

Taller de Proyecto II

Informe de Avance II

Sistema de monitoreo y control de temperatura en tiempo real utilizando con MQTT y GPRS (G2)

Grupo de Desarrollo

Dehan, Lucas–565/1

Duarte, Víctor–1055/7

Kleinubing, Hernán –1614/6

Palacio, Constantino – 1806/2

Contenido

Contenido	2
1. Introducción	3
2. Objetivos	4
2.1. Objetivos Principales.....	4
2.2. Objetivos Secundarios.....	5
2.2.1. Adición de módulos controlados por el sistema.....	5
2.2.2. Adición de módulo RTC	5
2.2.3. Adición de módulo de alimentación por batería.....	5
3. Identificación de Partes	6
3.1. Componentes de Hardware y Presupuesto	6
3.2. Componentes de Software.....	6
3.2.1. Firmware del Sistema.....	6
3.2.2. Servidor MQTT (Interfaz EMQX)	6
3.2.3. Comunicación entre Arduino y Servidor MQTT	7
3.2.4. Aplicaciones y Procesamiento de Datos	7
4. Grado de Avance.....	8
4.1. Hardware y Firmware.....	8
4.2. Servidor e Interfaz Web	11
4.2.1. Servidor	11
4.4. Aplicación Web temperatura.....	14
4.5. Problemas más importantes y soluciones.....	15
5. Documentación Relacionada.....	16

1. Introducción

En un mundo cada vez más interconectado, el monitoreo preciso de la temperatura desempeña un papel fundamental en numerosos aspectos de la vida cotidiana y en diversas industrias.

Sin embargo, el acceso a estos datos en tiempo real ha sido un desafío constante, especialmente en ubicaciones remotas o en áreas donde la comunicación es limitada. La falta de una solución efectiva para la adquisición y visualización de datos en tiempo real ha llevado a ineficiencias operativas, pérdida de productos sensibles a la temperatura y, en algunos casos, incluso a situaciones de seguridad crítica.

El presente proyecto se centra en abordar esta problemática mediante la creación de un dispositivo electrónico capaz de medir con precisión la temperatura en diversos entornos y transmitir esos datos a través de una conexión GPRS (*General Packet Radio Service*) haciendo uso de la red de telefonía celular. La visualización de estos datos se realizará de manera sencilla y accesible por medio de, por ejemplo, una aplicación web.

2. Objetivos

2.1. Objetivos Principales

El objetivo principal del proyecto es realizar un prototipo como prueba de concepto que sea verificable en el ámbito del aula con la placa Arduino de un sistema capaz de realizar distintas tareas como:

1. Medir la temperatura de un ambiente.
2. Usar las redes de telefonía celular mediante un módulo GPRS para conectar la placa Arduino a un servicio de IoT. En este proyecto se usará el servicio EMQX con el protocolo MQTT para enviar los sensores.
3. La aplicación web será una interfaz de usuario la cual obtendrá y mostrará los datos proporcionada por EMQX, proporcionando gráficos y mensajes de texto al usuario de las temperaturas y una interfaz que le permitirá establecer dicha temperatura dentro de un rango máximo y mínimo.

El siguiente diagrama de bloques ilustra lo descrito anteriormente en términos generales.

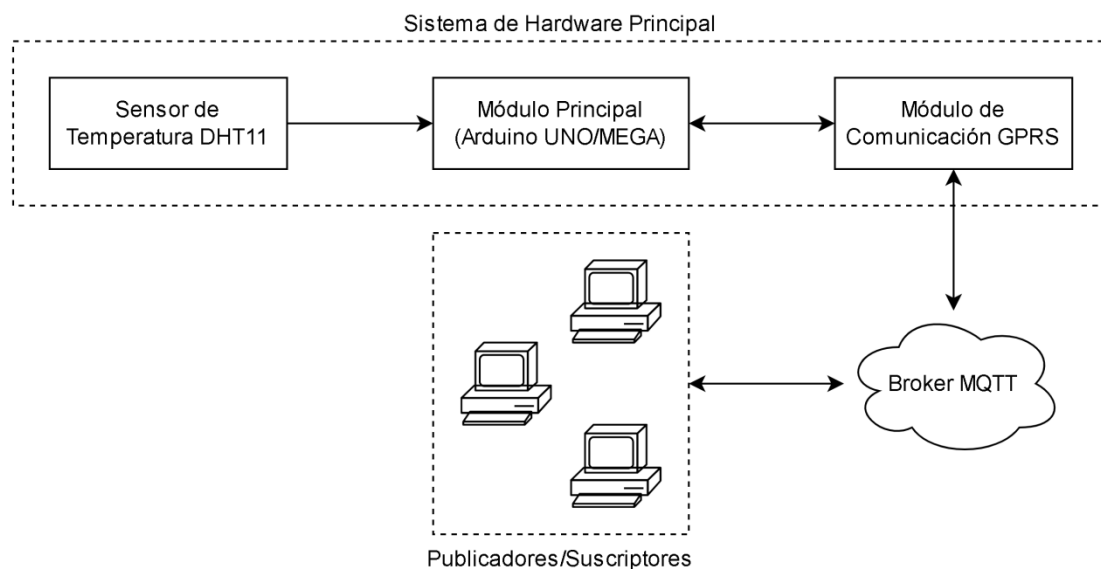


Figura 1. Diagrama de bloques del sistema

De acuerdo a la figura 1, la placa Arduino recibiría los datos de temperatura del módulo de sensor DHT11, para luego empaquetarlos y transmitirlos a través de la red celular usando el módulo GPRS hacia un servidor MQTT. A éste se conectarán usuarios suscriptores que acceden a los datos del sensor y los almacenarán localmente, quienes a su vez podrían comunicarse en sentido inverso con la placa Arduino para establecer el rango de temperatura del sistema. Un aspecto omitido del diagrama anterior es la alimentación del sistema.

2.2. Objetivos Secundarios

A continuación se incluye un listado de posibles mejoras o ampliaciones realizables al sistema una vez cumplidos los objetivos principales.

2.2.1. Adición de módulos controlados por el sistema

Podrían incluirse, una vez probado el funcionamiento del prototipo, dispositivos que reflejen el acondicionamiento de la temperatura cuando la misma se encuentra fuera de rango. Estos podrían ser:

- Un ventilador para cuando la temperatura es mayor al límite superior indicado
- Una resistencia para cuando la temperatura es menor al límite inferior indicado

De no ser posible la adición de dichos periféricos, el sistema incluiría al menos un indicador LED que se encendería sólo cuando la temperatura se encuentra fuera del rango seleccionado.

2.2.2. Adición de módulo RTC

Dado que se debe registrar la fecha y la hora de las mediciones, resultaría más preciso que el mismo módulo Arduino haga estas marcas temporales, puesto que las comunicaciones (y más aún las inalámbricas) conllevan un retardo considerable. Para este propósito se incluiría un módulo RTC (*Real-Time Clock*) y se adosaría esta marca temporal al paquete de datos transmitido al bróker MQTT. Estos módulos generalmente llevan una pila para mantener el circuito de reloj activo aun cuando se corta la alimentación del circuito principal y suelen tener una vida útil prolongada.

2.2.3. Adición de módulo de alimentación por batería

Como una posible mejora se propone "independizar" el sistema en funcionamiento de una computadora o incluso de la red eléctrica agregando una batería. Podría agregarse al software una verificación del nivel de carga y que se envíe una notificación por el mismo canal que las mediciones a los suscriptores informando que la batería de alimentación debe ser recargada o reemplazada.

3. Identificación de Partes

3.1. Componentes de Hardware y Presupuesto

Los siguientes materiales componen el sistema de hardware esencial para el funcionamiento del sistema. La tabla incluye los precios de cada componente:

<i>Componente</i>	<i>Precio</i>
Placa Arduino UNO	\$7600
Módulo Sensor DHT11	\$1800
Módulo GPRS SIM908C	\$43989
Chip Tuenti + Recarga	\$1200
Fuente de alimentacion 9V/2A	\$5600
Costo total	\$60189

Tabla 1. Presupuesto de los materiales obtenidos (Arduino UNO, DHT11, GPRS SIM 908C: mercadolibre, 2023).

Componentes adicionales de hardware omitidos del listado anterior incluyen el equipo informático sobre el que funcionará el servicio MQTT, los equipos desde los cuales se conectarán los suscriptores y las fuentes de alimentación de cada uno de los componentes de hardware utilizados, especialmente la fuente externa para el módulo de comunicación GPRS.

3.2. Componentes de Software

Los siguientes componentes forman el sistema de software:

3.2.1. Firmware del Sistema

Se utilizará para el control del sistema, monitoreo del sensor y comunicación con el bróker MQTT. Será programado en lenguaje C y será ejecutado directamente sobre la placa de desarrollo Arduino. El firmware abarca los siguientes procesos:

- Control del sistema: El firmware controla las acciones y operaciones de la placa Arduino, incluyendo la adquisición de datos de sensores y la toma de decisiones.
- Monitoreo del sensor: Implementa rutinas para leer datos de los sensores conectados y garantizar que los datos sean precisos y consistentes.
- Comunicación MQTT: Establece una comunicación bidireccional con un bróker MQTT para enviar y recibir datos de manera eficiente.

3.2.2. Servidor MQTT (Interfaz EMQX)

El servidor MQTT actúa como un componente central en la infraestructura del proyecto, y se compone de los siguientes procesos de software:

- Bróker MQTT: La interfaz EMQX actúa como un bróker MQTT que facilita la publicación y suscripción de datos entre los dispositivos IoT y otros sistemas.
- Gestión de temas: Gestiona la organización de datos en temas específicos para permitir una transmisión eficiente y una fácil suscripción por parte de los clientes.

3.2.3. Comunicación entre Arduino y Servidor MQTT

- **Establecimiento de Conexión:** El firmware en la placa Arduino inicia y mantiene una conexión segura con el servidor MQTT a través de la red.
- **Publicación de Datos:** Los datos recopilados por los sensores se publican en temas MQTT para su posterior procesamiento.
- **Suscripción de Datos:** Si es necesario, la placa Arduino puede suscribirse a temas MQTT específicos para recibir comandos o actualizaciones desde el servidor.

3.2.4. Aplicaciones y Procesamiento de Datos

- **Visualización y Análisis:** Las aplicaciones de visualización y análisis pueden consumir los datos desde el servidor MQTT para proporcionar información en tiempo real y análisis históricos.
- **Alertas y Notificaciones:** Las aplicaciones pueden generar alertas y notificaciones basadas en los datos recibidos, mejorando la capacidad de respuesta.
- **Almacenamiento de Datos:** Si es necesario, los datos también pueden almacenarse en bases de datos para su posterior referencia y análisis.

4. Grado de Avance

4.1. Hardware y Firmware

Se realizaron las conexiones de todos los componentes electrónicos que conforman el sistema como se muestra en la siguiente imagen (**Ver fig 3**). Una placa Arduino UNO conectada a un sensor dht11 y a una placa que contiene el módulo GPRS SIM908C la cual trabaja con una alimentación de 9V a 2A.

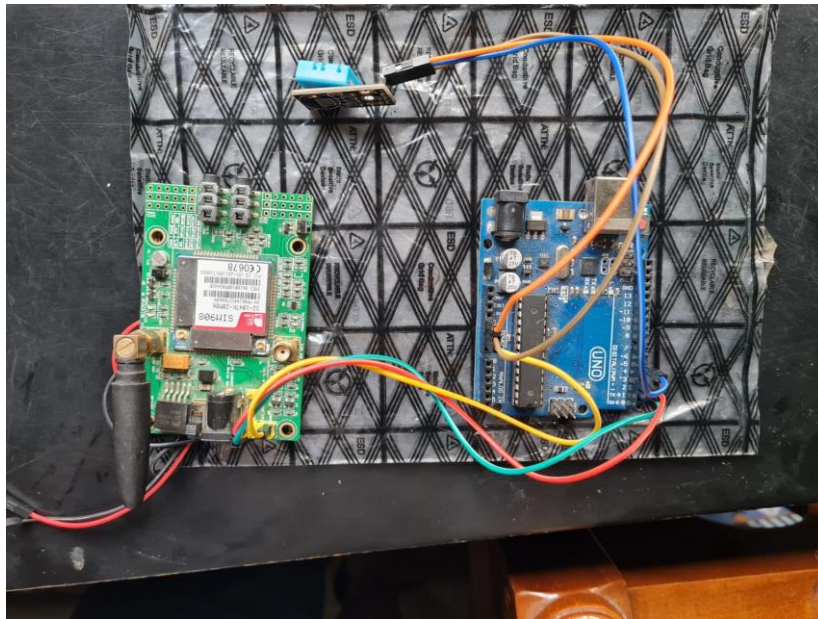


Figura 3. Esquema de conexiones físicas del sistema.

En la figura podemos observar la conexión preliminar de todos los componentes electrónicos que conforman el sistema. En el centro se observa el Arduino UNO, conectado al módulo GPRS (izquierda) a través de los pines 2 y 3 a TX y RX respectivamente. Por otro lado, se observa la conexión al sensor DHT11 (derecha) por los pines de alimentación de +5V y GND del Arduino; y al pin 5 como pin de datos.

Para el firmware, se utilizaron las siguientes librerías:

- DHT: Para el uso del sensor de temperatura.
- SoftwareSerial: Para habilitar la comunicación serial en pines digitales además de los designados por defecto por Arduino (pin 0 y 1 para Arduino UNO)
- ArduinoJson: Para el envío de datos en formato JSON.
- TinyGsmClient y PubSubClient: Para facilitar la comunicación por GPRS.

A continuación, se describe en forma de pseudocódigo el firmware del sistema, tomando como referencia el estilo de programación usado en Arduino IDE:


```

void setup () {
    Inicializar la comunicación serie entre Arduino y módulo GPRS
    Inicializar el sensor de temperatura DHT11
    Inicializar el módulo GPRS
    Conectar a la red telefónica
    Conectar a la red GPRS
    Configurar los datos del servidor MQTT
}

void loop () {
    Loop para reconectarse en caso de desconectarse de la red
    telefónica y/o GPRS
    Si no está conectado al servidor MQTT, conectarse al
    servidor.
    Caso contrario, si se cumplió 1 minuto desde la anterior
    comunicación/inicio de programa:
        Leer los datos del sensor
        Si no ocurrieron errores al obtener los datos del sensor:
            Se crea un objeto JSON y se publica como mensaje MQTT
        Caso contrario, se transmite un mensaje informando del
        error
        Llama un método loop para que no se corte la conexión al
        servidor.
}

```

Para conectarse a la red GPRS, el firmware debe tener los datos necesarios del APN (*Access Point Name*) del proveedor del servicio. En este proyecto se utiliza Tuenti como proveedor. Para la configuración de la conexión con el servidor MQTT se utiliza la dirección y puerto del servidor junto con las credenciales para acceder. Las mismas se especifican dentro del código (ubicación en el repositorio: `/GPRS/test-sim8001.ino`).

En principio el sistema podía leer periódicamente las muestras del sensor de temperatura, pero no era capaz de enviar dichos datos al servidor debido a la falta de una fuente de alimentación para el módulo GPRS. Luego de obtener una fuente de alimentación para el SIM908C de 9V/2A y se intentó realizar pruebas con el primer prototipo del firmware pero no se obtenía respuesta por parte del módulo por lo que se concentró en descubrir el motivo de la falla. La causa era por dos motivos:

- Falta de una conexión de GND en el módulo.
- Cada vez que se enciende el SIM908C, es necesario realizar un reinicio por medio de apretar un botón al costado de la placa por menos de 3 segundos.

Una vez encontrada la falla se probaron distintas funcionalidades del SIM908C como enviar mensajes, hacer llamadas y handshakes.

Con las anteriores pruebas arrojando resultados satisfactorios, se procedió a probar si se podía realizar una conexión exitosa al MQTT Broker. Durante estas pruebas se

demostraron las limitaciones de redes GSM/GPRS debido a la realización de pruebas en lugares con limitada señal de red, aunque eventualmente se realizó una comunicación exitosa con el servidor.

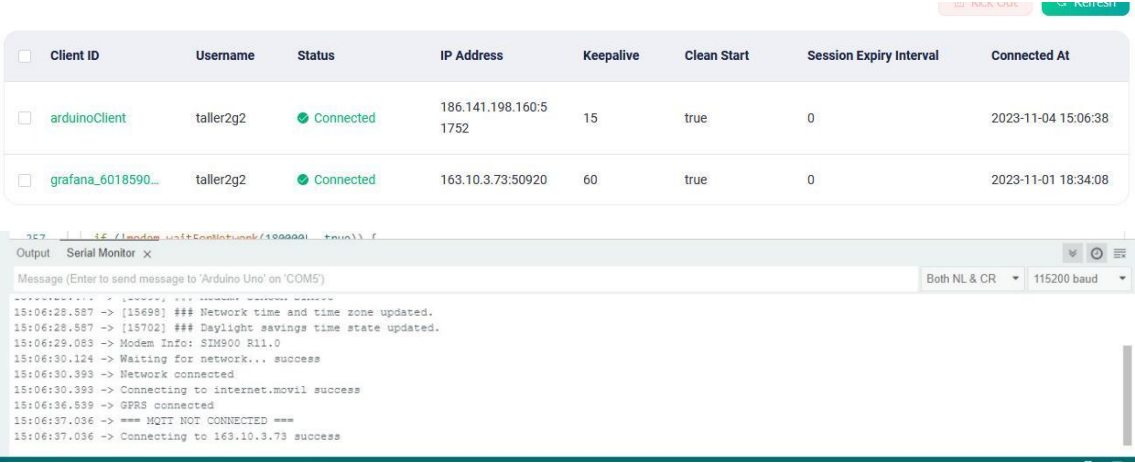


Figura 4. Prueba de conexión al bróker MQTT exitosa

En la Figura 4 se observa en la parte superior el Client ID “arduinoClient” el cual es el ID configurado en el firmware de prueba para la identificación. La parte inferior es la salida del Serial Monitor del Arduino IDE, el cual muestra la información del Modem, luego continúa informando que se realizó la conexión a la red de telefonía, al APN y al servidor de forma exitosa.

Al corroborar que la prueba de conexión fue un éxito, se agregó la funcionalidad de enviar mensajes JSON al servidor. Las primeras pruebas mostraron que no era posible establecer la conexión siquiera a la red telefónica, esto fue debido a la asignación de recursos para el JSON: el primer prototipo establece un JSON de tamaño dinámico hasta 200 bytes, con un buffer del mismo tamaño. Esto era una demanda de recursos excesiva para el Arduino Uno al considerar el algoritmo y librerías para el manejo del SIM908C.

Para solucionar este problema, se redujo tanto el objeto JSON y el buffer a 20 bytes y se intentó otra vez. El resultado fue el envío de mensajes JSON desde el cliente Arduino hasta el servidor de forma exitosa.

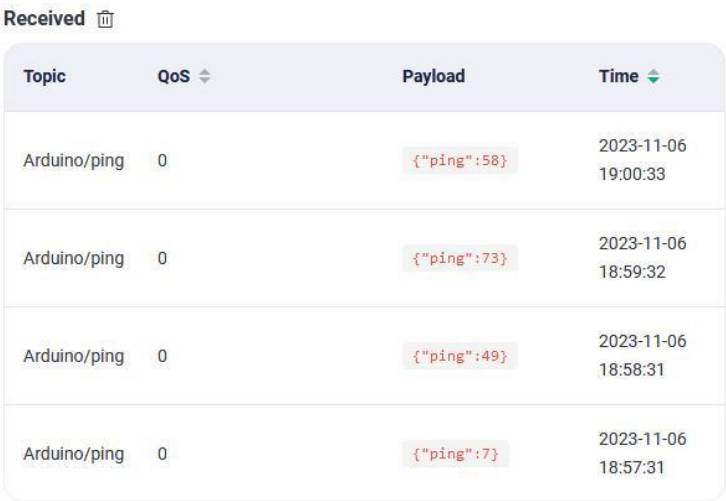


Figura 5. Prueba de envío de mensajes JSON al bróker MQTT exitosa.

En la figura 5 se observa los mensajes JSON enviados al servidor por el topic “Arduino/ping” con un tiempo de alrededor de 1 minuto entre cada mensaje, lo que se destaca de esta prueba es el payload el cual muestra {“ping”: valor} el cual “ping” indica el nombre del campo mientras que el valor indica un número al azar desde 0 a 99.

Ya con estas pruebas realizadas, se procedió a la prueba de enviar mensajes desde el sensor de temperatura al servidor MQTT.

La implementación de la parte de sensado de temperatura conllevó realizar cambios en el firmware ya que al añadir la librería para el manejo del DHT11 y el código necesario para su uso mostraba problemas durante la ejecución de las primeras pruebas debido al alto uso de memoria por parte del firmware.

Después de realizar los cambios necesarios, se realizaron pruebas de envío de medidas del sensor al servidor MQTT de forma exitosa.

Received 

Topic	QoS 	Payload	Time 
Arduino/temp	0	{“temp”:25}	17:12:01
Arduino/temp	0	{“temp”:25}	2023-11-08 17:11:01
Arduino/temp	0	{“temp”:25}	2023-11-08 17:09:57
Arduino/temp	0	{“temp”:26}	2023-11-08 17:08:57
Arduino/temp	0	{“temp”:25}	2023-11-08 17:08:00

Figura 6. Prueba de envío de medidas de temperatura al bróker MQTT exitosa.

4.2. Servidor e Interfaz Web

4.2.1. Servidor

Se configuró un equipo de escritorio como servidor para cumplir con las tareas de bróker MQTT y servidor de base de datos. Se instalaron los servicios EMQX y Mosquitto para comunicaciones con protocolo MQTT, y MySQL para gestionar una base de datos relacional. Como medida de seguridad, se configuró un firewall (UFW), en el cual deben explicitarse los puertos de conexión de los servicios para permitir la conectividad.

Se realizaron pruebas de conexión satisfactorias mediante comandos de diagnóstico (ping) hacia el servidor y se habilitaron los puertos de conexión para todos los servicios y se establecieron conexiones y realizaron pruebas de funcionamiento mediante programas en lenguaje C y Python.

Los programas de prueba en C para el servicio Mosquitto (ubicados en el directorio /mosquitto/test del repositorio del proyecto) configuran la conexión al puerto 15883 del servidor, para luego realizar pruebas. Uno de los programas (test-mosquitto-sub) se suscribe a un tópico de prueba e imprime en la consola los mensajes recibidos hasta que se presiona CTRL-C. El otro programa (test-mosquitto) se conecta al mismo puerto del servidor y publica datos de prueba (en formato JSON) a un tópico de prueba cada un cierto tiempo, los cuales deben ser recibidos por el programa suscriptor, hasta que se finaliza ingresando CTRL-C. Ambos programas hacen uso de la librería mosquitto.h.

Se ejecutaron inicialmente ambos programas en dos terminales separadas (estaciones de trabajo Linux conectadas a Internet) y se comprobó el funcionamiento de la comunicación. La figura 7 muestra que los mensajes se envían desde una terminal y se reciben en la otra.

```

constantino@EXOX352 ~/ProyectoTaller2/mosquitto/test
constantino@EXOX352 ~/ProyectoTaller2/mosquitto/test $ ./test-mosquitto
Inicializando...
Conectando...
Message with nid 1 has been published
Message with nid 2 has been published
Message with nid 3 has been published
Message with nid 4 has been published
Message with nid 5 has been published
Message with nid 6 has been published
Message with nid 7 has been published
Message with nid 8 has been published
Message with nid 9 has been published
Message with nid 10 has been published

constantino@vanessa: ~
constantino@vanessa:~$ ./test-mosquitto-sub
Inicializando...
Conectando...
on connect: Connection Accepted.
on subscribe: 0: granted qos = 0
/test/temperature 0 3
/test/temperature 0 85
/test/temperature 0 10
/test/temperature 0 32
/test/temperature 0 67
/test/temperature 0 77
/test/temperature 0 1
/test/temperature 0 0
/test/temperature 0 50
/test/temperature 0 8

```

Figura 7. Pruebas de envío (izquierda) y recepción (derecha) con Mosquitto, en C

Para la prueba del servicio EMQX se escribió un programa en Python (test-emqx-cli ubicado en el directorio /emqx) que utiliza la librería paho.mqtt. Establece una conexión al puerto 1883 del servidor y envía 100 mensajes a un tópico de prueba utilizando un formato de mensaje similar al usado en las pruebas con Mosquitto. La recepción se monitorea inicialmente desde un programa en Python (test-emqx-sub) que se conecta al puerto 1883, se suscribe al tópico de prueba e imprime en la consola los mensajes recibidos hasta que se ingresa CTRL-C. La figura 8 muestra que la prueba es exitosa.

```

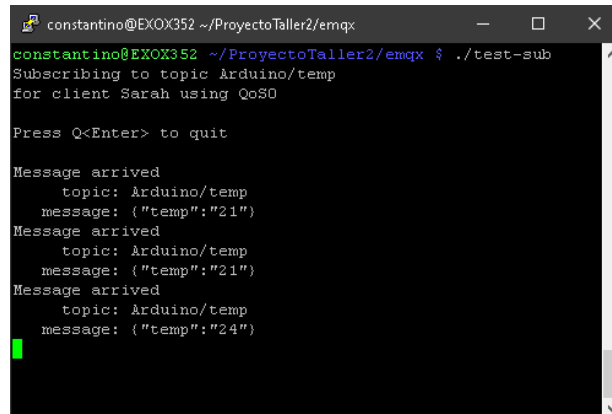
constantino@EXOX352 ~/ProyectoTaller2/emqx
constantino@EXOX352 ~/ProyectoTaller2/emqx $ python test-emqx-cli.py
Conectado al broker 163.10.3.73:1883
Enviado mensaje {"temperatura": "2", "fh_medicion": "2023-10-06 23:30:54.043352"} a topico emqx/test
Enviado mensaje {"temperatura": "44", "fh_medicion": "2023-10-06 23:30:55.046419"} a topico emqx/test
Enviado mensaje {"temperatura": "25", "fh_medicion": "2023-10-06 23:30:56.049312"} a topico emqx/test
Enviado mensaje {"temperatura": "10", "fh_medicion": "2023-10-06 23:30:57.052817"} a topico emqx/test
Enviado mensaje {"temperatura": "35", "fh_medicion": "2023-10-06 23:30:58.055654"} a topico emqx/test
Enviado mensaje {"temperatura": "2", "fh_medicion": "2023-10-06 23:30:59.059095"} a topico emqx/test
Enviado mensaje {"temperatura": "59", "fh_medicion": "2023-10-06 23:31:00.062796"} a topico emqx/test
Enviado mensaje {"temperatura": "38", "fh_medicion": "2023-10-06 23:31:01.065748"} a topico emqx/test
Enviado mensaje {"temperatura": "20", "fh_medicion": "2023-10-06 23:31:02.068543"} a topico emqx/test
Enviado mensaje {"temperatura": "59", "fh_medicion": "2023-10-06 23:31:03.071376"} a topico emqx/test

constantino@vanessa: ~
constantino@vanessa:~$ python3 test-emqx-sub.py
Conectado al broker 163.10.3.73:1883
emqx/test b'{"temperatura": "2", "fh_medicion": "2023-10-06 23:30:54.043352"}'
emqx/test b'{"temperatura": "44", "fh_medicion": "2023-10-06 23:30:55.046419"}'
emqx/test b'{"temperatura": "25", "fh_medicion": "2023-10-06 23:30:56.049312"}'
emqx/test b'{"temperatura": "10", "fh_medicion": "2023-10-06 23:30:57.052817"}'
emqx/test b'{"temperatura": "35", "fh_medicion": "2023-10-06 23:30:58.055654"}'
emqx/test b'{"temperatura": "2", "fh_medicion": "2023-10-06 23:30:59.059095"}'
emqx/test b'{"temperatura": "59", "fh_medicion": "2023-10-06 23:31:00.062796"}'
emqx/test b'{"temperatura": "38", "fh_medicion": "2023-10-06 23:31:01.065748"}'
emqx/test b'{"temperatura": "20", "fh_medicion": "2023-10-06 23:31:02.068543"}'
emqx/test b'{"temperatura": "59", "fh_medicion": "2023-10-06 23:31:03.071376"}'

```

Figura 8. Envío (izquierda) y recepción (derecha) con EMQX, en Python

Otra prueba fue hecha mediante un programa escrito en C (/emqx/test-sub.c), el cual utiliza la librería `paho-mqtt`. Este programa se conecta al servidor y se suscribe al tópico `Arduino/temp` e imprime en la terminal los valores recibidos.



```

constantino@EXOX352 ~/ProyectoTaller2/emqx
constantino@EXOX352 ~/ProyectoTaller2/emqx $ ./test-sub
Subscribing to topic Arduino/temp
for client Sarah using QoS0

Press Q<Enter> to quit

Message arrived
  topic: Arduino/temp
  message: {"temp": "21"}
Message arrived
  topic: Arduino/temp
  message: {"temp": "21"}
Message arrived
  topic: Arduino/temp
  message: {"temp": "24"}

```

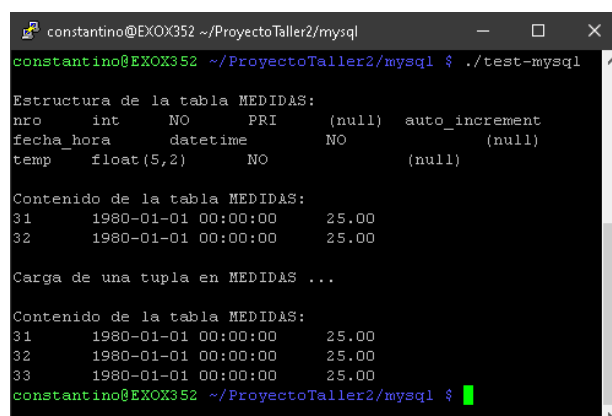
Figura 9. Prueba exitosa de recepción de mensajes de datos

Para almacenar los datos relativos a este proyecto se creó una base de datos compuesta de una única tabla (medidas), con los siguientes atributos:

1. Número de medida: es un número de secuencia único para cada medida registrada. Es un campo auto-incremental que no debe ser nulo.
2. Fecha/Hora: es una marca temporal agregada en la recepción. Es la fecha y hora de recepción de los datos. Es un campo de formato datetime.
3. Temperatura: son las mediciones de temperatura realizadas. Es un campo numérico.

Para probar el funcionamiento, se escribió un programa en C que establecería una conexión y realizaría una serie de consultas:

1. Describir la estructura de la tabla “medidas”
2. Listar todas las tuplas de la tabla
3. Insertar una tupla con datos de ejemplo
4. Volver a listar todas las tuplas para verificar la correcta inserción.



```

constantino@EXOX352 ~/ProyectoTaller2/mysql
constantino@EXOX352 ~/ProyectoTaller2/mysql $ ./test-mysql

Estructura de la tabla MEDIDAS:
nro      int      NO      PRI      (null)  auto_increment
fecha_hora datetime NO      NO      (null)
temp     float(5,2) NO      NO      (null)

Contenido de la tabla MEDIDAS:
31      1980-01-01 00:00:00      25.00
32      1980-01-01 00:00:00      25.00

Carga de una tupla en MEDIDAS ...

Contenido de la tabla MEDIDAS:
31      1980-01-01 00:00:00      25.00
32      1980-01-01 00:00:00      25.00
33      1980-01-01 00:00:00      25.00
constantino@EXOX352 ~/ProyectoTaller2/mysql $

```

Figura 10. Resultados de la prueba del servidor SQL

4.4. Aplicación Web temperatura

Para poder visualizar el valor actual de temperatura se optó por desarrollar una aplicación React ignorando la idea principal de hacerlo con Grafana, se tomó esta decisión debido a la complejidad de Grafana y a la facilidad que ofrece React para implementar la solución.

Nos basamos en un ejemplo de la comunidad que utiliza la librería MQTT desde el front para poder suscribirse al tópico “**Arduino/temp**”, luego para mostrar la información recibida desde el broker utilizamos la librería “**react-thermometer-component**”.

Se realizaron pruebas en tiempo real, enviando mensajes al tópico mencionado y se puede comprobar que el refresco del valor del termómetro se realiza con éxito.

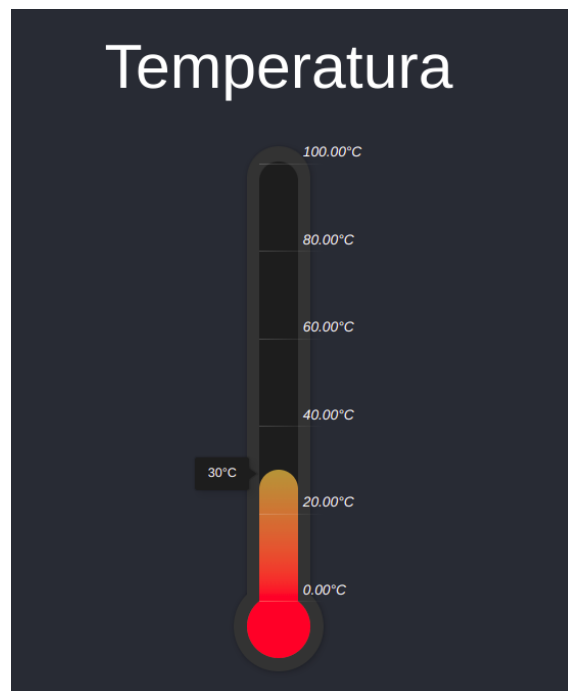


Figura 11. Aplicación Web en donde muestra la temperatura del arduino en tiempo real.

4.5. Problemas más importantes y soluciones

- En la aplicación Web de React se encontraron dificultades para encontrar la librería específica a utilizar, debido a que existe una gran oferta de librerías que no necesariamente son compatibles entre sí. Se optó por utilizar la versión v14.21.3 de Node, la cual es compatible con todas las librerías utilizadas.

5. Documentación Relacionada

- El repositorio de GitHub del proyecto es accesible desde el siguiente enlace web. Es de acceso público y contiene tanto el código fuente de los módulos de hardware/software del proyecto como la bitácora e instrucciones de configuración para los servicios web en el servidor:
<https://github.com/tpII/2023-G2-MQTT-GRPS.git>
- La bitácora está almacenada dentro del repositorio de GitHub del proyecto, en un directorio del mismo nombre y en formato markdown para su correcta visualización en un navegador
- Los videos de las pruebas de funcionamiento están en formato (.mp4) se suben al repositorio del trabajo (directorio /docs)

Sitios web consultados para armado de presupuesto:

- Mercadolibre. Arduino Uno Ch340 C/cable Usb Compatible (2023). Disponible en internet: https://articulo.mercadolibre.com.ar/MLA-925594160-arduino-uno-ch340-ccable-usb-compatible-JM#position=1&search_layout=grid&type=item&tracking_id=ca13e077-19a1-4eae-bea0-c149b530f095 Consultado el 5/10/2023.
- Mercadolibre. Shield Arduino Gsm Gprs Gps Bluetooth Con Módulo Sim808 (2023). Disponible en internet: https://articulo.mercadolibre.com.ar/MLA-627307217-shield-arduino-gsm-gprs-gps-bluetooth-con-modulo-sim808-JM#position=5&search_layout=grid&type=item&tracking_id=436fc6ec-2ab3-49a4-a422-764096a882f7 Consultado el 5/10/2023.
- Mercadolibre. Sensor De Temperatura Y Humedad Relativa Dht11 Arduino (2023). Disponible en internet: https://articulo.mercadolibre.com.ar/MLA-773164135-sensor-de-temperatura-y-humedad-relativa-dht11-arduino-JM#position=1&search_layout=grid&type=item&tracking_id=fbf3a1ac-242c-4ef7-8b03-582d76927350 Consultado el 5/10/2023.
- Ejemplo aplicación suscripción con Broker MQTT usando React. <https://www.emqx.com/en/blog/how-to-use-mqtt-in-react>
- Librería react para mostrar temperatura: (<https://zamarrowski.github.io/react-thermometer-component/>)