

Web app en soluciones IoT

Objetivo

Lograr que el estudiante no solo comprenda estos conceptos (web app) sino que lo haga en el contexto de IoT, comprobando el valor que tienen en las soluciones IoT o IIoT.
Conocimientos previos

Conocimientos previos

Modelo cliente servidor

El modelo cliente-servidor es una arquitectura de diseño de software que se utiliza en aplicaciones y sistemas distribuidos. Este modelo divide una aplicación en dos partes principales: el cliente y el servidor. Cada una de estas partes tiene roles y responsabilidades específicas en la comunicación y el procesamiento de datos. Aquí hay una explicación detallada de cómo funciona el modelo cliente-servidor:

Cliente:

El cliente es la parte de la aplicación que interactúa directamente con el usuario final. Su función principal es enviar solicitudes al servidor y recibir respuestas del servidor. Los clientes pueden ser aplicaciones de software, navegadores web, dispositivos móviles, o cualquier dispositivo o programa que solicite y consuma servicios del servidor. Los clientes generalmente proporcionan interfaces de usuario y funciones para que los usuarios interactúen con la aplicación. Cuando un usuario realiza una acción (como hacer clic en un enlace en un navegador web), el cliente envía una solicitud al servidor para obtener información o realizar una acción específica. Los clientes pueden funcionar de manera independiente y acceder a diferentes servidores según sea necesario.

Servidor:

El servidor es la parte de la aplicación que procesa las solicitudes del cliente y proporciona respuestas adecuadas. Su función principal es escuchar y responder a las solicitudes de los clientes. Los servidores suelen ejecutarse en hardware más potente y confiable que los clientes, ya que deben manejar múltiples solicitudes simultáneamente. Los servidores almacenan y gestionan los datos de la aplicación, realizan cálculos, aplican lógica empresarial y brindan servicios a los clientes.

Un servidor puede estar diseñado para manejar un tipo específico de solicitud, como un servidor web que proporciona páginas web a los navegadores, o un servidor de bases de datos que almacena y recupera datos.

El servidor procesa la solicitud del cliente y envía una respuesta, que generalmente incluye los datos solicitados o un resultado de la acción realizada.

Puede haber múltiples clientes que se conectan a un solo servidor, y el servidor puede manejar varias solicitudes al mismo tiempo utilizando hilos o procesos.

Web App

Una Web App es una aplicación en la nube que se ejecuta en un navegador web cuando se invoca mediante su URL. Consiste en dos componentes clave:

Front-End: Es la parte de la aplicación que se carga en el navegador del usuario en forma de código HTML y JavaScript. Front-End es la interfaz de usuario con la que interactúa el usuario y representa la parte visible de la aplicación.

Back-End: Es el código que respalda la Web App sin ser descargado al navegador del usuario. En cambio, se ejecuta en servidores en la nube. El Back-End realiza tareas como el procesamiento de datos, la gestión de la lógica de la aplicación y la comunicación con bases de datos u otros servicios. Es responsable de la funcionalidad invisible para el usuario pero crítica para el funcionamiento de la aplicación.

Las características distintivas de una Web App incluyen:

Está alojada en un servidor en la web y, por lo tanto, es accesible a través de una URL.

La parte visible de la Web App se presenta como páginas HTML con elementos de JavaScript, lo que constituye el Front-End.

El Back-End, en contraste, ejecuta código JavaScript en la nube y se encarga de procesos que no se ejecutan directamente en el navegador del usuario.

Esta definición captura adecuadamente la arquitectura de una Web

Instanciación de una Web App

El término "instanciación" se refiere al proceso en el cual una Web App, que se encuentra alojada en un servidor en la nube, se adapta para su uso por múltiples usuarios. Cuando un usuario invoca o accede a la Web App a través de su navegador web, se crea una versión única y específica de la aplicación para ese usuario en ese momento. Esto significa que:

Creación de una Versión Personalizada: Cada vez que un usuario accede a la Web App, se genera una versión personalizada de la aplicación para ese usuario en particular. Esta versión puede incluir datos específicos del usuario, configuraciones personalizadas o interacciones únicas.

Descenso del Front-End: El Front-End de la Web App, que es la parte visible y con la que el usuario interactúa, se descarga y se ejecuta directamente en el navegador web del usuario.

Esto permite que el usuario interactúe con la aplicación de manera ágil y receptiva, ya que los recursos se gestionan localmente en su dispositivo.

La instanciación es fundamental en la naturaleza de las aplicaciones web, ya que permite que múltiples usuarios accedan y utilicen la misma aplicación de manera independiente, al tiempo que mantienen una experiencia personalizada y exclusiva.

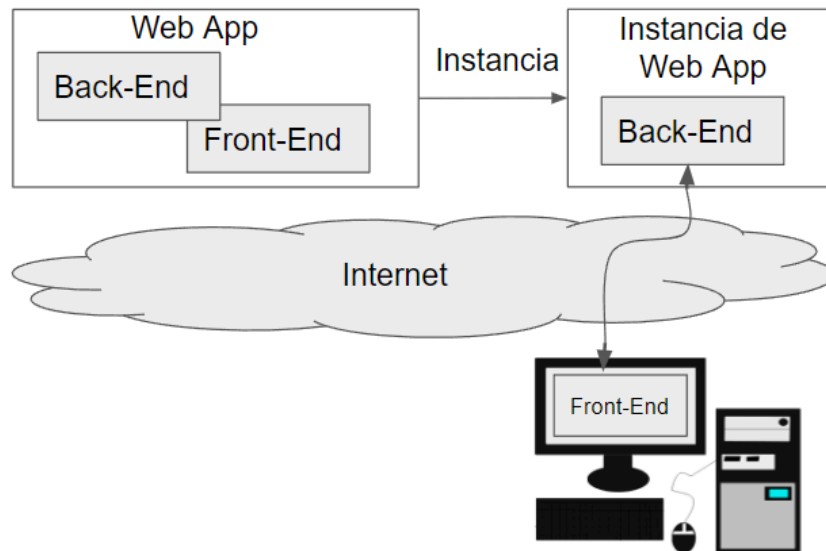


Fig. 1. la instanciación de una Web App

Las Web App en Google Apps Script

Las Web Apps de Google Apps Script se distinguen porque:

- El Front-End consiste en una combinación de HTML, CSS y JavaScript. Es posible que otros lenguajes o tecnologías puedan ser involucrados pero con intermediación de JavaScript.
- HTML (HyperText Markup Language): HTML se utiliza para estructurar el contenido de una página web, definir los elementos y etiquetas que componen la interfaz de usuario y establecer la semántica de la página.
- CSS (Cascading Style Sheets): CSS se utiliza para dar estilo y diseño a los elementos HTML. Controla aspectos como la presentación, el diseño, los colores, las fuentes y la disposición de los elementos en la página.
- JavaScript: JavaScript es un lenguaje de programación que se utiliza para agregar interactividad y funcionalidad dinámica a una página web. Puede manipular el contenido de la página, responder a eventos del usuario, realizar solicitudes a servidores, entre otras tareas.
- Además de HTML, CSS y JavaScript, el Front-End puede incluir otros marcos y bibliotecas que se utilizan para desarrollar interfaces de usuario más complejas. Por ejemplo, bibliotecas como React, Angular o Vue.js son populares para crear aplicaciones web de una sola página (SPA) altamente interactivas.

Las partes de una URL para HTML, Web Apps, API

Primer caso. La siguiente es una URL típica de una Web App

<https://script.google.com/macros/s/AKfycbxUoNqtBZoF2t40UmYbQGjLxbvgKAoeLuglayxlhpYwJ5i4QYPivmV8Z8iCJrGvOZwZpg/exec?a=4&b=7>

Vemos las siguientes partes:

- https://: indica que se trata de el protocolo https que es un http seguro
- script.google.com: se trata del dominio del servidor que aloja la Web App. Está claro que este es el caso para las WebApps creadas con Google Apps Script.
- /macros/s/: Representa la ruta destinada por Google Apps Script para almacenar las WebApps y es que anteriormente, a las WebApps Google las llamaba macros.
- AKfycbxUoNqtBZoF2t40UmYbQGjLxbvgKAoeLuglayxlhpYwJ5i4QYPivmV8Z8iCJrGvOZwZpg: Esas letras cambian para cada WebApp y se trata del ID que representa de manera única a la WebApp. Esto es así al menos en el caso de WebApps creadas con google Apps Script
- exec: refleja que se trata de un código ejecutable, que se ejecuta en la nube.
- ?a=4&b=7: Lo que está después de "?" Es lo que se conoce como una "query string" o "palabra de consulta". Antes de continuar, debemos recordar que cuando se escribe una URL en un navegador y se oprime la tecla "Enter", el navegador envía una solicitud de tipo GET a esa dirección. Pues bien, la "query string" son los parámetros que se le trasladan a la función doGet() que recibe esa solicitud, si existe. En otras palabras, la query string es la parte de la dirección web usada para transmitir datos de manera estructurada a un servidor, que puede ser una WebApp o una API. Los componentes básicos de una query string son usualmente unos pares "nombre=valor" separados por el símbolo "&". Pero no siempre es así, en algunos casos, los parámetros pueden ser más complejos y estructurados, utilizando un formato como JSON o XML para representar datos más elaborados.

Un ejemplo simple de una query string:
?nombre=Juan&edad=30&ciudad=NuevaYork
La URL junto con la query string
https://www.ejemplo.com/pagina?nombre=Juan&edad=30&ciudad=NuevaYork

Ejemplo de una query string con formato JSON
?parametros={"filtro":"precio","orden":"ascendente","pagina":1}
La URL junto con la query string
https://www.ejemplo.com/buscar?parametros={"filtro":"precio","orden":"ascendente","pagina":1}

Segundo caso:

https://script.googleusercontent.com/a/macros/e3t.uis.edu.co/echo?user_content_key=17RvE3DiQrj1BzPOmOyEXLtzXrH8162wiNd9LxfppmVf5gxWkXghu_e-xsyNtxu8NJ4z8TQk2fphHxTIUf-P4j1ELbuHbb01m5_BxDIH2jW0nuo2oDemN9CCS2h10ox_nRPgeZU6HP-C-Fz6y7-u6bviHnw2xp1zS4chwnYR2uFmAjcECZr3w9vdETDh7XoiuuHrQycSe_bMjEMqqOTPe_oy3TtfWa22RUZy-Vz8oVE9RIQYNCkw&lib=MGVLvEMMvYmPMqHY6cMCf-7IZxIdTy3z6

Vemos que:

- La “query string” está dada por”: ?user_content_key=código. Cuando se invoca una WebApp desde el navegador, Google la redirige por cuestiones internas y la URL de destino adopta una forma similar a esta.

Tercer caso (la URL de API):

https://v6.exchangerate-api.com/v6/API_KEY

Donde API_KEY es un código que cada usuario tiene para acceder a unos recursos puntuales de una API. Como no hay un “?” antes de ese código no sigue la estructura de una “query string” pero si pasa como un parámetro a una función doGet() o a lo que la API tenga previsto. Por eso, la URL junto con la API_key se conoce como End-Point para esos recursos específicos.

Método que el profesor **Homero Ortega Boada** propone para crear WebApps

El profesor ha escrito una metodología más detallada para crear WebApps, la cual se detalla en el siguiente: [enlace](#)

Reto general para esta práctica

Implementar una WebApp como solución a la oficina de despacho, que requiere consultar datos de pedidos que se están registrando en una google sheet. Esto como alternativa a las soluciones que se implementaron en prácticas anteriores. Para las personas que no ha seguido las guías anteriores vamos a plantear el problema de manera más concreta así:

- Se tiene un archivo de GoogleSheet que contiene datos que la oficina de despachos quiere ver en una página web. La Google sheet tiene:
 - una URL. Si no cuentas con esa Google Sheet, simplemente crea una y obtén su URL
 - tiene una hoja con el nombre que puede ser “**RegistrosIoT**”
 - Las columnas están marcadas como en la siguiente figura y tiene datos cambiantes. Nota: para las pruebas preliminares nos da lo mismo que los datos sean cambiantes o no.

	A	B	C	D	E	F
1	numero	fecha	nevera	producto	cantidad	pesos
2	1	2023-09-14 11:12:53	neveraChachon	huevos	10	30000
3	2	2023-09-14 11:13:06	neveraChachon	huevos	10	30000
4	3	2023-09-14 11:16:29	neveraChachon	huevos	10	30000
5	4	2023-09-16 21:25:24	neveraChachon	huevos	10	30000

+ ≡ RegistrosIoT ▾

- La WebApp tendrá una función capaz de leer los datos de esa hoja
- La WebApp incluirá una página html en la que habrá una tabla esperando esos datos
- Vamos a suponer que la WebApp es capaz de presentar la información de diversas maneras y que la entregará en forma de tabla solo si recibe una orden, por una query string que traiga el par “tipodatos=tabla”, de lo contrario entregará los datos en forma de texto.

Sprint 1. Crear la solución más simple posible basada en la query string a usar

Se debe crear una WebApp que lo único que contiene es una función doGet(e). Una vez implementada, obtendremos la URL y la probaremos desde un navegador. Le enviaremos la “query string” con el par “tipodatos=tabla”, así como otros valores con el fin de comprobar cómo esa información llega a la función doGet(e)

Paso 1. Pruebas de envío y recepción de un “query string”

Crea un proyecto de Google Apps Script que solo contiene un archivo .gs y escribe el código siguiente:

```
function doGet(e) {
  console.log(e);
}
```

No podrás probarlo directamente porque está pidiendo un parámetro “e” que debe llegar desde el exterior. Implementa el proyecto como una WebApp de pruebas así: Menu superior derecho > Implementar > Implementaciones de prueba > copia la URL

Nota 1: Curiosamente los pasos anteriores para crear una implementación de pruebas funcionó bien en unos casos, pero cuando se quiso hacer lo mismo en la red de la UIS, no entregaba la URL sino la ID de la implementación. No profundizamos mucho en eso, la solución que seguimos fué: comentar haciendo una primera implementación normal, luego sí permitió crear una implementación de pruebas y obtener la URL

Nota 2: Ten en cuenta que las implementaciones de prueba funcionan solo cuando quien la usa es el mismo desarrollador. Tiene la ventaja que no necesitas estar haciendo una nueva implementación cuando realizas un cambio.

URL de tu WebApp

https://script.google.com/macros/s/AKfycbwavBM_gcZHYWMchGTM7cyeaNNKhA5zhiTOUrPzbN8/dev

LLama la WebApp y pásala una “query string”, por ejemplo “nombre=Juan&edad=30&ciudad=NuevaYork”

Nota recuerda que en el navegador desde escribir:
la URL de tu WebApp seguida de ?nombre=Juan&edad=30&ciudad=NuevaYork

Nota: No te preocupes si el navegador muestra error. Esto es debido a que el navegador espera una respuesta, pero nuestro código es tan simple que no le retorna ninguna respuesta por ahora.

Escribe el valor de “e” recibido por doGet(e)

Nota: En tu navegador en modulo consola no podrás ver el resultado del comando console.log(e) porque este está a nivel de back-end. La única manera es consultarlo en el IDE de Google Apps Script así: Menu lateral izquierdo > Ejecuciones > abres la última ejecución.

?nombre=Juan&edad=30&ciudad=NuevaYork

Principal	doGet	Aplicación web	26 ago 2024, 16:43:57	0.482 s	Completada	⋮ ^
Registros de Cloud						
26 ago 2024, 16:43:57	Depuración	{ queryString: 'nombre=Juan&edad=30&ciudad=NuevaYork', contextPath: '', parameter: { nombre: 'Juan', edad: '30', ciudad: 'NuevaYork' }, parameters: { ciudad: ['NuevaYork'], nombre: ['Juan'], edad: ['30'] }, contentLength: -1 }				
						ACTUALIZAR

Cuando se ejecuta la función directamente en GAS

Principal	doGet	Editor	26 ago 2024, 16:44:25	0.365 s	Completada	⋮ ^
Registros de Cloud						
26 ago 2024, 16:44:26	Depuración	undefined				
						ACTUALIZAR

?nombre=Diego&edad=23&ciudad=Bucaramanga

Principal	doGet	Aplicación web	26 ago 2024, 16:47:06	0.4 s	Completada	⋮ ^
Registros de Cloud						
26 ago 2024, 16:47:07	Depuración	{ contentLength: -1, parameters: { ciudad: ['Bucaramanga'], edad: ['23'], nombre: ['Diego'] }, contextPath: '', parameter: { ciudad: 'Bucaramanga', edad: '23', nombre: 'Diego' }, queryString: 'nombre=Diego&edad=23&ciudad=Bucaramanga' }				
						ACTUALIZAR

Análisis: observa que los datos han llegado a “e” en forma de un objeto de datos y que hay más elementos que los enviados por el navegador.

Paso 2. Interpretación de los datos enviados desde el navegador

Ingenia la manera de obtener los datos puros enviados desde el navegador.

Nota: Sólo preocúpate por obtener un objeto de datos que contenga la información enviada por el navegador. Por ejemplo, si el navegador envía “nombre=Juan&edad=30&ciudad=NuevaYork” el objeto de datos sería:

```
{  
  ciudad: 'NuevaYork',  
  nombre: 'Juan',  
  edad: '30'  
}
```

Escribe tu código nuevo para la función doGet(e) para obtener los datos puros

```
function doGet(e) {  
  // Crear un objeto de datos vacío para almacenar los valores  
  let datosPuros = {};  
  
  // Verificar si hay parámetros en la solicitud  
  if (e.parameters) {  
    // Recorrer los parámetros y asignarlos al objeto datosPuros  
    for (let key in e.parameters) {  
      // Obtener el valor del parámetro  
      let valor = e.parameters[key];  
  
      // Si el valor es un array, convertirlo a string (manejo básico)  
      datosPuros[key] = Array.isArray(valor) ? valor.join(", ") : valor;  
    }  
  }  
  
  // Log para depuración (esto aparecerá en el registro de ejecución)  
  console.log(datosPuros);  
  
  // Devolver la respuesta como JSON (opcional)  
  return ContentService.createTextOutput(JSON.stringify(datosPuros))  
    .setMimeType(ContentService.MimeType.JSON);  
}
```

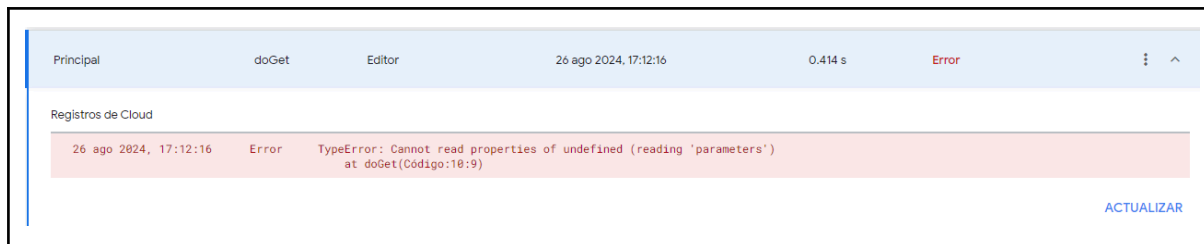
Pega aquí los datos obtenidos

Principal	doGet	Aplicación web	26 ago 2024, 17:11:32	0.491 s	Completada	⋮ ^
Registros de Cloud						
26 ago 2024, 17:11:33 Depuración { nombre: 'Juan', ciudad: 'NuevaYork', edad: '30' }						
						ACTUALIZAR

Modificando la query string:

Principal	doGet	Aplicación web	26 ago 2024, 17:11:55	0.599 s	Completada	⋮ ^
Registros de Cloud						
26 ago 2024, 17:11:55 Depuración { nombre: 'Diego', edad: '23', ciudad: 'Bucaramanga' }						
						ACTUALIZAR

Ejecutando en GAS:



Sprint 2. Retornar al navegador una respuesta que depende de lo que demande la “query string”

La idea es que si la “query string” es la siguiente “tipodatos=tabla” en el navegador aparecerá una página html diciendo algo. De no ser así, en el navegador aparecerá el contenido de “e” en formato JSON.

Agrega al proyecto un archivo html que se llame “mainTabla.html”. Agrégale el siguiente código

```
<!DOCTYPE html>
<html>
<head>
  <title>Servicios a la oficina de reparto</title>
</head>
<body>
  <h1>Órdenes recibidas para el despacho</h1>

  <table border="1">
    <tr>
      <th>numero</th>
      <th>fecha</th>
      <th>nevera</th>
      <th>producto</th>
      <th>cantidad</th>
      <th>pesos</th>
    </tr>
    <tr>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
    </tr>
    <tr>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
    </tr>
  </table>
</body>
</html>
```

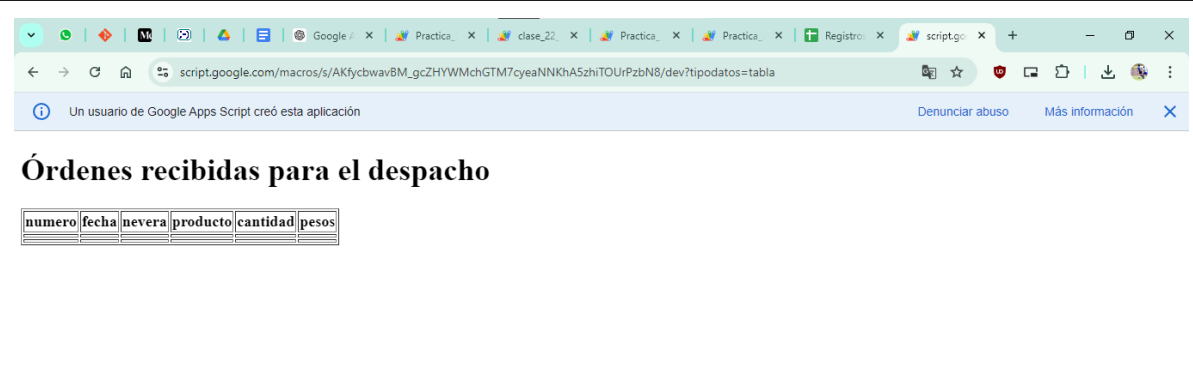
En el archivo Codigo.gs usa el siguiente código

```
function doGet(e) {
  var tipoDatos = e.parameter.tipodatos;
  if (tipoDatos == 'tabla') {
    var htmlParaElCliente = HtmlService.createHtmlOutputFromFile('mainTabla.html');
    return htmlParaElCliente;
  }
  else {
    var salidaEntexto = ContentService.createTextOutput("Acabo de recibirtelo siguiente y te lo reenvío traducido a JSON: "
+ JSON.stringify(e));
    salidaEntexto.setContentType(ContentService.MimeType.JSON);
    return salidaEntexto;
  }
}
```

Convierte el proyecto en una WebApp de pruebas

Realízale pruebas a la Web App llamándola desde un navegador y pasándole la “query string” “tipodatos=tabla”

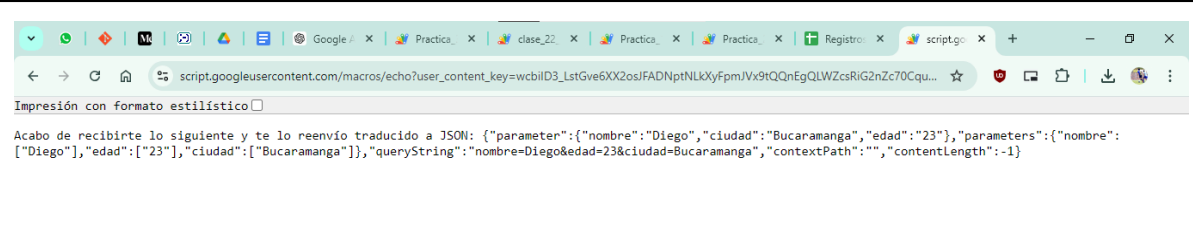
Resultado en el navegador



repite la prueba con una “query string” diferente. Escribe tu “query string”

"nombre=Diego&edad=23&ciudad=Bucaramanga"

Resultado en el navegador



Análisis del código. Explica cómo funciona el código para que en un caso retorne al navegador una página html, en otro caso que retorne un texto. Explique también por qué se usa JSON.stringify(e)

En Google Apps Script, cuando se recibe una solicitud GET (como al visitar una URL), el script puede manejar la solicitud en la función doGet(e). La función doGet(e) es responsable de generar y devolver el contenido adecuado al navegador según los parámetros recibidos en la solicitud.

1. Retorno de Texto en el Navegador (URL 1)

Cuando visitas una URL y obtienes un texto como respuesta, significa que el script está generando y devolviendo un contenido en formato de texto plano. Aquí se utiliza `ContentService.createTextOutput()` para crear una respuesta de texto:

```
function doGet(e) {  
  return ContentService.createTextOutput("Este es un texto de ejemplo");  
}
```

`ContentService.createTextOutput()`: Esta función devuelve un objeto `TextOutput`, que el navegador interpretará como texto plano. Este es el motivo por el que ves un texto directamente en la página web.

2. Retorno de HTML en el Navegador (URL 2)

En el segundo caso, donde la URL devuelve una página HTML, el script está generando una respuesta HTML usando `HtmlService`:

- **`HtmlService.createHtmlOutputFromFile()`**: Esta función devuelve un objeto `HtmlOutput`, que el navegador interpretará como una página HTML. La función carga un archivo HTML que está en el proyecto de Google Apps Script (en este caso, probablemente un archivo llamado `tabla.html`).
- **Condición en `doGet()`**: La función está verificando el valor del parámetro `tipodatos` en la query string. Si `tipodatos` es igual a `"tabla"`, se genera la página HTML con la tabla; de lo contrario, puede devolver un texto o manejar la solicitud de otra manera.

¿Por qué se usa `JSON.stringify(e)`?

`JSON.stringify(e)` se utiliza para convertir el objeto `e` en una cadena de texto en formato JSON. Esto es útil para depuración o para ver exactamente qué datos están siendo enviados al servidor en la solicitud. El objeto `e` contiene información importante como:

- **`e.parameters`**: Los parámetros de la solicitud como pares clave-valor (por ejemplo, `tipodatos=tabla`).
- **`e.queryString`**: La cadena completa de la query (por ejemplo, `tipodatos=tabla`).
- **`e.pathInfo`**: La parte restante del camino en la URL (si existe).

Cuando conviertes `e` en una cadena con `JSON.stringify()`, puedes ver todo esto como una cadena en formato JSON, lo que facilita la inspección de los datos de la solicitud.

Sprint 3. Llenar la tabla en el html con datos ficticios

Vamos a suponer que tenemos los siguientes datos en un array 2D para llenar la tabla datos:

```
[[9, 2023-09-17 9:36:09, "neveraChachones", "huevos", 10, 30000],[10, 2023-09-17 9:58:00, "neveraMax", "huevos", 8, 10000]]
```

Paso 1

Agrega al archivo Codigo.gs la función que simula los datos

```
function manejoDatos() {  
  var datos = [[9, '2023-09-17 9:36:09', 'neveraChachones', 'huevos', 10, 30000],  
    [10, '2023-09-17 9:58:00', 'neveraMax', 'huevos', 8, 10000]];  
  Logger.log(datos);  
  return datos;  
}
```

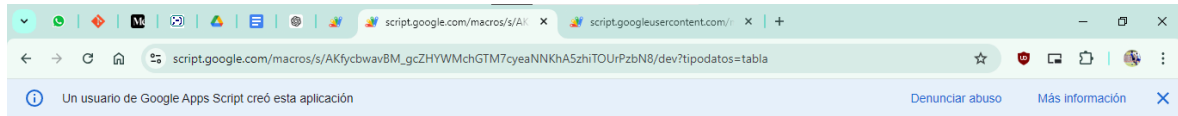
Cambia el código html por una versión que contenga un script capaz de llenar la tabla con los datos

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Servicios a la oficina de reparto</title>  
</head>  
<body>  
  <h1>Órdenes recibidas para el despacho</h1>  
  
  <table border="1" id="tablaDatos">  
    <tr>  
      <th>numero</th>  
      <th>fecha</th>  
      <th>nevera</th>  
      <th>producto</th>  
      <th>cantidad</th>  
      <th>pesos</th>  
    </tr>  
  </table>  
  
  <script>  
    // Función para llenar la tabla con datos  
    function llenarTabla(datos) {  
      var tabla = document.getElementById("tablaDatos");  
      datos.forEach(function(fila) {  
        var row = tabla.insertRow(-1);  
        fila.forEach(function(valor) {  
          var cell = row.insertCell();  
          cell.innerHTML = valor;  
        });  
      });  
    }  
  
    // Llama a la función para llenar la tabla cuando la página se carga  
    google.script.run.withSuccessHandler(llenarTabla).manejoDatos();  
  </script>  
</body>  
</html>
```

Captura pantalla del resultado que ver en el navegador

Usando la URL:

https://script.google.com/macros/s/AKfycbwavBM_gcZHYWMchGTM7cyeaNNKhA5zhiTOUrPzbN8/dev?tipodatos=tabla



Órdenes recibidas para el despacho

numero	fecha	nevera	producto	cantidad	pesos
9	2023-09-17 9:36:09	neveraChachones	huevos	10	30000
10	2023-09-17 9:58:00	neveraMax	huevos	8	10000

Análisis

- Explica de qué manera logra el script obtener los datos para llenar la tabla
- investiga a fondo como funciona el comando `“google.script.run.withSuccessHandler() ...”`

1. ¿Cómo obtiene el script los datos para llenar la tabla?

El script en el archivo HTML se comunica con el servidor Google Apps Script para obtener los datos mediante

```
google.script.run.withSuccessHandler(llenarTabla).manejoDatos();
```

A continuación, se describe el flujo de cómo el script obtiene los datos:

1. **Comando `google.script.run.withSuccessHandler()`:** Este comando permite que el cliente (el navegador) ejecute una función del servidor de Google Apps Script. En este caso, la función llamada es `manejoDatos()`.
2. **Llamada a la función `manejoDatos()` en el servidor:** Cuando el navegador ejecuta `google.script.run.manejoDatos()`, el servidor de Google Apps Script responde ejecutando la función `manejoDatos()`. Esta función simplemente devuelve un arreglo bidimensional que contiene los datos que se utilizarán para llenar la tabla HTML.
La función `manejoDatos()` devuelve un arreglo de arreglos (matriz) que contiene las filas y columnas que llenarán la tabla.
3. **Uso del `withSuccessHandler()`:** El `withSuccessHandler()` permite definir una función de manejo de éxito en el lado del cliente. Esta función se ejecutará cuando el servidor de Google Apps Script haya terminado de ejecutar la función `manejoDatos()` y haya devuelto los datos. En este caso, la función que maneja los datos es `llenarTabla()`.
4. **Llenado de la tabla HTML:** La función `llenarTabla(datos)` toma los datos devueltos por `manejoDatos()` y los inserta en la tabla HTML. Cada fila de datos se convierte en una nueva fila en la tabla, y cada elemento de la fila se convierte en una celda dentro de esa fila.

2. ¿Cómo funciona `google.script.run.withSuccessHandler()`?

`google.script.run.withSuccessHandler()` es una función que forma parte de la biblioteca `google.script.run`, la cual es una API específica de Google Apps Script utilizada para invocar funciones en el lado del servidor desde el lado del cliente

(HTML/JavaScript en el navegador). Este enfoque es útil cuando necesitas realizar tareas asíncronas o ejecutar funciones que requieren acceso a los servicios de Google.

Detalles de su funcionamiento:

1. **google.script.run**: Es el punto de entrada para llamar funciones del lado del servidor desde la interfaz del cliente. Este objeto permite ejecutar funciones de servidor de Apps Script directamente desde un script en una página HTML.
2. **withSuccessHandler()**: Esta es una función encadenada que se utiliza para definir una función de éxito, la cual se ejecutará cuando la función del servidor haya terminado exitosamente y haya devuelto un resultado. El resultado devuelto por la función del servidor se pasa como argumento a la función manejadora del éxito.

```
google.script.run.withSuccessHandler(llenarTabla).manejoDatos();
```

funcionManejadora: Es la función del cliente (HTML/JavaScript) que manejará el resultado devuelto por la función del servidor.

nombreFuncionServidor: Es el nombre de la función que se está ejecutando en el servidor de Google Apps Script.

3. **Función de manejo de éxito**: La función definida en **withSuccessHandler()** recibe el valor devuelto por la función del servidor como su argumento. Este valor puede ser un string, número, objeto o cualquier dato que haya sido devuelto.
4. **Proceso Asíncrono**: La ejecución es asíncrona, lo que significa que el navegador no se bloquea mientras espera la respuesta del servidor. Una vez que la función en el servidor completa su ejecución, el navegador ejecuta la función de éxito.

Paso 2

Mejorar la solución para que los datos se refresquen cada segundo.

Mejora el archivo html para que incluya el refresco de datos. El siguiente puede ser el código html

```
<!DOCTYPE html>
<html>
<head>
  <title>Servicios a la oficina de reparto</title>
</head>
<body>
  <h1>Órdenes recibidas para el despacho</h1>

  <table border="1" id="tablaDatos">
    <tr>
      <th>numero</th>
      <th>fecha</th>
      <th>nevera</th>
      <th>producto</th>
      <th>cantidad</th>
      <th>pesos</th>
    </tr>
  </table>

  <script>
    // Función para llenar la tabla con datos
    function llenarTabla(datos) {
      var tabla = document.getElementById("tablaDatos");
```

Resultado

https://script.google.com/macros/s/AKfycbwavBM_gcZHYWMchGTM7cyeaNNKhA5zhiTOUrPzbN8/dev?tipodatos=tabla

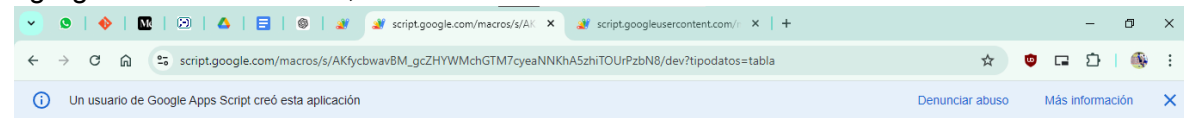
The screenshot shows the Google Apps Script editor interface. On the left, there's a sidebar with icons for information, code, main file (mainTabla.html), libraries, and services. The main area displays a JavaScript script with a function named `manejoDatos`. Below the script, the 'Registro de ejecución' (Execution Log) panel is open, showing three entries:

- 18:17:07 Aviso**: Se ha iniciado la ejecución
- 18:17:03 Información**: [[{"id": "9.0", "timestamp": "2023-09-17 9:36:09", "content": "neveraChachones", "category": "huevos", "count": 10}, {"id": "10.0", "timestamp": "2023-09-17 9:58:00", "content": "neveraMax", "category": "huevos", "count": 8}, {"id": "20.0", "timestamp": "2024-08-26 18:14:00", "content": "neveraDAGD", "category": "carne", "count": 2}, {"id": "10.0", "timestamp": "2024-08-26 18:14:00", "content": "neveraDAGD", "category": "queso", "count": 5}, {"id": "10.0", "timestamp": "2024-08-26 18:16:00", "content": "neveraDAGD", "category": "jamón", "count": 3}]]
- 18:17:07 Aviso**: Se ha completado la ejecución

At the bottom, a browser window shows the URL: `script.google.com/macros/s/AKfycbwavBM_gcZYHWMchGTM7rCyeaNNkHA5zhiTOUrPzbN8/dev/tipodatos=tabla`. A notification at the very bottom states: 'Un usuario de Google Apps Script creó esta aplicación'.

numero	fecha	nevera	producto	cantidad	pesos
9	2023-09-17 9:36:09	neveraChachones	huevos	10	30000
10	2023-09-17 9:58:00	neveraMax	huevos	8	10000
20	2024-08-26 18:14:00	neveraDAGD	carne	2	20000
10	2024-08-26 18:14:00	neveraDAGD	queso	5	25000
10	2024-08-26 18:16:00	neveraDAGD	jamón	3	35000

Agregando un nuevo dato, se observa:



Órdenes recibidas para el despacho

numero	fecha	nevera	producto	cantidad	pesos
9	2023-09-17 9:36:09	neveraChachones	huevos	10	30000
10	2023-09-17 9:58:00	neveraMax	huevos	8	10000
20	2024-08-26 18:14:00	neveraDAGD	carne	2	20000
10	2024-08-26 18:14:00	neveraDAGD	queso	5	25000
10	2024-08-26 18:16:00	neveraDAGD	jamón	3	35000
44	2024-08-26 18:18:00	neveraNueva	tomates	6	4500

Análisis. Explique cómo funciona el script en el html.

Este sistema implementa una página web que presenta una tabla dinámica con datos que se actualizan cada segundo desde el servidor de Google Apps Script. La función `doGet(e)` maneja las solicitudes HTTP GET, devolviendo un HTML o un texto JSON dependiendo del parámetro `tipodatos`. Los datos se recuperan del servidor mediante la función `manejoDatos()` y se insertan en la tabla utilizando `google.script.run.withSuccessHandler()` para manejar la llamada al servidor y actualizar la interfaz de usuario en el navegador.

El archivo `mainTabla.html` define la estructura de la página web que muestra una tabla con los datos recibidos desde el servidor de Google Apps Script.

JavaScript en el navegador

El código JavaScript en el navegador realiza las siguientes funciones:

- **Función `llenarTabla(datos)`:**
 - Esta función recibe los datos desde el servidor y los usa para llenar la tabla HTML.
 - Primero, elimina cualquier fila de datos antigua de la tabla para evitar la duplicación.
 - Luego, recorre los datos y agrega nuevas filas a la tabla, insertando cada valor en su correspondiente celda.
- **Función `actualizarDatos()`:**
 - Esta función se encarga de hacer una llamada al servidor de Google Apps Script para obtener los datos actuales.
 - Usa `google.script.run.withSuccessHandler(llenarTabla).manejoDatos()`; para ejecutar la función `manejoDatos()` en el servidor y, una vez que se obtienen los datos, se llama a la función `llenarTabla(datos)` para actualizar la tabla en el navegador.
- **Temporizador (`setInterval()`):**
 - Esta línea de código configura un temporizador que ejecuta la función `actualizarDatos()` cada segundo (1000 milisegundos). Esto garantiza que la tabla se actualice automáticamente cada segundo con los datos más recientes.

Sprint 4. Cambiar la simulación de los datos por la lectura de los datos en la base de datos en Google Sheet.

Hasta el momento, la función `manejoDatos()` ha estado simulando los datos para llenar la tabla. Necesitamos que eso cambie, que esa función los datos que se registran en la google sheet.

Escriba aquí el nuevo código para la función `manejoDatos()`

Nota: En este caso no usaremos la URL del archivo de google sheet sino su ID. La ID es una secuencia de letras que son parte de la URL y que es diferente para cada archivo, lo identifica de manera única. Puedes obtener la URL y de allí deducir la ID.

Ejemplo:

Si la URL es:

https://docs.google.com/spreadsheets/d/cBod4ZFa5ULbaDoQhMHad0vNu8r1YphS491SF19O8ngs/edit?usp=share_link

La ID es: cBod4ZFa5ULbaDoQhMHad0vNu8r1YphS491SF19O8ngs

```
const Libro_Id = "aqui va la ID del archivo de google sheet";
const HojaNombre = "Aquí va el nombre de la hoja de la google sheet que tiene los datos";

function manejoDatos() {

  // Definimos la fuente de la información desde el libro donde se
  // encuentra, la hora y el rango
  var libro = SpreadsheetApp.openById(Libro_Id);
  var hoja = libro.getSheetByName(HojaNombre);
  var ultimaFila = hoja.getLastRow();
  var rangoNombre = "A2:F" + ultimaFila;
  var rango = hoja.getRange(rangoNombre);

  // Leemos los datos en el rango de interés
  var datos = rango.getValues();

  // Lo siguiente es para corregir una falla que se presenta cuando se lee una
  // celda que contiene fecha ya que el formato que resulta no es compatible para
  // escribirlo en una tabla. Entonces buscamos la columna donde ocurre esto y la
  // convertimos a tipo string.
  datos.forEach(function (fila) {
    var fecha = new Date(fila[1]);
    fila[1] = fecha.toISOString();
  });

  console.log(datos);
  return datos;
}
```

Muestre un pantallazo del resultado obtenido en el navegador

script.google.com/macros/s/AKfycbwav8M_gcZHYWMchGTM7cyeaNNKhA5zhiTOUrPzbN8/dev?tipodatos=tabla

Un usuario de Google Apps Script creó esta aplicación

Denunciar abuso Más información

Órdenes recibidas para el despacho

numero	fecha	nevera	producto	cantidad	pesos
--------	-------	--------	----------	----------	-------

Agregando una variable de datos en la función manejoDatos() que se modifíco, se observa lo siguiente:

script.google.com/macros/s/AKfycbwav8M_gcZHYWMchGTM7cyeaNNKhA5zhiTOUrPzbN8/dev?tipodatos=tabla

Un usuario de Google Apps Script creó esta aplicación

Denunciar abuso Más información

Órdenes recibidas para el despacho

numero	fecha	nevera	producto	cantidad	pesos
1	2024-08-26T23:37:00.000Z	neveraDAGD	carne	2	20000
2	2024-08-26T23:37:00.000Z	neveraDAGD	queso	5	25000
3	2024-08-26T23:37:00.000Z	neveraDAGD	jamón	3	35000
4	2024-08-26T23:37:00.000Z	neveraNueva	tomates	6	4500

Sprint 5. Finalizar la solución

Paso 1. Es necesario implementar la WebApp, ya no como una versión de pruebas, sino para la producción.

Paso 2. Embeber la WebApp en un Google Sites

Paso 3. Comprobar que la solución funciona usando el publicador, el broker y el suscriptor

Muestre un pantallazo del resultado

docs.google.com/spreadsheet...

Registros_lot_2195533

Archivo Editar Ver Insertar Formato ...

100%

\$ % .00 123 Predet...

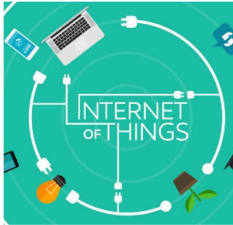
	A	B	C	D	E	F
	numero	fecha	nevera	producto	cantidad	pesos
1	1	26/08/2024	neveraMafe	huevos	30	15000
2	2	26/08/2024	neveraDAGD	carne	2	24000
3	3	26/08/2024	neveraAndres	pollo	3	18000
4	4	26/08/2024	neveraDiego	mora	3	6000
5	5					

Datos_1 Datos_2 Datos_3

Practica_3

PRÁCTICA 3 - WEB APP

DIEGO ANDRÉS GARCÍA DÍAZ - 2195533.
2024-2



Órdenes recibidas para el despacho

numero	fecha	nevera	producto	cantidad	pesos
1	2024-08-26T04:00:00.000Z	neveraMafe	huevos	30	15000
2	2024-08-26T04:00:00.000Z	neveraDAGD	carne	2	24000
3	2024-08-26T04:00:00.000Z	neveraAndres	pollo	3	18000
4	2024-08-26T04:00:00.000Z	neveraDiego	mora	3	6000

Se adjunta documento proporcionado por el profesor *Homero Ortega*
Roada para crear Web Apps

Link de Google Sites:

<https://sites.google.com/view/practica3-dagd-2195533/p%C3%A1gina-principal>

Preguntas de control

Crees que la "query string" "?tipodatos=tabla" ha sido introducida por el profesor para que el estudiante aprenda a usarla y ver su utilidad, pero que en realidad, para atender las necesidades del departamento de atención, el código podría haber sido escrito sin considerar una "query string"?

Sí, es muy probable que la "query string" `?tipodatos=tabla` haya sido introducida con fines educativos para que los estudiantes comprendan cómo se puede utilizar en aplicaciones web para enviar parámetros al servidor y manejar diferentes tipos de solicitudes. Esta técnica es comúnmente utilizada en desarrollo web para modificar el comportamiento de una aplicación en función de los parámetros recibidos en la URL.

¿Es necesaria para la funcionalidad actual?

- **No estrictamente necesaria:** Si el único objetivo es mostrar la tabla HTML cada vez que se accede a la Web App, el código podría haberse escrito sin considerar la query string. En ese caso, el servidor siempre devolvería el mismo HTML sin necesidad de parámetros adicionales.
- **Uso directo:** En una implementación más simple y directa, podrías prescindir del parámetro `tipodatos` y simplemente devolver el HTML de la tabla cada vez que se llama a `doGet()`.

```
function doGet(e) {
    var htmlParaElCliente =
    HtmlService.createHtmlOutputFromFile('mainTabla.html');
```

```
    return htmlParaElCliente;
}
```

Este código siempre devolvería el archivo HTML con la tabla, sin necesidad de interpretar una query string.

Si deseas que el resultado del comando `console.log()` sea visto en el navegador, en modo desarrollador, donde deberías escribir el comando `console.log()`?

Para que el resultado de `console.log()` sea visible en el navegador, en modo desarrollador, el comando debe ser escrito en el código JavaScript del archivo HTML o en un archivo JavaScript vinculado al HTML. Esto es porque el `console.log()` de JavaScript se ejecuta en el contexto del navegador y muestra la salida en la consola del navegador.

Pasos para ver el resultado de `console.log()` en el navegador:

1. Escribe el comando `console.log()` en el código JavaScript del archivo HTML:

- Esto debe estar en un bloque `<script>` en el archivo HTML o en un archivo JavaScript externo que se incluya en la página.
- Ejemplo:

2. Abre las herramientas de desarrollo en tu navegador:

En la mayoría de los navegadores, puedes abrir las herramientas de desarrollo presionando **F12** o haciendo clic derecho en la página y seleccionando "Inspeccionar" o "Inspect".

Luego, selecciona la pestaña "Consola" (Console).

3. Ve el resultado en la consola del navegador:

- El mensaje que hayas escrito en `console.log()` aparecerá en la consola.

```
<script>
console.log("Mensaje desde el navegador");
</script>
```

Si el comando `console.log()` se escribe en Google Apps Script (del lado del servidor), el resultado no será visible en la consola del navegador, sino en los registros del script en Google Apps Script. Para verlo en el navegador, debe estar en el código que se ejecuta en el cliente (HTML/JavaScript).

Bibliografía

- Una práctica antigua sobre WebApp. [Enlace a una guía útil](#)

