LABORATORIO 2: PROGRAMACIÓN EN PLATAFORMA ILYZAELLE

---

1. Introducción:

En este laboratorio se trabajará con la plataforma Ilyzaelle para la programación de dispositivos Arduino (Uno/MEGA) interactuando con el Gateway Ilyzaelle. A través de este entorno se simularán lecturas de sensores, se gestionarán variables globales y se visualizarán datos en tiempo real mediante el dashboard integrado en la plataforma.

---

2. Objetivos:

1. Conectar y configurar dispositivos Arduino en la plataforma Ilyzaelle.
2. Simular lecturas de sensores y gestionar variables globales (gVar).
3. Implementar una variable contador que registre cuántas veces la variable estado se pone en true.
4. Visualizar el comportamiento de la variable contador en un gráfico dentro del dashboard.

---

3. Requisitos

3.1 Requisitos Software:

1. ilyzaelleGatewayManager_0.1.bat

3.2 Requisitos Hardware:

1. 2x Arduino Uno/MEGA
2. 1x Computadora Windows 11

---

4. Marco Teorico:

4.1 Definiciones claves:

Gateway IoT: Dispositivo o software que actúa como intermediario entre los nodos finales (sensores y actuadores) y la nube, gestionando protocolos, seguridad y enrutamiento de datos para garantizar la comunicación eficiente en sistemas de Internet de las Cosas (Stallings, 2013).

Plataforma IoT: Entorno integrado de hardware, software y servicios en la nube que permite el registro, la gestión, el almacenamiento y el análisis de datos generados por dispositivos conectados, facilitando la creación y el despliegue de aplicaciones IoT (Greengard, 2015).

Protocolo de comunicación: Conjunto de reglas y convenciones que definen el formato, la secuencia y la semántica de los mensajes intercambiados entre dos o más entidades de red, asegurando interoperabilidad y fiabilidad en la transmisión de información. (Tanenbaum & Wetherall, 2011).

JavaScript: Lenguaje de programación interpretado, orientado a eventos y de un solo hilo, que se ejecuta en el motor de JavaScript del entorno (por ejemplo, en el navegador o en Node.js). A diferencia de Arduino, donde el programa se basa en un bucle infinito loop(), JavaScript emplea un event loop que gestiona una cola de eventos y callbacks, procesando cada uno de manera asíncrona según la disponibilidad de recursos y las operaciones de E/S (Flanagan, 2011).

Socket-IO: Biblioteca de JavaScript que facilita la creación de aplicaciones web en tiempo real mediante una comunicación bidireccional basada en eventos entre el cliente y el servidor. Utiliza WebSocket cuando está disponible y, de forma transparente, recurre a otras tecnologías como polling largo o Flash Sockets en navegadores que no lo soportan (Buna, 2012).

4.2 Preguntas claves:

¿En qué se diferencian setInterval y setTimeout al programar tareas en Node.js?
Mientras que setTimeout(función, retraso) ejecuta la función indicada una sola vez después del retraso especificado (en milisegundos), setInterval(función, intervalo) lanza la misma función de forma repetitiva cada vez que transcurre el intervalo definido. De este modo, setTimeout resulta ideal para demoras puntuales, por ejemplo, esperar a que un recurso externo responda, mientras que setInterval es la opción más adecuada para realizar tareas periódicas, como muestrear continuamente un sensor o actualizar el estado de la interfaz en intervalos regulares.

¿Por qué es importante limpiar (clearTimeout o clearInterval) los temporizadores cuando ya no se necesitan?
En Node.js, todo temporizador activo mantiene vivo el proceso, consumiendo recursos y potencialmente causando fugas de memoria si no se detiene adecuadamente. Al invocar

clearTimeout(idTimeout) o clearInterval(idIntervalo) tan pronto como la tarea ha concluido o el controlador se desconecta, se liberan esos recursos y se evita que el programa ejecute código de forma inesperada, además de permitir que el proceso concluya correctamente cuando ya no hay más trabajo que realizar. En Ilyzaelle esta limpieza se ejecuta de manera automática mediante la asignación de {_id} en cada tipo de intervalo.

¿Para qué sirve una variable global como gVar en la plataforma Ilyzaelle?
Gracias a su carácter global dentro del proyecto, gVar actúa como punto único de intercambio de estado entre múltiples controladores: cada uno puede leer y escribir arrays, números o booleanos sin necesidad de implementar mecanismos complejos de mensajería. Así, gVar no solo facilita la persistencia temporal de datos, por ejemplo, registros de temperatura o conteos incrementales, sino que también resulta directamente accesible para el dashboard y las API REST de la plataforma, simplificando la generación de gráficos y la creación de interfaces HMI interactivas.

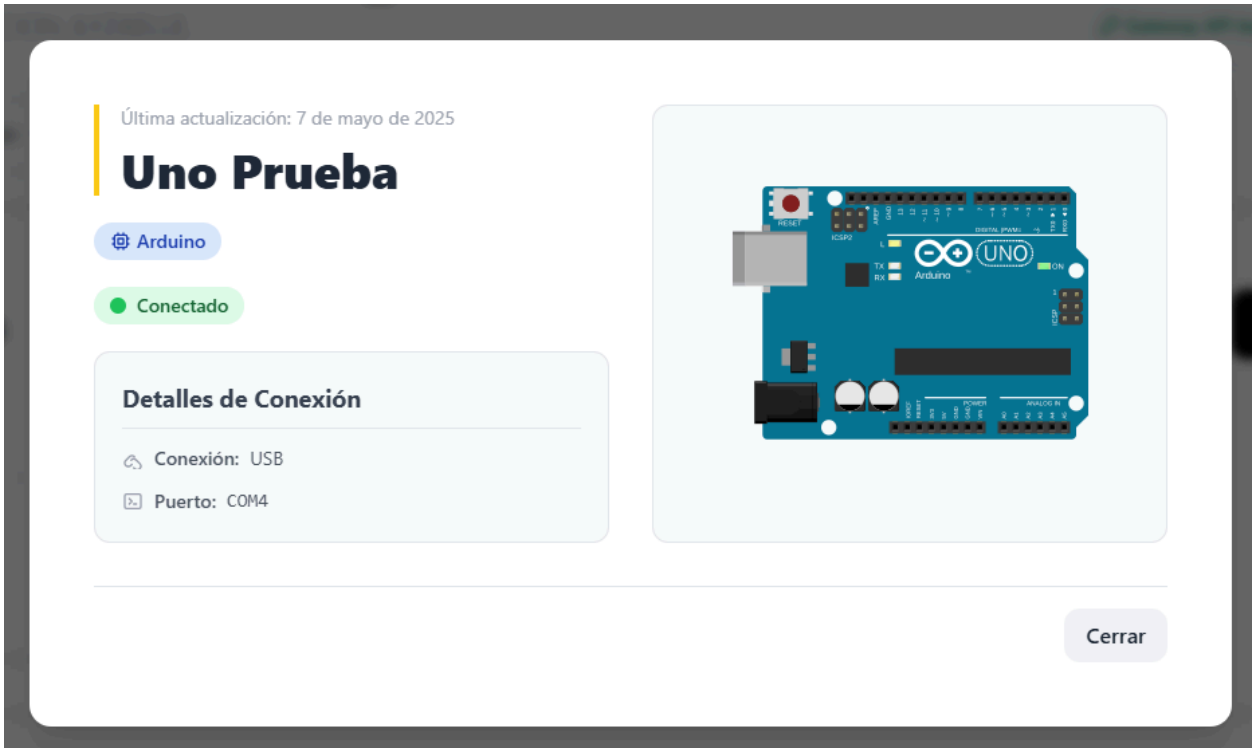
---

5. Procedimiento:

Paso 1: Debemos conectar los controladores con la plataforma. Para ello, haz clic en el nombre del primer controlador. Esto abrirá un menú donde podrás ver las especificaciones que configuraste previamente. En este menú, también encontrarás un botón que inicialmente dice **Desconectado**.

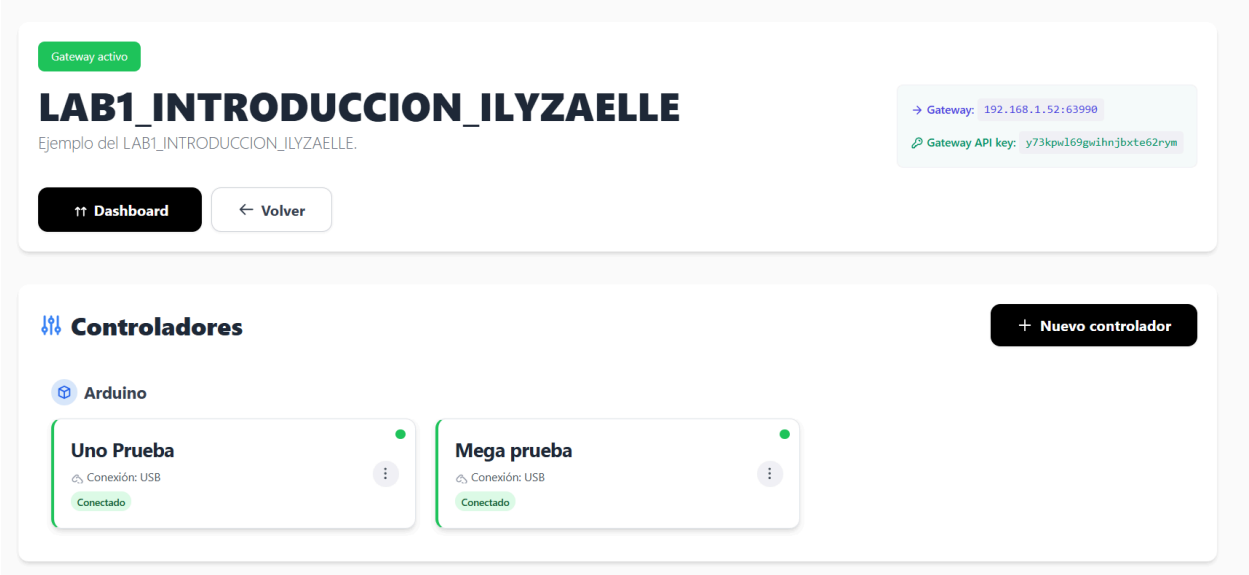
Haz clic en este botón para establecer la conexión con el Arduino.

Importante: Si oprimimos **Desconectado**, se realizará la conexión con el gateway y, después de unos segundos, el estado cambiará a **verde**. **OPRIME SOLO UNA VEZ Y ESPERA**.



Realizaremos lo mismo con el siguiente Arduino.

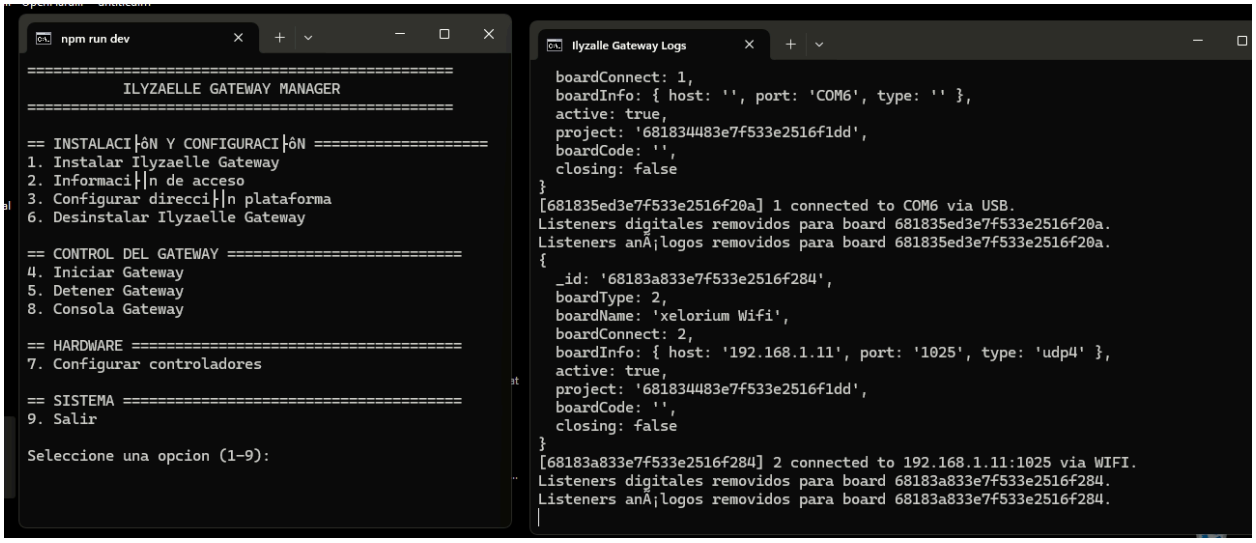
Veremos algo así en la vista general.



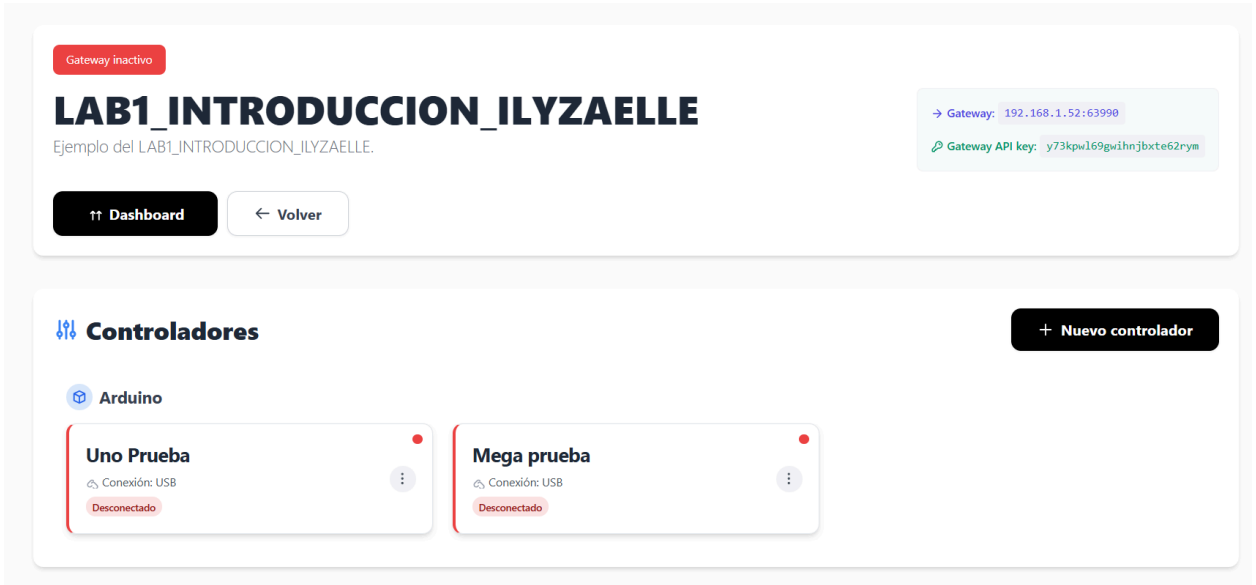
Los controladores se conectaron correctamente al gateway. Si ingresamos nuevamente a cada uno y seleccionamos la opción "Conectado", se desconectarán del gateway de forma manual.

Es posible conectar múltiples controladores mediante USB, siempre que la capacidad física del Gateway lo permita. De igual forma, se pueden conectar varios controladores a través de Wi-Fi, limitado únicamente por la capacidad del router.

Nota: Mientras los controladores estén en uso, la aplicación Illyzaelle Gateway Manager debe permanecer abierta. En caso de problemas de conexión, se puede acceder a la consola del Gateway seleccionando la opción 8. Esto abrirá una ventana emergente llamada Illyzaelle Gateway Logs, donde se mostrará información detallada sobre errores y mensajes del sistema. Aunque esta ventana puede cerrarse sin afectar el funcionamiento, la ventana principal del Gateway debe permanecer abierta en todo momento.

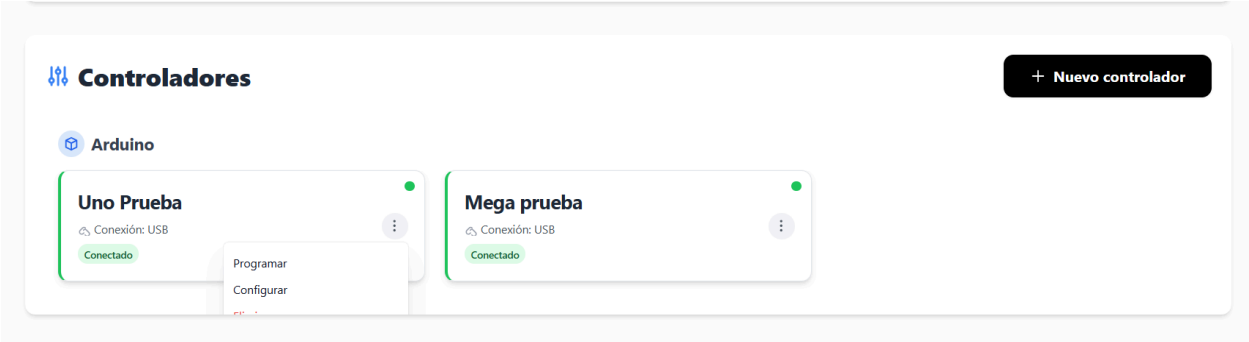


Si el **Illyzaelle Gateway Manager** se cierra por error, la plataforma desconectará automáticamente todos los controladores después de aproximadamente **5 segundos**. Esta desconexión se reflejará en la interfaz visualmente: todos los elementos aparecerán en **color rojo**, indicando la pérdida de comunicación.



Abrimos nuevamente *Illyzaelle Gateway Manager*, seleccionamos la opción 4 y procedemos a reconectar los controladores.

Paso 2. Entramos a programar, en el controlador número uno, desplegamos el menú (*drop-down menu*), seleccionamos la opción **Prueba 1** y presionamos el botón **Programar** para abrir el editor de código correspondiente al controlador.

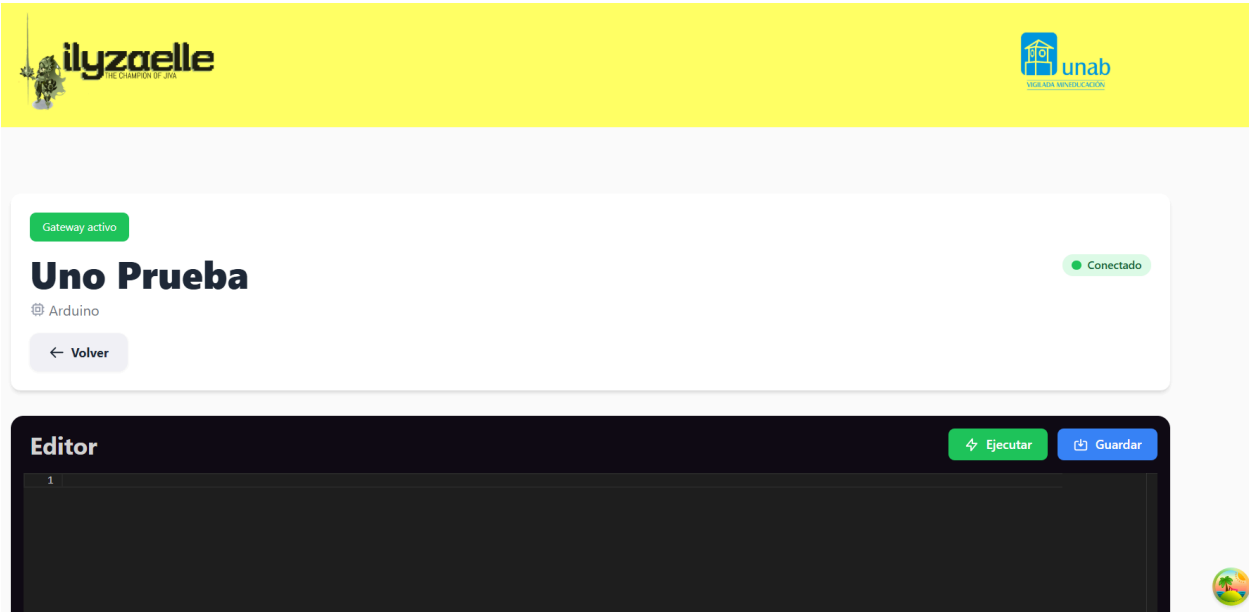


Se abrirá una página que permitirá programar los controladores. Esta interfaz incluye un editor para escribir el código, un botón verde **Ejecutar** y un botón **Guardar**.

- **Ejecutar**: guarda el código en la base de datos y lo envía al controlador para su ejecución inmediata.

- **Guardar**: almacena el código en la base de datos, pero no lo envía al controlador en ese momento.

Es importante tener en cuenta que, si se utiliza la opción **Guardar**, al desconectar y volver a conectar el controlador, este ejecutará automáticamente el último código almacenado en la base de datos.

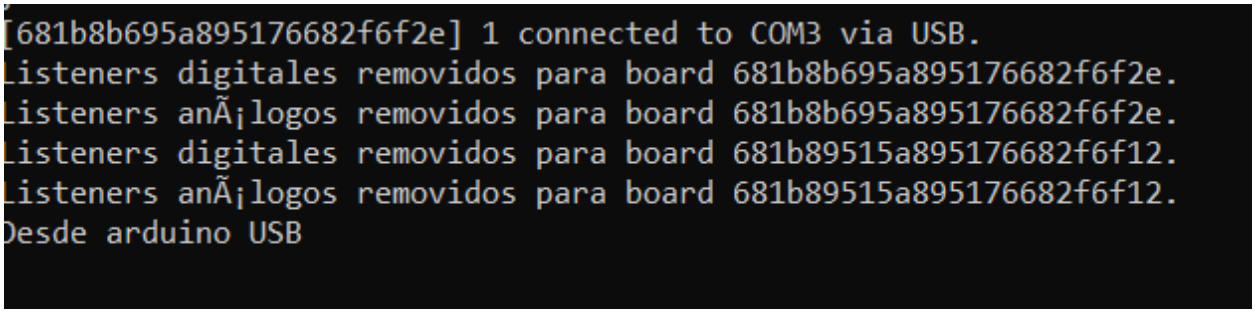


En el editor ingresamos el siguiente código, el cual nos permitirá visualizar un mensaje en la consola proveniente del controlador Arduino.

```
console.log("Desde arduino USB")
```

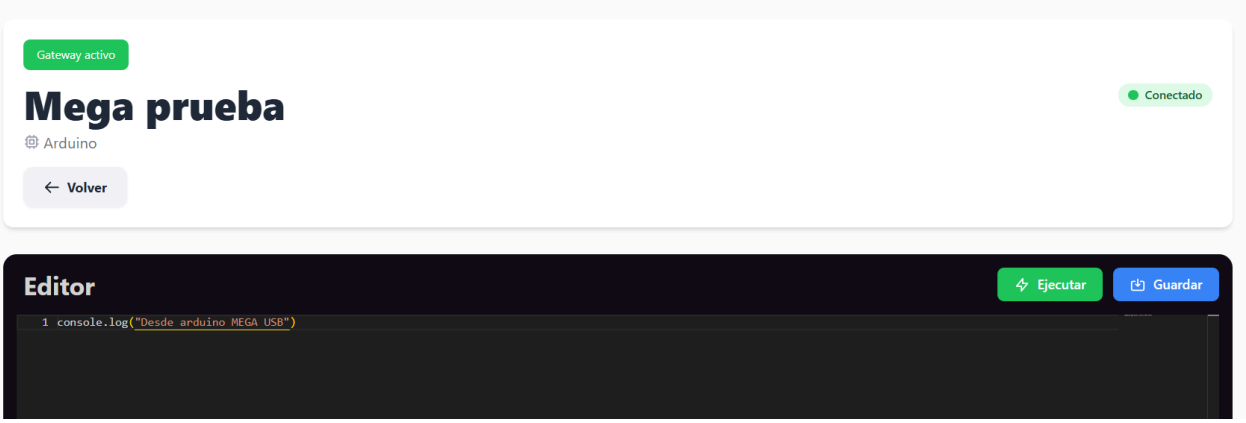
Y oprimimos en ejecutar.

A continuación, accedemos al menú del **Gateway Manager** y seleccionamos la opción **8**. En la consola que se abre, podremos visualizar el mensaje enviado desde el controlador, lo cual confirma que el código se está ejecutando correctamente.



Si nos fijamos, el código se ejecutó solo una vez. Esto ocurre porque, al estar basado en **Node.js**, no incluye un bucle (loop) de forma nativa. La ejecución en Node.js se maneja a través de **eventos** o, en algunos casos, mediante **bucles artificiales** que simulan este comportamiento.

Realizamos lo mismo con el otro arduino.



```
[681b8b695a895176682f6f2e] 1 connected to COM3 via USB.
Listeners digitales removidos para board 681b8b695a895176682f6f2e.
Listeners anÃ¡logos removidos para board 681b8b695a895176682f6f2e.
Listeners digitales removidos para board 681b89515a895176682f6f12.
Listeners anÃ¡logos removidos para board 681b89515a895176682f6f12.
Desde arduino USB
Listeners digitales removidos para board 681b8b695a895176682f6f2e.
Listeners anÃ¡logos removidos para board 681b8b695a895176682f6f2e.
Desde arduino MEGA USB
```

Teniendo esto en cuenta, existen dos opciones para crear un bucle artificial:

1. **setInterval**: Es una función nativa de **Node.js** que ejecuta un bloque de código en intervalos regulares. Sin embargo, no incluye manejo de errores (`try/catch`). Si ocurre un error, el Gateway podría fallar, requiriendo un reinicio.

2. **xelInterval**: Es una función propia de la plataforma que incorpora manejo de errores mediante `try/catch`. Esto reduce significativamente el riesgo de que el Gateway se caiga en caso de un fallo.

Con esto claro, procedemos a insertar el siguiente código en el **Arduino Mega** y ejecutarlo para probar su funcionamiento.

```
console.log("Desde arduino MEGA USB")
setInterval(() => {
 console.log("repeticion")
}, 1000, { _id })
```

En el código, **"Desde Arduino MEGA USB"** se ejecutará una sola vez, mientras que **"repetición"** se ejecutará cada segundo (1000 ms).

```
Listeners digitales removidos para board 681b8b695a895176682f6f2e.
Listeners anÃ¡logos removidos para board 681b8b695a895176682f6f2e.
Desde arduino MEGA USB
repeticion
repeticion
repeticion
repeticion
repeticion
```

Es obligatorio incluir el identificador {_id} para asegurar que la plataforma reconozca correctamente los eventos y los ejecute de manera adecuada.

```
xelInterval(() => {
 console.log("repeticion plataforma")
}, 2000, _id);
```

Con **xelInterval**, no es necesario poner el **_id** entre llaves **{}**.

Si revisamos la consola, veremos exactamente lo que se anticipó previamente.

```
Listeners digitales removidos para board 681b8b695a895176682f6f2e.
Listeners anÃ¡logos removidos para board 681b8b695a895176682f6f2e.
Desde arduino MEGA USB
repeticion
repeticion plataforma
repeticion
repeticion
repeticion plataforma
repeticion
repeticion
repeticion plataforma
repeticion
repeticion
```

Si definimos una variable y la incrementamos cada 4 segundos de la siguiente manera:

```
console.log("Desde arduino MEGA USB")
let contador = 0

xelInterval(() => {
 contador++
  console.log("Desde arduino MEGA USB " + contador)
}, 4000, _id);
```

En consola obtenemos:

```
Listeners digitales removidos para board 681b8b695a895176682f6f2e.
Listeners anÃ¡logos removidos para board 681b8b695a895176682f6f2e.
Desde arduino MEGA USB
Desde arduino MEGA USB 1
Desde arduino MEGA USB 2
Desde arduino MEGA USB 3
Desde arduino MEGA USB 4
Desde arduino MEGA USB 5
Desde arduino MEGA USB 6
Desde arduino MEGA USB 7
Desde arduino MEGA USB 8
```

Si insertamos el mismo código en Arduino USB, pero modificando el console.log, quedaría de la siguiente manera:

```
console.log("Desde arduino USB")
let contador = 0


xelInterval(() => {
 contador++
 console.log("arduino USB " + contador)
}, 2000, _id);
```

Podemos observar que el mensaje se imprime cada 2 segundos debido a la configuración del intervalo de ejecución en el código.

```
Listeners digitales removidos para board 681b89515a895176682f6f12.
Listeners anÃ¡logos removidos para board 681b89515a895176682f6f12.
Desde arduino USB
arduino USB 1
Desde arduino MEGA USB 20
arduino USB 2
arduino USB 3
Desde arduino MEGA USB 21
arduino USB 4
arduino USB 5
Desde arduino MEGA USB 22
```

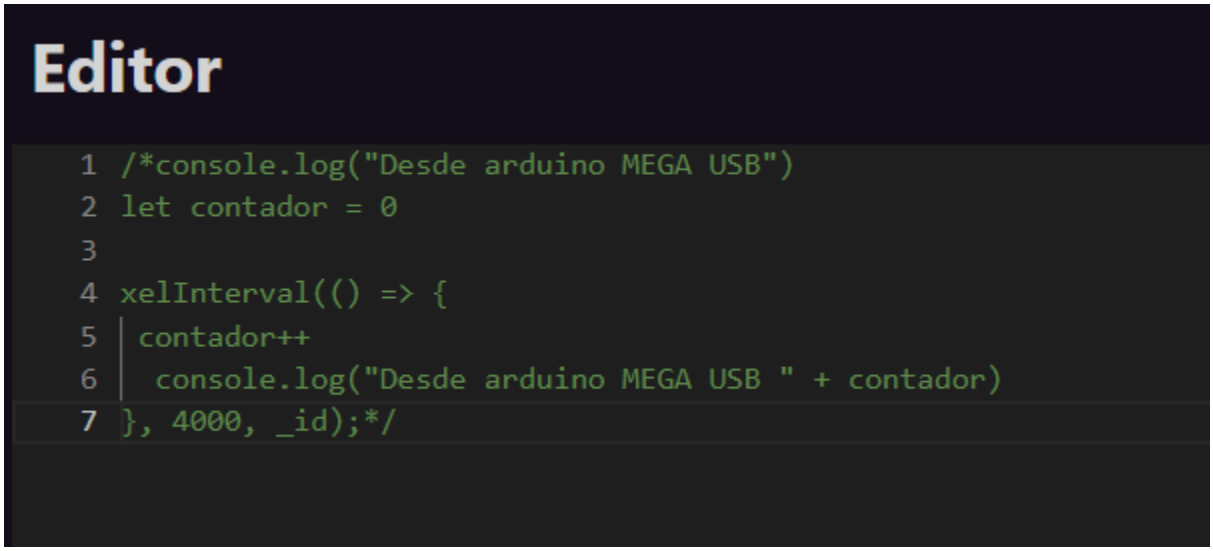
Con esto, podemos observar tres aspectos interesantes:

1. La ejecución del código es asincrónica, lo que significa que al ejecutar un controlador, no se reinicia el otro. La única manera de reiniciar ambos controladores simultáneamente es reiniciar el gateway y conectar los controladores al mismo tiempo. Esto se debe a que, en un entorno asincrónico, las tareas se ejecutan de manera independiente y no bloquean el flujo de ejecución de otras tareas.

2. Node.js no tiene un bucle de ejecución tradicional como en otros lenguajes. En lugar de un ciclo continuo, Node.js se basa en un event loop (bucle de eventos), que maneja las tareas asincrónicas de forma no bloqueante. Este bucle de eventos permite que el código se ejecute en un solo hilo, pero con la capacidad de manejar múltiples operaciones simultáneamente. Para simular un bucle o repetición periódica en Node.js, podemos usar funciones como setInterval(), que ejecutan un bloque de código a intervalos regulares sin bloquear el hilo principal. Esto crea un comportamiento similar a un "bucle" sin la necesidad de un ciclo explícito.

3. Usar el mismo nombre de variable para ambos controladores no genera conflictos porque cada controlador tiene su propio contexto de ejecución. En Node.js, las variables declaradas dentro de funciones o bloques de código (como los controladores) son locales a esos contextos. Por lo tanto, podemos repetir el nombre de la variable sin inconvenientes, ya que cada controlador mantendrá su propio espacio de memoria y no interferirá con el otro. Esto es posible gracias al alcance de las variables, que está limitado al bloque o función en la que fueron definidas.

Ahora procederemos a comentar ambos códigos y ejecutamos.



Observamos consola

```
Listeners digitales removidos para board 68183a833e7f533e2516f284.
Listeners anÃ¡logos removidos para board 68183a833e7f533e2516f284.
```

Los intervalos han detenido su ejecución. Si volvemos a ejecutar ambos códigos, las variables se reiniciarán desde cero.

Ahora bien, para permitir la comunicación entre los controladores, utilizamos una variable global, la cual se encarga de almacenar y gestionar las variables que seleccionemos, ya sean de tipo número, array o booleano. Es importante destacar que, al crear una variable en el contexto global, no debe existir otra con el mismo nombre; de lo contrario, se sobrescribirá y perderemos la referencia necesaria.

6. Variables Globales

Paso 1. Ingresamos al **Arduino Mega** y escribimos el siguiente código.

Nota: Cuando se usan **variables globales**, no es necesario usar **let** ni **const**, ya que estas palabras clave son para declarar variables con un alcance local o bloque específico.

```
gVar[project].registroTemperatura = []
xelInterval(() => {
 let temperatura = Math.random() * (40 - 20) + 20;
 gVar[project].registroTemperatura.push(temperatura);
}, 1000, _id);
```

gVar[project] es la variable global, y se le asignan propiedades utilizando el operador punto (.).

El siguiente código se encarga de inicializar las variables necesarias y, cada segundo, agregar un nuevo valor al array registroTemperatura, simulando la lectura de un sensor.

- estado es una variable booleana que indica la presencia de una alerta.
- margenTemperatura define el umbral a partir del cual se activa la alerta (estado).

Ahora, vamos al controlador Arduino Uno y escribimos el siguiente código:

```
gVar[project].estado = false
gVar[project].margenTemperatura = 31
```

```
xelInterval(() => {
  let temperatura = gVar[project].registroTemperatura;
  let margen = gVar[project].margenTemperatura;



  if (temperatura[temperatura.length - 1] > margen) {
    gVar[project].estado = true;
  } else {
    gVar[project].estado = false;



  }
}, 1000, _id);
```

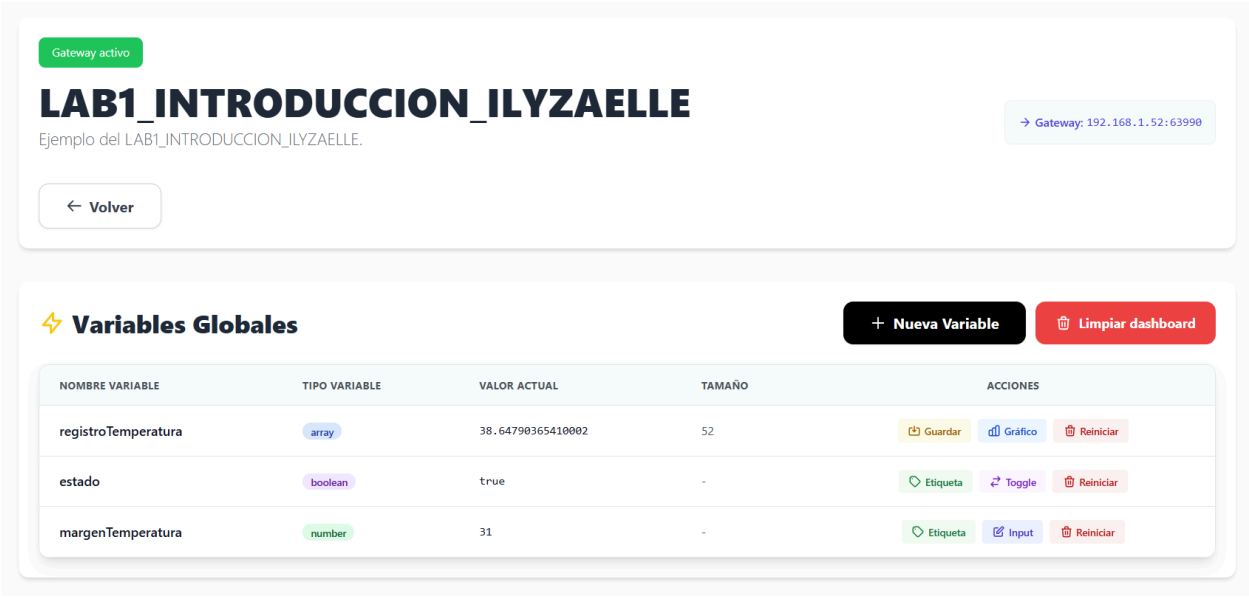
Con este código, tomamos el último valor del array registroTemperatura y evaluamos su valor:

- Si es mayor a 31, la variable estado se establece en true.
- Si es menor o igual a 31, el estado se establece en false.

7. Dashboard

Ejecutamos el código y luego ingresamos al dashboard para observar el comportamiento en tiempo real.

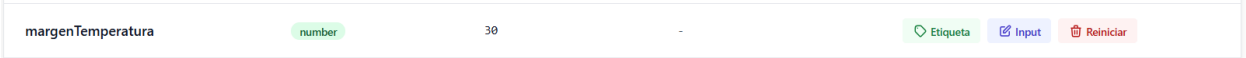
En el **Dashboard**, además de visualizar las variables globales que ya han sido creadas, también podemos crear una nueva variable global utilizando el botón **+ Nueva variable**. Una vez creada, esta variable puede ser utilizada desde cualquier controlador.
Además, contamos con el botón **Limpiar dashboard**, que nos permite eliminar las representaciones gráficas creadas previamente, facilitando así la organización y visualización de los elementos activos.



Las **variables globales** también ofrecen diferentes acciones según su tipo:

- **Array:**
    - Muestra su **tamaño**, es decir, la cantidad actual de datos almacenados.
    - **Guardar:** Permite almacenar el vector de datos actual junto con el vector de tiempo, para su uso posterior.
    - **Gráfico:** Genera una visualización gráfica del contenido del array.
    - **Reiniciar:** Elimina los datos y devuelve la variable a su estado inicial.

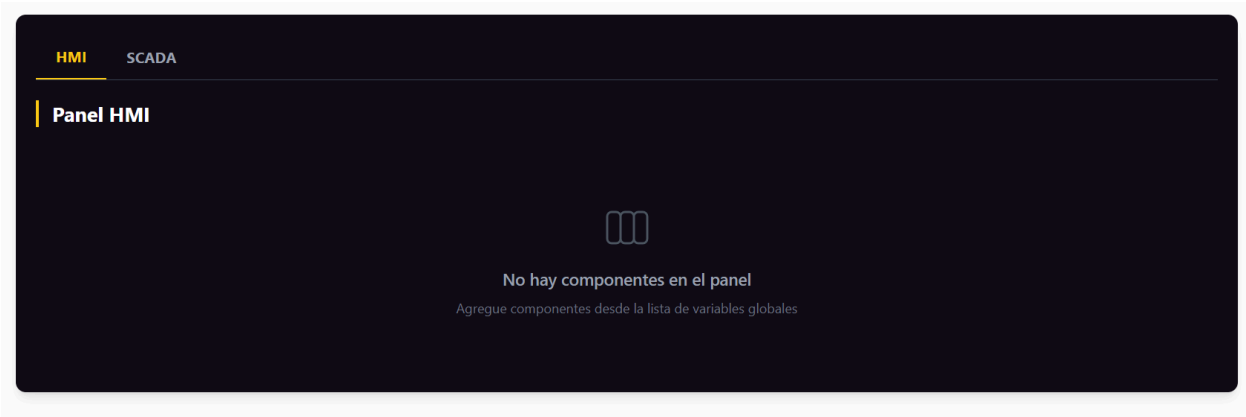
- **Booleano:**
    - Puede representarse como una **etiqueta** que muestra el valor actual (true/false).
    - También puede mostrarse como un **toggle** (interruptor) que permite modificar el valor manualmente.
    - Incluye la opción **Reiniciar** para restablecer su valor inicial.



- **Number (número):**
    - Se visualiza mediante una **etiqueta** que indica el valor actual.
    - Tiene una opción **Input** que permite cambiar el valor manualmente.
    - También incluye el botón **Reiniciar** para restaurar el valor original.

Estas opciones permiten una gestión flexible e interactiva de las variables desde el Dashboard.

Por otro lado el dashboard también permite ver gráficamente las variables esto con un apartado de HMI, donde podremos agregar elementos para ver las variables globales.



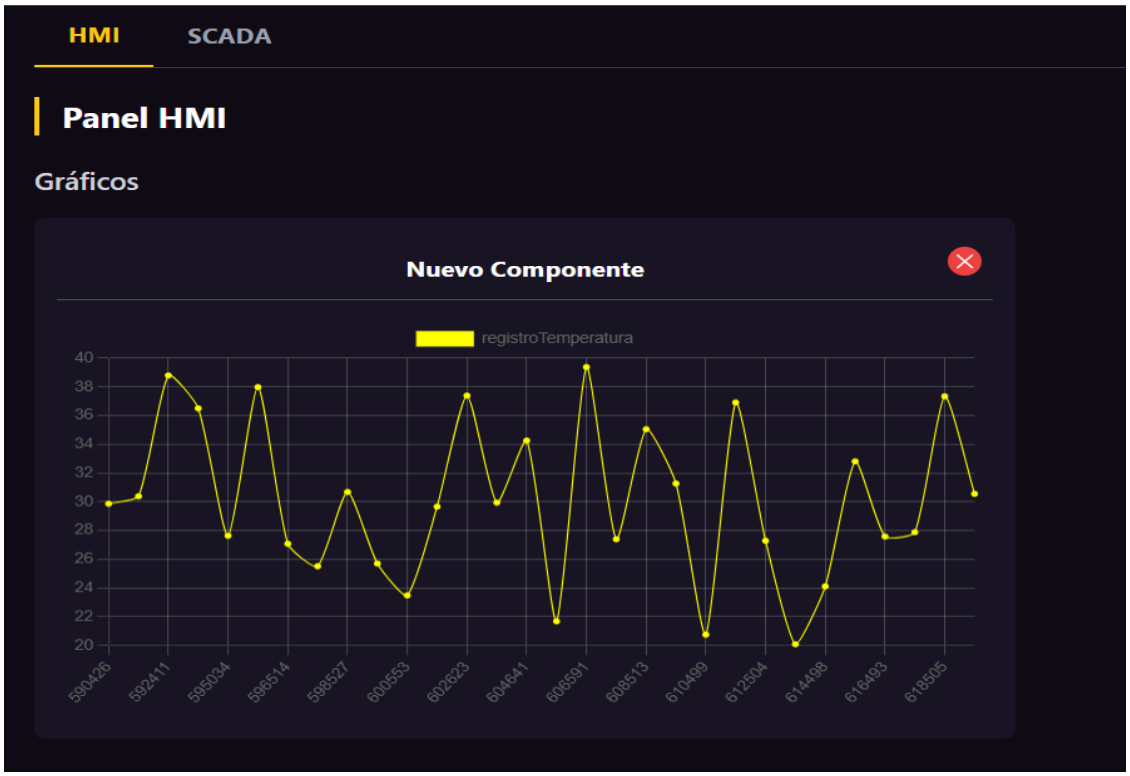
Paso 1: Agregar variables al HMI

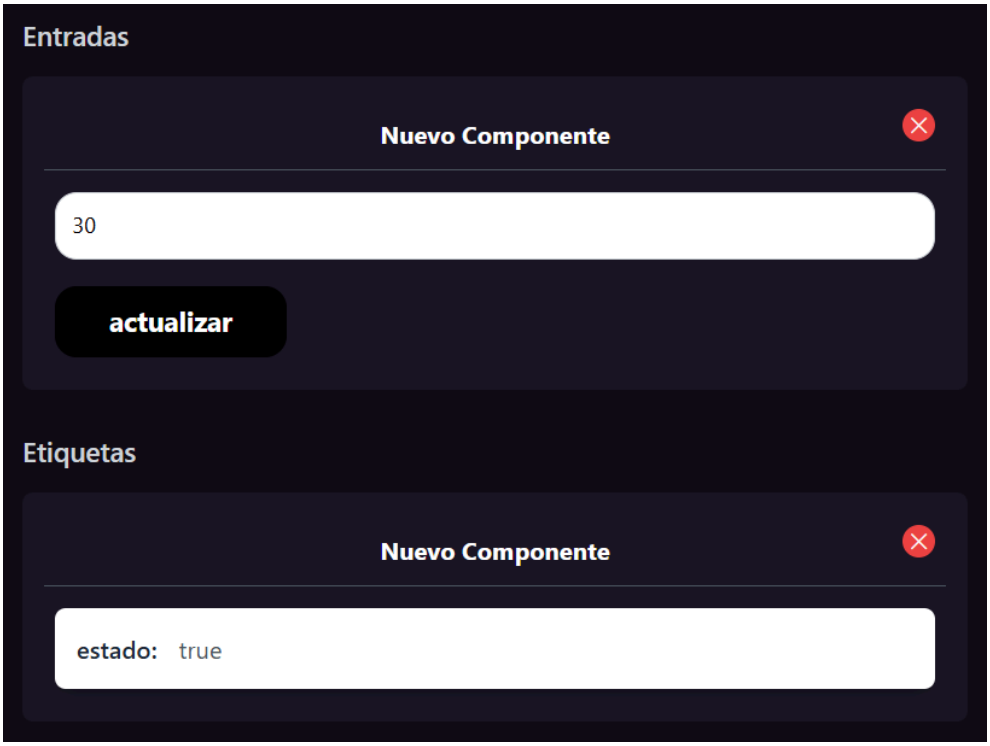
Utilizando las acciones disponibles en las variables globales descritas anteriormente, realizaremos lo siguiente:

- De la variable global **registroTemperatura**, agregamos un **gráfico** oprimiendo el botón **Gráfico** en las acciones.

- De la variable **estado**, agregamos una **etiqueta** para visualizar su valor actual.

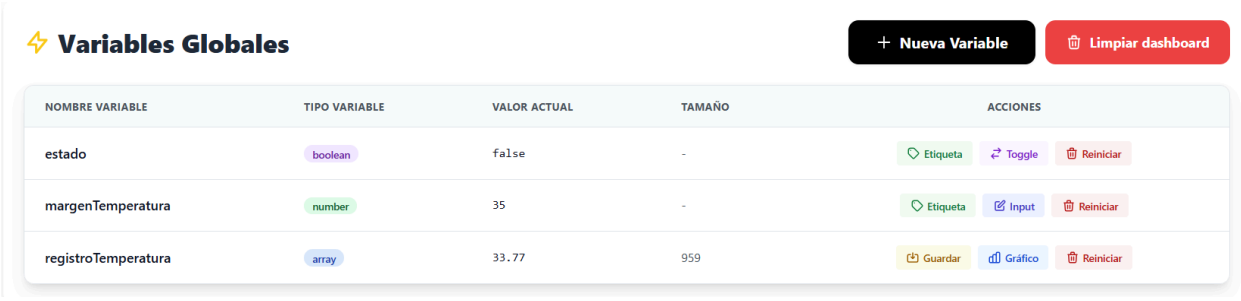
- De **margenTemperatura**, agregamos un **input** que nos permitirá modificar su valor manualmente desde el HMI.





A continuación, en el **HMI** aparecerán las opciones que seleccionamos previamente, mostradas en el mismo orden en que las agregamos.

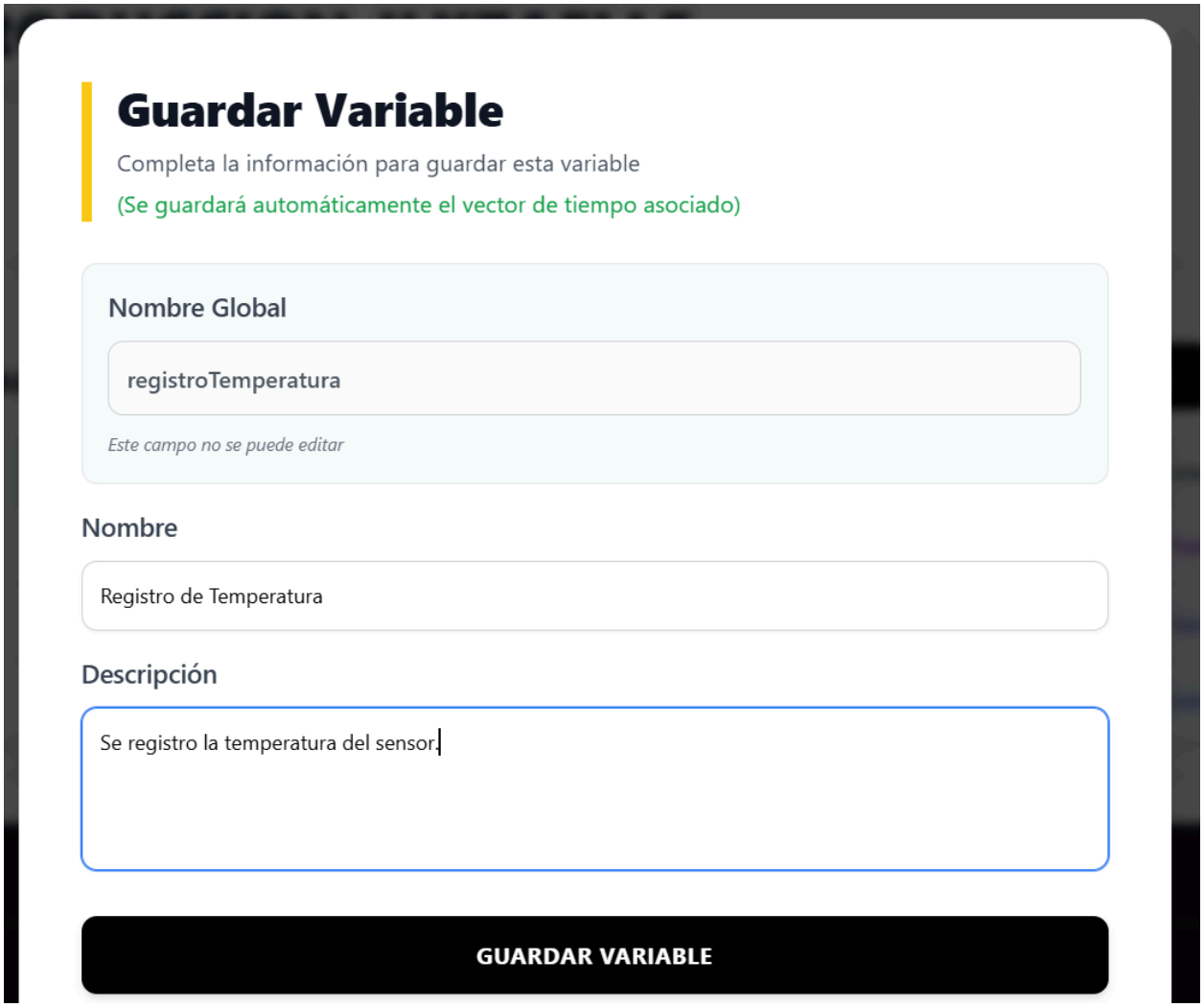
Ahora, en la sección **Entradas**, localizamos la variable **margenTemperatura**. Cambiamos su valor por otro número, por ejemplo **35**, y luego oprimimos el botón **Actualizar** para que el nuevo valor se aplique en tiempo real.



Este cambio se refleja automáticamente en las variables globales, y, de igual forma, modificará la lógica de la variable estado, ya que esta depende directamente del valor de margenTemperatura.

Paso 2. Guardar Datos.

La opción para guardar datos está disponible en el array. Dentro de las acciones, encontraremos un botón que, al ser presionado, mostrará un menú para guardar una variable. Allí deberemos completar la información requerida y luego hacer clic en "Guardar variable".

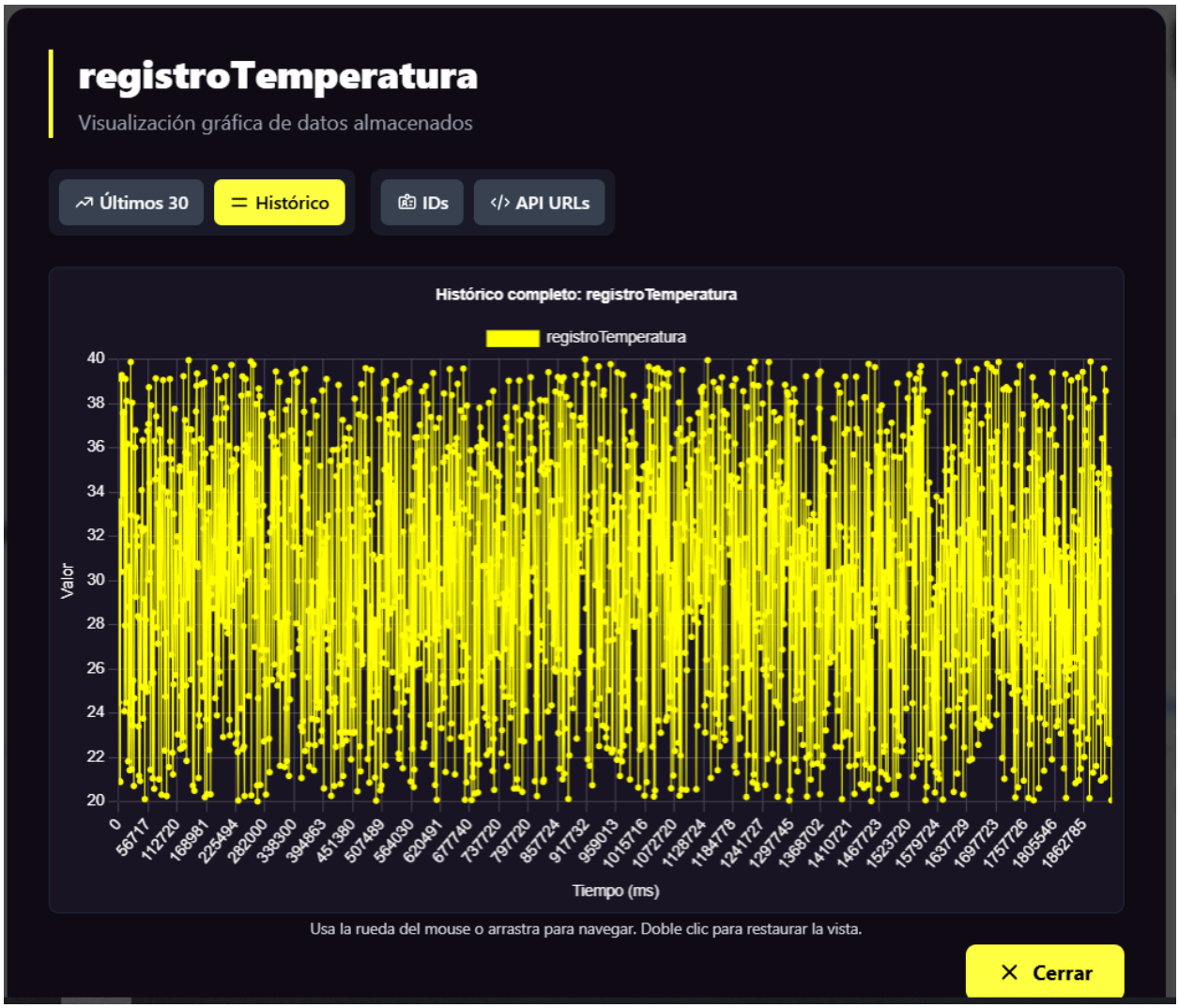


Luego, al regresar al proyecto, encontraremos la variable guardada en la sección "Variables almacenadas".

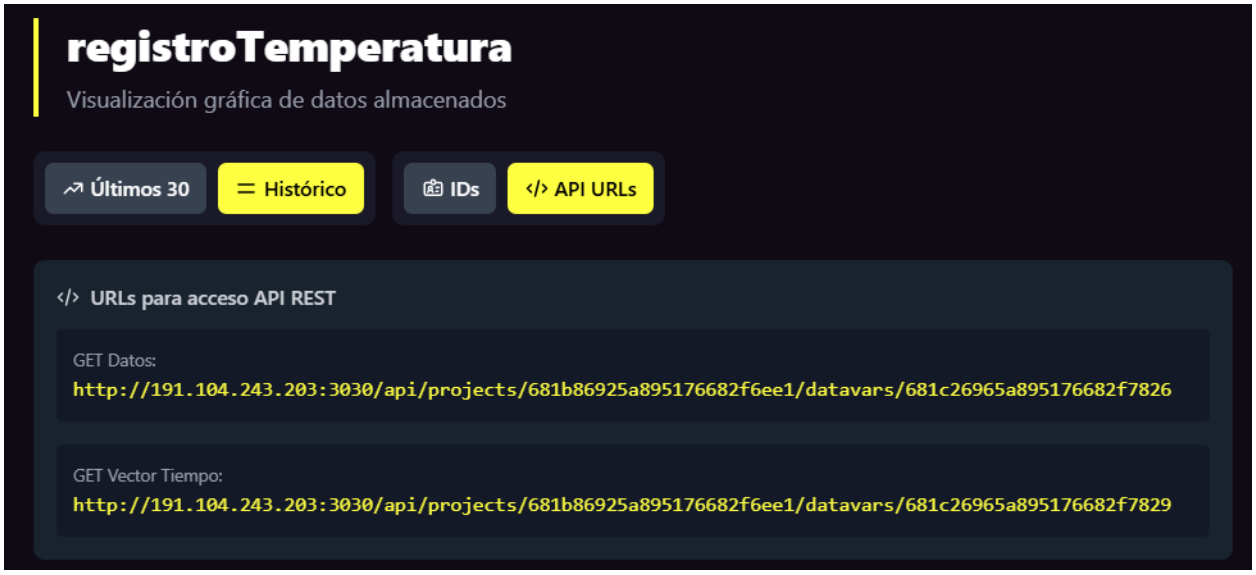
Al presionar el botón "Ver gráfico", se mostrará la variable global representada en una gráfica completa. En la interfaz encontraremos botones como:

- **Últimos 30**: Muestra los últimos 30 datos registrados.
- **Histórico**: Permite ver todos los datos registrados a lo largo del tiempo.
- **IDs**: Muestra los identificadores de MongoDB asociados a los datos.

- **API URLs**: Proporciona las URLs necesarias para acceder a la API REST.



En nuestro caso lo guardaremos por **API URLs,**



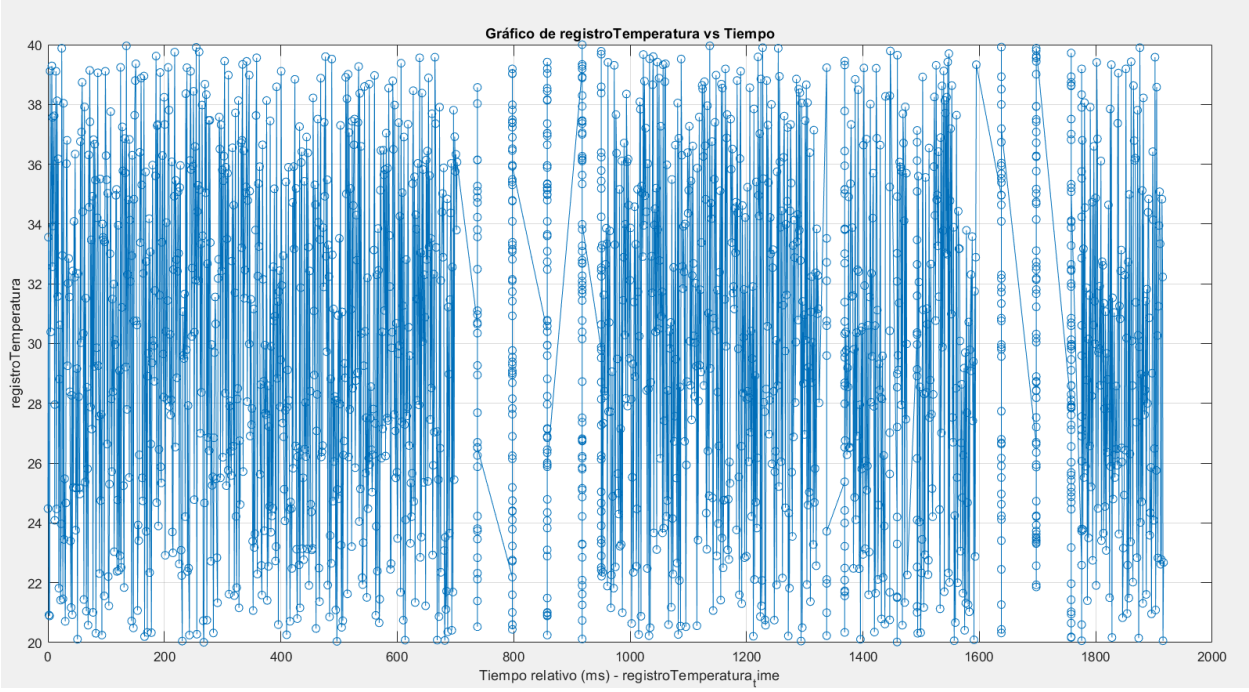
Paso 3. Exportar datos via matlab.

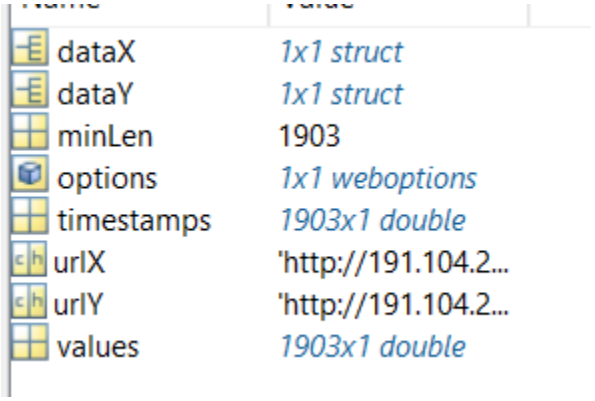
Teniendo los URLs de los datos podemos exportarlo via matlab usando el siguiente codigo en matlab.

```
clc

clear all

% URLs

urlY = 'http://191.104.243.203:3030/api/projects/681b86925a895176682f6ee1/datavars/681c26965a895176682f7826';

urlX = 'http://191.104.243.203:3030/api/projects/681b86925a895176682f6ee1/datavars/681c26965a895176682f7829';

% Opciones

options = weboptions('ContentType', 'json');

% GET

dataX = webread(urlX, options);

dataY = webread(urlY, options);

% Tiempos: restar el primero para que comience en 0

timestamps = dataX.gVar - dataX.gVar(1);  % tiempo relativo en ms
```

```
values = dataY.gVar;

% Recorte si son de distinta longitud

minLen = min(length(timestamps), length(values));

timestamps = timestamps(1:minLen);

values = values(1:minLen);

% values = str2double(values);  Convierte la matriz de celdas en una matriz
normal cualdo el dato pasa por tofixed

% Graficar

figure;

plot(timestamps/1000, values, '-o');

xlabel(['Tiempo relativo (ms) - ' dataX.nameGlobalVar]);

ylabel(dataY.nameGlobalVar);

title(['Gráfico de ' dataY.nameGlobalVar ' vs Tiempo']);

grid on;
```

Los datos pueden exportarse a **MATLAB** para su posterior procesamiento o análisis avanzado. Desde allí, también es posible guardarlos en **Excel**. En este caso, los datos fueron graficados para visualizar su comportamiento en el tiempo.





A partir de los campos **"timestamps"** y **"values"**, es posible exportar los datos a Excel para su almacenamiento o continuar con el análisis.

6. Actividad:

- Define una variable global que contabilice cuántas veces la variable estado adquiere el valor true y visualiza su evolución temporal en un gráfico dentro del dashboard.

7. Bibliografía:

- Flanagan, D. (2011). JavaScript: The Definitive Guide (6th ed.). O'Reilly Media.
- Greengard, S. (2015). The Internet of Things. MIT Press.
- Stallings, W. (2013). Data and Computer Communications (10th ed.). Pearson.
- Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks (5th ed.). Prentice Hall.
- Buna, S. (2012). Socket.IO Real-time Web Application Development. Packt Publishing.