esto se eliminó el bloque de JSON, tal como se muestra a continuación:



Reemplazo de la URL de Google App Script por la webhook única que se nos asigna:



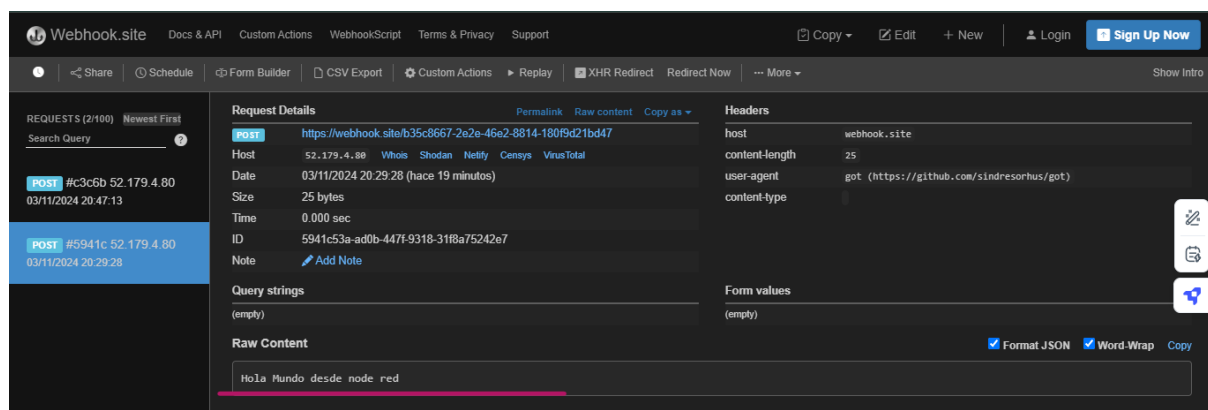
Una vez realizado todo este proceso se oprime el botón de deploy y se puede evidenciar en la salida del mismo node-RED el siguiente mensaje:

```
3/11/2024, 8:29:28 p. m. node: salida1
msg.payload : string[145]

"This URL has no default content configured. <a
href="https://webhook.site/#!/view/b35c8667-2e2e-
46e2-8814-180f9d21bd47">View in Webhook.site</a>."
```

Sin embargo, es importante tener en cuenta que es una respuesta estándar de Webhook.site y no es un error. Webhook.site está simplemente indicando que no tiene una respuesta configurada para devolver cuando recibe una solicitud POST. Esto es porque Webhook.site está diseñado para recibir y mostrar datos, pero no para responder con un mensaje personalizado como lo haría un servidor configurado específicamente para ese propósito. Pero una vez vayamos nuevamente a webhook evidenciamos que el mensaje enviado ya se encuentra allí, lo que confirma que **Node-RED está funcionando correctamente** y envía la solicitud POST de manera adecuada, ya que los datos enviados ("Hola Mundo desde Node-RED") llegaron exitosamente.

Esto indica que el problema no está en Node-RED, sino en algún aspecto de la **WebApp de Google Apps Script**

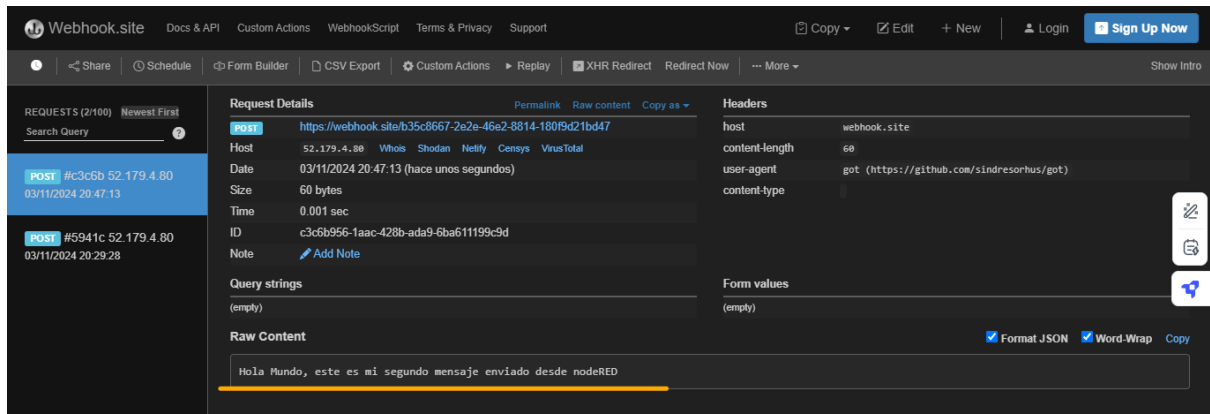


Cada vez que se cambia el mensaje o se envía uno diferente, este responde de forma correcta, pues también realiza el cambio en la nueva interfaz, por ejemplo, el segundo

mensaje enviado fue: “Hola mundo, este es mi segundo mensaje enviado desde nodeRED”, tal como se muestra a continuación:

```
3/11/2024, 8:47:13 p. m. node: salida1
msg.payload : string[145]

"This URL has no default content configured. <a
href="https://webhook.site/#!/view/b35c8667-2e2e-
46e2-8814-180f9d21bd47">View in Webhook.site</a>."
```



¿Por qué fue bueno usar Webhook site como prueba?

En este caso, utilizar **Webhook.site** fue una excelente herramienta de prueba por varias razones:

1. **Verificación de la Solicitud de Node-RED:** Nos permitió confirmar si **Node-RED** estaba configurado correctamente y si podía enviar datos a una URL externa. Al ver que la solicitud llegaba a Webhook.site con los datos correctos, comprobamos que el problema no estaba en Node-RED.
2. **Diagnóstico del Problema:** Al eliminar Google Apps Script de la ecuación y usar Webhook.site, pudimos confirmar que el flujo de datos de Node-RED era funcional. Esto sugirió que el problema era específico de Google Apps Script o de la configuración de permisos de la WebApp, no de la configuración de Node-RED ni de la red.
3. **Rapidez y Simplicidad:** Webhook.site proporciona una URL temporal de prueba de manera inmediata y muestra el contenido de cualquier solicitud recibida, incluyendo detalles de los encabezados y el contenido de los datos. Esto simplifica el proceso de prueba sin necesidad de crear un servidor o una aplicación adicional.
4. **Confirmación Visual:** Webhook.site nos permite ver visualmente que los datos enviados ("Hola Mundo desde Node-RED") llegaron con éxito, eliminando dudas sobre posibles errores de formato o en la configuración de la URL.

En resumen, usar **Webhook.site** nos permitió confirmar que Node-RED estaba enviando correctamente los datos y que el problema estaba en algún aspecto de Google Apps Script, ayudando a aislar y diagnosticar el problema de manera más precisa.