

Taller de Proyecto II

Informe Final

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

1. Descripción General del Proyecto	3
1.1. Introducción	3
1.2. Objetivos Principales.....	3
1.3. Objetivos Secundarios.....	3
1.3.1. Adición de módulos controlados por el sistema.....	3
1.3.2. Adición de módulo RTC	4
1.3.3. Adición de módulo de alimentación por batería.....	4
2. Presentación Esquemática del Proyecto	5
2.1. Gráfico de procesos y funciones del proyecto.....	5
2.2. Esquemático de conexiones de hardware.....	5
3. Documentación del Software del Proyecto	7
3.1. Firmware del Sistema.....	7
3.1.1. Documentación del Firmware	8
3.2. Servidor y Programa Backend.....	8
3.2.1. Servidor	8
3.2.1.1. Configuración bróker EMQX.....	9
3.2.1.2. Configuración de MySQL	10
3.2.2. Programa Backend	11
3.2.2.1. Ejecución Local.....	13
3.2.2.2. Archivos del Backend	14
3.3. Aplicación Web Temperatura (Frontend)	14
3.3.1. Pseudocódigo	15
3.3.2. Guía de Uso.....	16
4. Otra Documentación Relacionada.....	17
4.1. Recursos Web para Configuración del Servidor.....	17
4.2. Guía de instalación: Entorno Arduino.....	17
4.3. Guía de uso: SIM908C.....	18
4.4. Repositorio del Proyecto	19
Apéndice A: Materiales y Presupuesto	20

1. Descripción General del Proyecto

1.1. 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.

1.2. Objetivos Principales

Originalmente se propuso como objetivo principal del proyecto la realización de 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 sensados.
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.

Respecto al desarrollo de este objetivo, no hubo cambios significativos en el sistema propuesto y se completó el desarrollo.

1.3. Objetivos Secundarios

Al inicio del desarrollo, se propusieron algunos objetivos secundarios como mejoras o ampliaciones al cumplir los objetivos principales. A continuación se nombran las mejoras propuestas junto con el avance realizado por el grupo:

1.3.1. 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.

Inicialmente se propuso la inclusión de dispositivos que reflejen el acondicionamiento de la temperatura cuando la misma se encuentra fuera de rango. Ejemplos:

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

Debido al tiempo que requeriría la implementación de dispositivos como un ventilador, se decidió hacer uso del LED interno del Arduino UNO para responder a comandos enviados desde un cliente conectado al servidor MQTT. De esta forma se puede realizar una comunicación desde un cliente en cualquier lugar hacia el sistema.

1.3.2. 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.

No se implementó esta mejora debido a que no se obtuvo un RTC a tiempo para implementar la funcionalidad.

1.3.3. 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.

No se implementó esta mejora debido a que no se consideró necesaria para este prototipo.

2. Presentación Esquemática del Proyecto

2.1. Gráfico de procesos y funciones del proyecto.

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

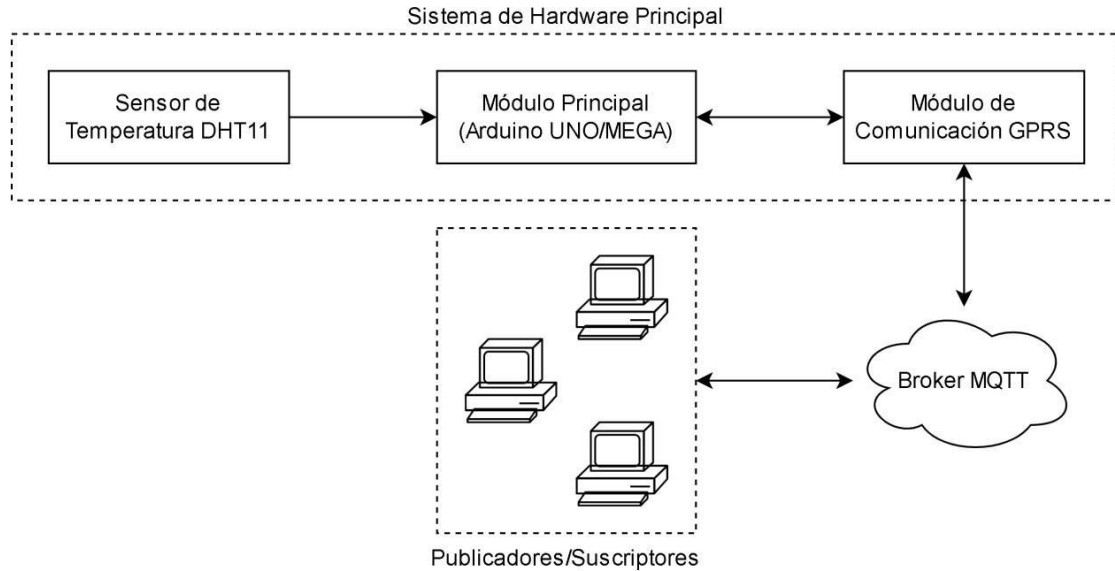


Figura 1. Diagrama de bloques del sistema

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. Esquemático de conexiones de hardware

Se realizaron las conexiones de todos los componentes electrónicos que conforman el sistema según lo especificado en la siguiente tabla:

Componente	Pin Componente	Pin Arduino
DHT11	VCC	5V
	GND	GND
	Data	5
SIM908C	VCC	-
	GND	GND
	GPRSTXD	2
	GPRSRXD	3

Tabla 1. Conexiones entre la placa Arduino UNO y los demás componentes

La alimentación del SIM908C se realiza mediante una fuente de 9V/2A. A continuación, la figura 2 muestra el esquema de conexiones mientras que la figura 3 presenta el conexionado físico del sistema.

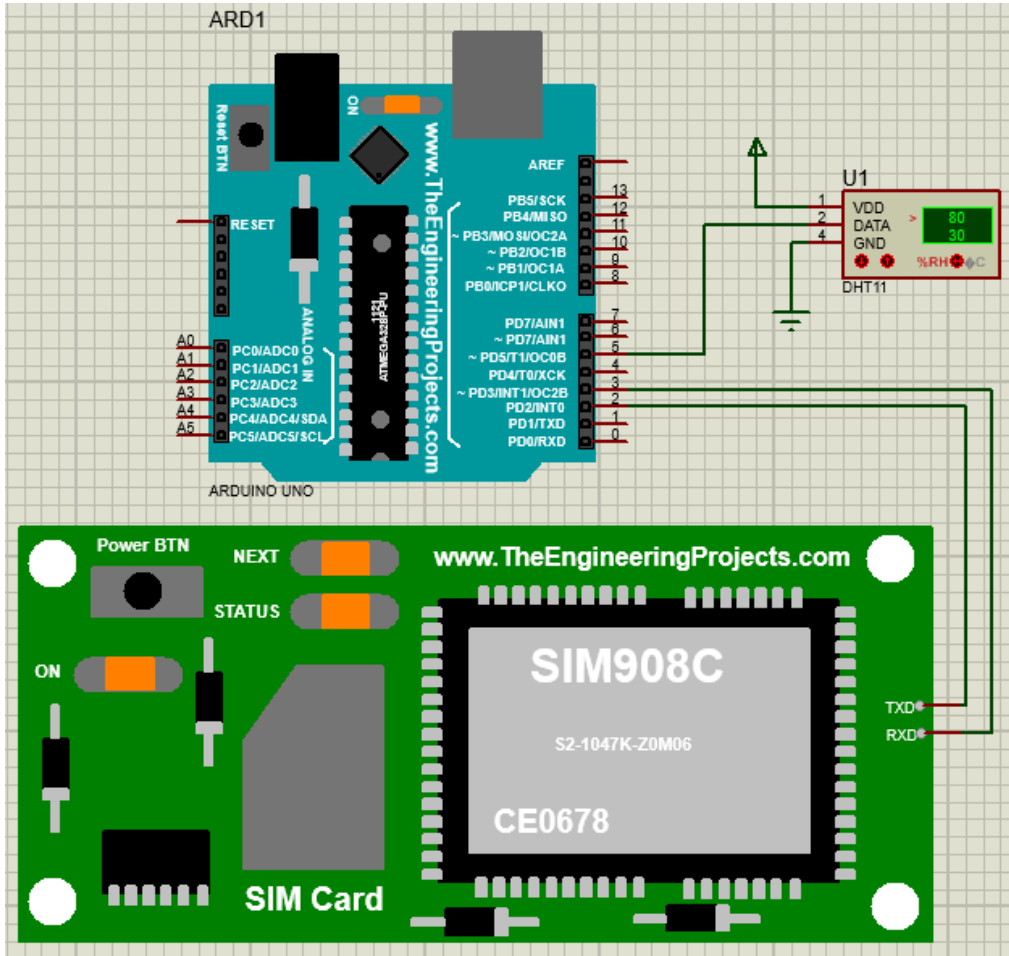


Figura 2. Diagrama de conexiones del sistema.

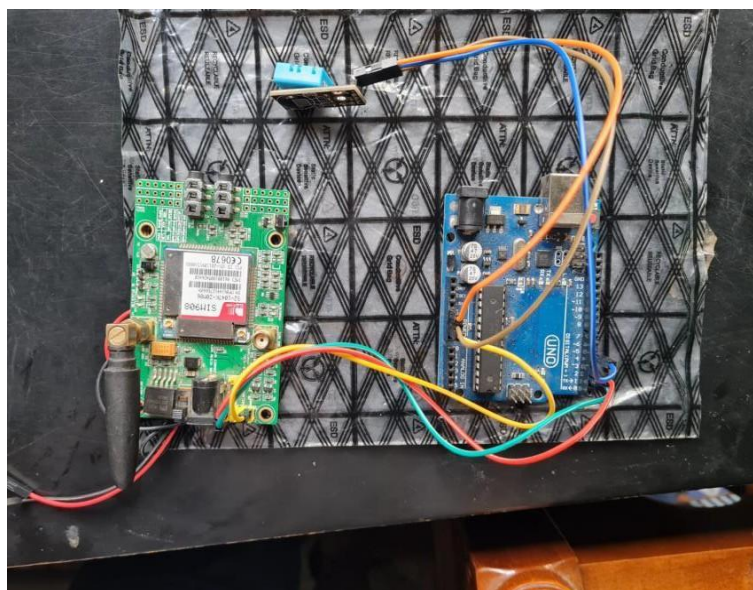


Figura 3. Esquema de conexiones físicas del sistema.

En la figura 3 podemos observar los 3 componentes: a la izquierda inferior se encuentra el SIM908C, a su derecha el Arduino UNO y en la parte superior el sensor de temperatura DHT11.

3. Documentación del Software del Proyecto

Los siguientes componentes forman el sistema de software:

3.1. Firmware del Sistema

Para la programación del firmware se utilizaron las siguientes herramientas, las cuales están adjuntas en este informe tanto el código fuente con sus respectivas librerías en la carpeta /arduinoClient:

- Arduino IDE
- Lenguaje C
- Librerías de Arduino para el manejo de los componentes conectados:
 - TinyGsm: Librería para el manejo de dispositivos GSM como el SIM908C.
 - PubSubClient: Librería para soporte de comunicación MQTT.
 - DHT: Librería para el manejo del sensor de temperatura DHT11
 - SoftwareSerial: Librería para habilitar la comunicación serial en pines digitales además de los designados por defecto por Arduino (pin 0 y 1 para el modelo Arduino UNO)
 - ArduinoJSON: Para el envío de datos en formato JSON.

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
    Configurar el pin correspondiente al LED interno de Arduino como
        salida
    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
    Configurar la función a llamar en caso de recibir mensajes por
        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 cumpliero 10 segundos 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.
}
```

El firmware utiliza dos canales para la comunicación MQTT:

- **Arduino/temp:** En este canal el firmware publica las medidas de temperatura.
- **Arduino/control:** El firmware está suscrito en este canal para recibir mensajes de control por parte de un usuario.

Cuando se recibe un mensaje por el canal de control, el firmware procesa el mensaje según el pseudocódigo a continuación, por medio de la función `mqttCallback`:

```
void mqttCallback () {  
    Imprime el mensaje recibido en el terminal de Arduino.  
    Alterna el estado del led para simular la parte de control.  
}
```

3.1.1. Documentación del Firmware

El código desarrollado para la placa Arduino está implementado en entorno Arduino IDE diferentes librerías disponibles para la gestión de cada uno de los módulos integrados en el archivo `arduinoClient.ino`. En este archivo se encuentran las siguientes variables de configuración:

- **GSM_PIN:** Las tarjetas SIM tienen una clave PIN asignada que puede usarse para desbloquear la tarjeta en caso de bloquearse.
- **apn[]:** APN de la red telefónica.
- **gprsUser[]:** Usuario para conectarse a la APN.
- **gprsPass[]:** Contraseña para conectarse a la APN.
- **broker:** Dirección del servidor MQTT. Puede ser tanto la IP como una dirección URL.
- **topicTemp:** Nombre del canal donde el firmware enviará los censados de temperatura.
- **DHTPIN:** Variable de configuración de la librería DHT para identificar a qué pin está conectado el pin DATA
- **DHTTYPE:** Variable de configuración de la librería DHT para identificar el modelo de sensor de temperatura.

El equipo desarrolló las siguientes funciones:

- `mqttConnect()`: se conecta al servidor MQTT especificado en la variable `broker` por con las credenciales especificadas para este proyecto.
- `mqttCallback(char* topic, byte* payload, unsigned int len)`: Maneja los mensajes MQTT recibidos por la placa Arduino.

3.2. Servidor y Programa Backend

3.2.1. Servidor

El sistema de hardware desarrollado (Arduino + GPRS) se apoya en un servidor físico para las funciones de comunicación y almacenamiento de los datos recibidos. Asimismo el frontend se apoya en el mismo servidor para las funciones de monitoreo de temperatura y consulta de lecturas históricas.

El servidor es un equipo de escritorio estándar con el sistema operativo Ubuntu Server 20.04 y los siguientes servicios instalados:

1. **EMQX:** provee funcionalidad de bróker MQTT para comunicaciones con el sistema de hardware
2. **MySQL:** provee la estructura y manejo de una base de datos relacional para almacenar

- los datos leídos por el hardware y transmitidos al servidor
3. Programa backend: componente de software escrito en lenguaje Python encargado de tomar las lecturas del hardware e insertarlas en la base de datos del proyecto.

Cabe destacar que el servidor no necesariamente debe ser un equipo aparte, podría configurarse una estación de trabajo Linux con los servicios necesarios o incluso una máquina virtual. Estas configuraciones no se discutirán en detalle en el presente informe.

A continuación se detallan las configuraciones realizadas para cada uno de los componentes mencionados anteriormente.

3.2.1.1. Configuración bróker EMQX

Para instalar el bróker EMQX se siguieron los pasos de [1]. Tener en cuenta que debe generarse, de ser necesario, una excepción en el firewall del sistema para permitir las conexiones hacia el bróker. En el caso del server usado, basta con indicarle los puertos al servicio UFW:

```
$ sudo ufw add 1883
$ sudo ufw add 18083
```

El puerto 1883 es para la transmisión de datos en sí, mientras que 18083 es para la gestión del bróker mediante el dashboard EMQX.

Una vez activo el servicio, desde el dashboard de EMQX se creó un usuario para su uso con el proyecto. Las credenciales de acceso elegidas para el proyecto son {taller2g2, taller2g2}.

Utilizar el código de prueba en Python suministrado en el repositorio del proyecto para probar que el servicio funciona correctamente:

```
$ python test-emqx-cli.py
$ python test-emqx-sub.py
```

El primer programa es un publicador y el segundo un suscriptor. Para ejecutar ambos programas se necesitan la librería `paho.mqtt`. Se la puede instalar ejecutando el siguiente comando:

```
$ pip install paho-mqtt
```

Una vez instalada, probar con dos terminales separadas y verificar que la salida es como las ilustradas en la figura 4. Se obtuvieron al ejecutar los programas en dos equipos diferentes.

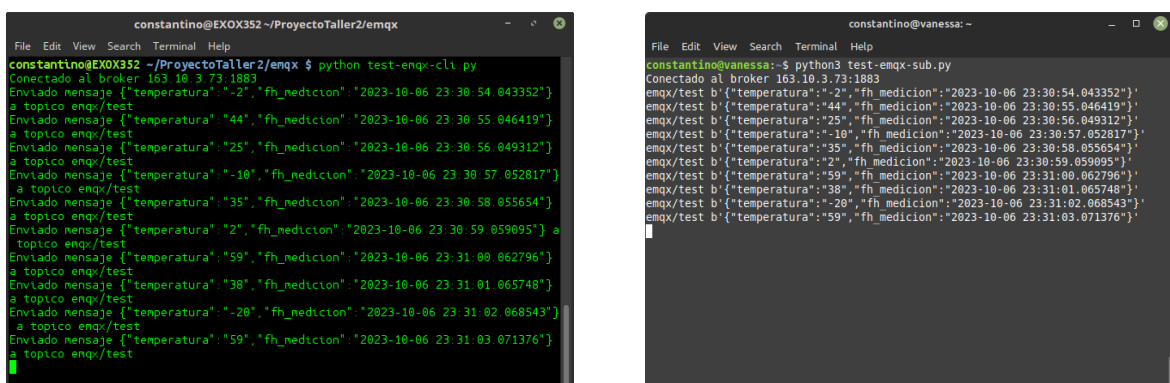


Figura 4. Resultados de los programas publicador (izquierda) y suscriptor (derecha)

Se puede también probar el funcionamiento del bróker ejecutando el programa publicador y utilizando el cliente WebSocket provisto por EMQX en el dashboard. Completar los campos con las credenciales de acceso del proyecto y realizar la conexión. Los mensajes enviados por el programa deberán aparecer en el dashboard. De no poder realizarse la conexión, verificar que los puertos requeridos por EMQX están entre los permitidos por el firewall y que el servicio en sí está activo.

3.2.1.2. Configuración de MySQL

Para la instalación del servicio de gestión de bases de datos MySQL se siguieron las instrucciones detalladas en [2]. De igual forma que para el bróker EMQX, deben explicitarse los puertos utilizados entre las excepciones del firewall (si corresponde). En el caso particular del servidor armado se ejecutaron los siguientes comandos:

```
$ sudo ufw add 3306/tcp
$ sudo ufw add 33060/tcp
```

Además debe permitirse la conexión desde cualquier dirección al servicio de bases de datos a través del archivo de configuración `/etc/mysql/my.cnf`:

```
[mysqld]
bind-address = 0.0.0.0
```

También modificar `/etc/mysql/mysql.cnf` para reflejar lo anterior. Esta configuración no es ideal desde un punto de vista de seguridad, ya que deja expuesta la base de datos a cualquier conexión. Lo ideal sería que las conexiones sólo sean permitidas desde la red local. Se tomó esta decisión para solucionar problemas de acceso desde el programa backend. Luego de aplicar las configuraciones, reiniciar el servicio con:

```
$ sudo systemctl restart mysql.service
```

Desde la terminal MySQL (ejecutada como superusuario), debe crearse un usuario para acceder a la base de datos del proyecto. Las credenciales usadas son las mismas que en EMQX por motivos de simplicidad:

```
$ sudo mysql
> create user 'taller2g2'@'%' identified by 'taller2g2';
```

El operador `%` indica que se permite que el usuario se conecte desde cualquier host. La base de datos se compone de una tabla que almacena las medidas de temperatura enviadas por el sistema de hardware a través de la red GPRS. Se sigue con el siguiente formato:

<i>Campo</i>	<i>Descripción</i>	<i>Tipo</i>
nro	Número/ID de muestra (autoincremental)	entero
fecha_hora	Marca de tiempo de la recepción de la medida	datetime
temp	Temperatura medida	float(5,2)

Tabla 2. Especificación de la tabla `medidas`.

Los siguientes comandos SQL crean la estructura de la tabla y dan control de la misma al usuario recién creado:

```

> create database taller2;

> grant all on taller2.* to 'taller2g2'@'%';

> use taller2;
> create table medidas (nro int not null auto_increment,
-> fecha_hora datetime not null, temp float(5,2) not null);

```

El primer comando crea la base de datos vacía bajo el nombre de `taller2`. El segundo le da control total de todas las tablas de la misma al usuario creado anteriormente. Los demás comandos cambian el control de la terminal a la base de datos creada y crean la tabla `medidas` ajustándose a la especificación de la tabla 2. Para mayor detalle en los comandos de MySQL, consultar [3].

Se puede ahora realizar una prueba mediante el programa de prueba suministrado en el repositorio del proyecto. Al estar escrito en lenguaje C, debe compilarse y contener la librería MySQL-Connector.

Para instalar esta librería, ejecutar el siguiente comando:

```
$ sudo apt-get install libmysql-dev
```

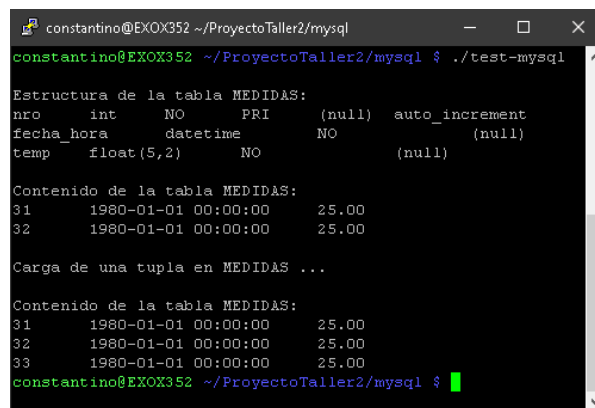
Una vez instalada la librería, compilar y ejecutar el programa de prueba:

```

$ gcc test-mysql.c -o test-mysql -lmysqlclient
$ ./test-mysql

```

Se debe obtener como salida un resultado similar al de la siguiente figura:



```

constantino@EXO352 ~/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      (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@EXO352 ~/ProyectoTaller2/mysql $

```

Figura 5. Resultados de la prueba de MySQL

Tener en consideración que el campo identificador de cada muestra (`nro`) es generado automáticamente y su valor se mantiene aunque se eliminen todas las tuplas de la tabla.

3.2.2. Programa Backend

El programa backend es un código escrito en lenguaje Python encargado de recibir los mensajes enviados por el hardware y agregar los datos recibidos a la base de datos del proyecto. En su esencia, se compone de varias funciones que realizar varias tareas en distintos momentos de la conexión.

Este programa se ejecuta directamente en el servidor, por lo que se creó un usuario nuevo en el sistema operativo del servidor. Este no es un requerimiento estricto del proyecto, pero mantiene separados los archivos del proyecto para un mejor control del mismo.

Al establecerse la conexión al bróker MQTT se ejecuta la función `on_connect`, que se suscribe al tópic `Arduino/temp` para recibir los datos. Esta función se ejecuta automáticamente, no es necesario invocarla desde ningún punto del programa. En forma de pseudocódigo:

```
función on_connect {  
    Asentar conexión en la bitácora  
    Suscribirse al tópic Arduino/temp  
}
```

Otra función ejecutada automáticamente es `on_message`. Esta función se ejecuta cada vez que se recibe un mensaje en el tópic al que está suscrito el programa. Su tarea es extraer el dato del mensaje recibido e insertarlo en la tabla `medidas` de la base de datos, agregando un campo de marca temporal con la fecha y hora actuales (fecha y hora de recepción del mensaje, a efectos del programa). Esta operación puede realizarse exitosamente o fallar. El programa backend informa por la terminal en ambos casos. En pseudocódigo:

```
función on_message {  
    Tomar datos recibidos del mensaje  
    Intentar {  
        Insertar datos en la BD e informar  
    }  
    En caso de excepción {  
        Informar error  
        Cerrar conexión a BD  
    }  
}
```

Adicionalmente, se incluye un script de Linux para invocar al backend con las librerías requeridas, aunque puede ser ejecutado manualmente mediante el siguiente comando:

```
$ python -m backend.py
```

Una vez funcionando el backend, itera indefinidamente ejecutando sus funciones. Al conectarse con el hardware produce una salida como la de la siguiente figura.

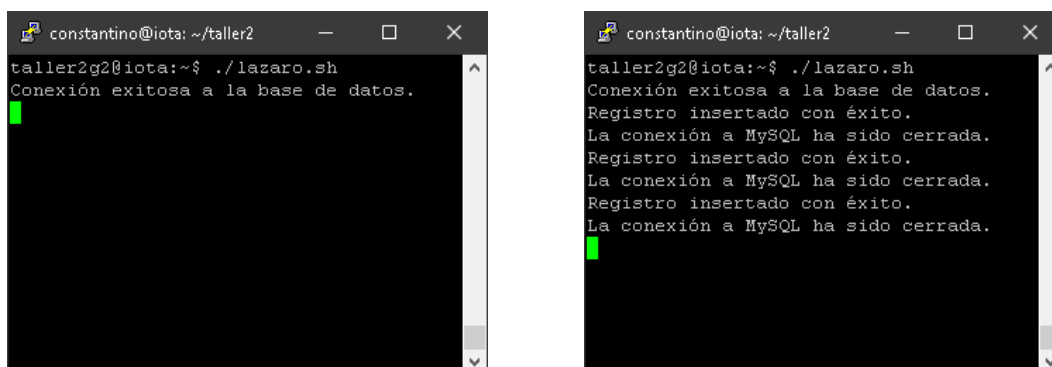


Figura 6. Programa backend en funcionamiento: conexión (izquierda) y recepción (derecha)

Es importante tener en cuenta que el backend seguirá activo aún ante fallos en la conexión. El único medio para terminar su ejecución es ingresar CTRL+C.

El programa backend adicionalmente está configurado para llevar una bitácora (log) de sus acciones y errores encontrados. Su ubicación es `/logs`, dentro del directorio principal del usuario Linux.

3.2.2.1. Ejecución Local

El archivo `/backend/backend.py` es una implementación alternativa del programa backend, pensada para su ejecución local en lugar de ejecutarse sobre el servidor. El programa es similar, aunque con una funcionalidad adicional descrita a continuación.

Se utiliza la librería de Python Flask (un micro framework de Python) para exponer un API REST. Haciendo uso de esta librería se expone el endpoint `"/temperatura"`, que permite recuperar los datos históricos de temperatura almacenados, retornando en formato JSON para la aplicaciones cliente. En forma de pseudocódigo:

```
función obtener_temperaturas {
    Obtener credenciales de BD
    Establecer conexión a BD
    Obtener información de BD
    Convertir resultado en formato JSON
    Si (hay error) {
        Registrar error
    } sino {
        Cerrar conexión
    }
}
```

Para la ejecución local se requieren las siguientes librerías de Python:

- `paho-mqtt`
- `mysql-connector`
- `Flask`
- `Flask-CORS`

Todas ellas pueden ser instaladas mediante el comando `pip`:

```
$ pip install <nombre_librería>
```

Para poder ejecutar el backend localmente se invoca el siguiente comando desde una terminal:

```
$ python backend.py
```

Se espera un resultado como el siguiente:

```
* Serving Flask app 'backend'
```

3.2.2.2. Archivos del Backend

Los siguientes archivos dentro del repositorio del proyecto permiten probar y poner en funcionamiento el programa backend. Se utiliza la notación de directorios de UNIX y se asume como punto de partida (/) el directorio principal del repositorio del proyecto:

/backend/backend.py	Programa backend de ejecución local
/backend/sv/backend.py	Programa backend para servidor
/backend/sv/lazaro.sh	Script lanzador de backend de servidor
/emqx/test-emqx-cli.py	Programa de prueba EMQX "publicador"
/emqx/test-emqx-sub.py	Programa de prueba EMQX "subscriber"
/mysql/test-mysql.c	Programa de prueba MySQL

Tabla 3. Archivos mencionados en esta sección

3.3. Aplicación Web Temperatura (Frontend)

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.

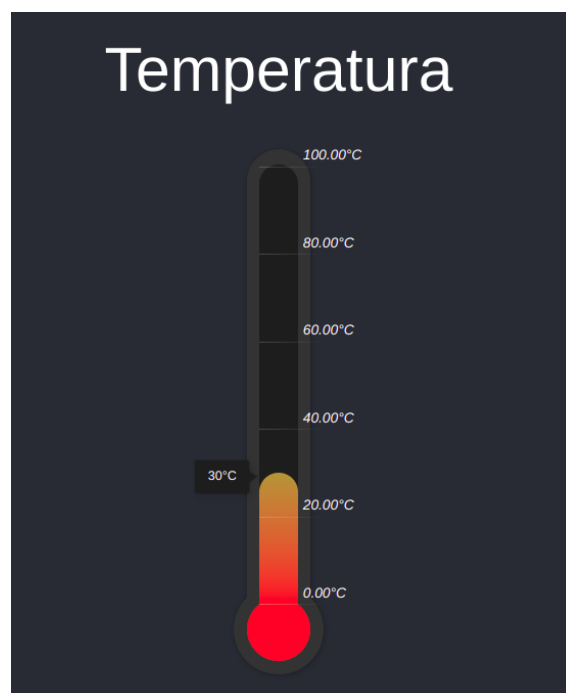


Figura 7. Aplicación Web en donde muestra la temperatura del arduino en tiempo real.

Se realizó la adición de un gráfico de datos históricos de temperatura (ver Figura 8). Este gráfico se implementó utilizando la biblioteca "Recharts", reconocida por su facilidad de uso y

versatilidad para la representación de distintos tipos de datos gráficos. En particular, optamos por el componente "LineChart" de Recharts para visualizar la evolución de la temperatura a lo largo del tiempo. La información necesaria para alimentar este gráfico se obtiene directamente desde un endpoint específico en nuestro backend, diseñado para proveer estos datos históricos en un formato adecuado para su visualización. Esta integración asegura que el gráfico siempre refleje los datos más recientes y completos disponibles.

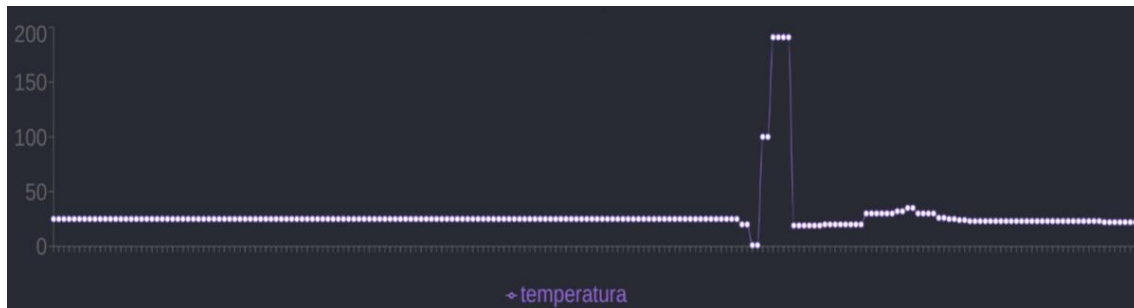


Figura 8. Gráfico XY de temperatura

3.3.1. Pseudocódigo

1. Establecer Conexión con el Broker MQTT:

Para poder conectar el broker MQTT primero se obtienen las credenciales necesarias para establecer la conexión como: protocolo, host, clientId, puerto, usuario y contraseña. Luego se procede a conectar al broker MQTT y actualizar el estado de conexión.

```
función definir configuración {
    Obtener protocolo, host, cliente, etc.
}
```

2. Manejo de Eventos de Cliente MQTT:

Cada vez que se produce un evento en el objeto cliente, se ejecuta una acción:

```
función establecer conexión {
    Siempre que se produzca un cambio el "cliente" {
        Si (se realizó la conexión exitosamente) {
            Actualizar estado de conexión a "connected"
            Informar por consola
        }
        Si (hubo error) {
            Manejar e informar errores
        }
        Si (se reconecta) {
            Actualizar estado de conexión a "reconnecting"
        }
        Si (se recibe un mensaje) {
            Actualizar el estado payload con el mensaje
            Informar por consola
        }
    }
}
```

3. Suscribirse a Tópicos MQTT:

Utiliza el cliente MQTT para suscribirse a un tópico, actualizar estado de suscripción una vez suscrito al tópico.

4. Renderizar Componente:

Renderizar el componente Receiver pasando el mensaje transformado en un json como parámetro, para así poder mostrar el componente.

3.3.2. Guía de Uso

- **Prerrequisitos**
 - [Node LTS](#) version v14.21.3 (recomendado usar nvm)
- **Dependencias**
 - npm install (para instalar las dependencias)
- **Ejecutar el proyecto**
 - Levantar primero el backend.
 - Ejecutar el comando en el directorio raíz del proyecto
 - **npm run start**

4. Otra Documentación Relacionada

4.1. Recursos Web para Configuración del Servidor

Los siguientes recursos fueron consultados durante el proceso de configuración del servidor. Se verificó que todas las direcciones web fueran válidas al 03/12/2023. Estas fuentes se citan en la sección 3.2 utilizando el número de referencia entre corchetes.

- [1] *Install EMQX on Ubuntu*: <https://www.emqx.io/docs/en/latest/deploy/install-ubuntu.html>.
- [2] *Install and Configure a MySQL Server*: <https://ubuntu.com/server/docs/databases-mysql>
- [3] *MySQL Tutorial*: <https://www.javatpoint.com/mysql-tutorial>

4.2. Guía de instalación: Entorno Arduino

El firmware fue desarrollado y compilado en el entorno Arduino IDE, disponible para su descarga en: <https://www.arduino.cc/en/main/software>.

Una vez descargado el IDE de Arduino, se deben descargar las librerías utilizadas para gestionar los módulos. Para ello, en el menú “Herramientas” seleccionar la opción “Gestionar bibliotecas...” y en dicha ventana buscar e instalar cada una de las librerías listadas a continuación. Las librerías que no son descargables desde el IDE, se agrega el link de descarga:

- TinyGsm
- PubSubClient
- DHT sensor library
- ArduinoJSON

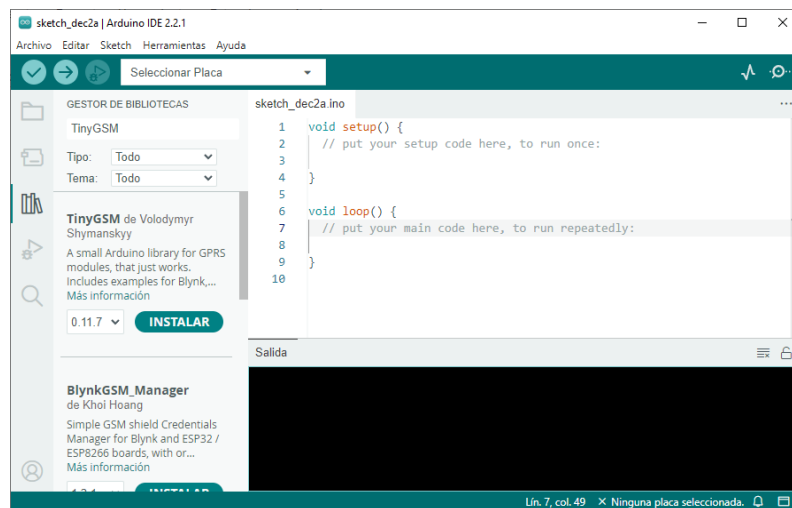


Figura 9. Ejemplo de instalación de librerías en Arduino

En la figura 9 se muestra a modo ejemplo la búsqueda de la librería “TinyGSM”, para instalar la librería hay que presionar el botón “Instalar” que permitirá agregarla al entorno. Una vez instaladas las librerías, para poder compilar el código se debe completar los datos de la configuración como la dirección del broker MQTT y los datos de la APN de la tarjeta SIM colocada en el SIM908C y presionar el botón “Cargar” en el Arduino IDE.

4.3. Guía de uso: SIM908C

En caso de desear probar el funcionamiento del módulo SIM908C, la librería TinyGSM ofrece ejemplos que puede probar accediendo al menú “Archivo”, seleccionar “Ejemplos”, luego en “Ejemplos de bibliotecas personalizadas” seleccionar “TinyGSM”. Cabe destacar que estos ejemplos pueden ser complejos para alguien sin experiencia con el uso de placas GSM por lo que una alternativa para comprobar el funcionamiento es utilizar un código de pruebas desarrollado por el grupo llamado `test-sim8001.ino` que realiza pruebas utilizando comandos AT utilizados frecuentemente. Simplemente hay que descargar el archivo *.ino y cargarlo a la placa utilizando el botón “Cargar”.

Cada vez que se enciende la placa, es necesario presionar un botón por un segundo o dos para iniciarla por completo, de otra forma no se conectará a la red móvil.

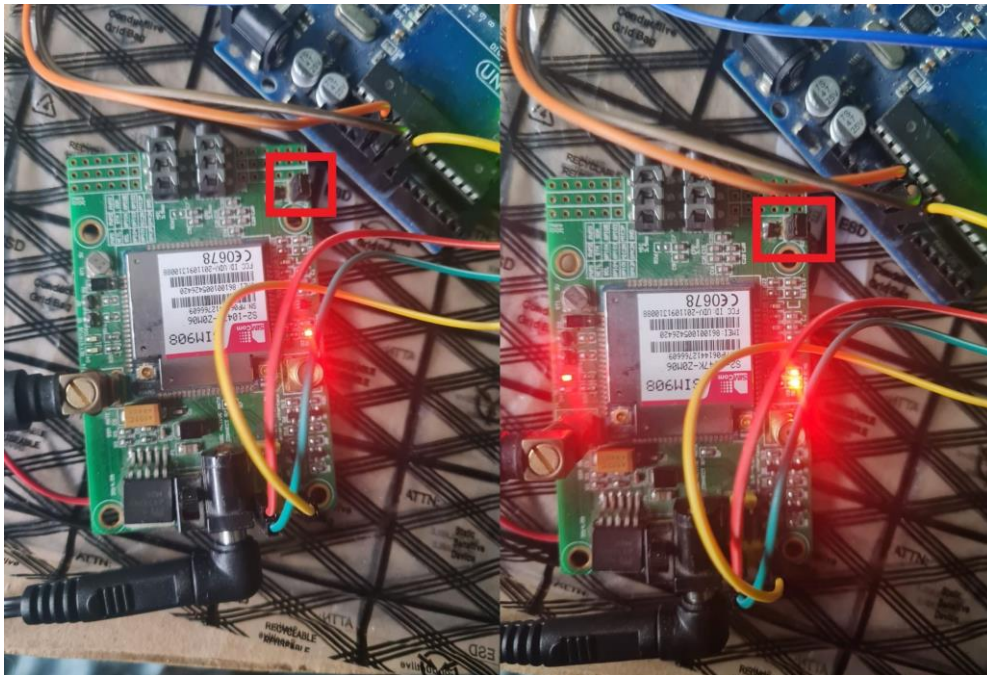


Figura 10. Estado de la placa al realizar el encendido correctamente.

En la figura anterior se observa la placa antes y después de presionar dicho botón. La parte izquierda de la imagen representa la placa al antes de presionar el botón, que está marcado con un cuadrado rojo. A la derecha, después de presionarlo, se puede observar que ahora hay más leds encendidos.

Siguiendo los mismos pasos anteriores, pero ejecutando el DHTtester en el Arduino IDE uno puede probar el funcionamiento del sensor de temperatura DHT11.

4.4. Repositorio del Proyecto

Los archivos que componen al presente proyecto se encuentran alojados en un repositorio de [GitHub](#) público proporcionado por la cátedra. Dentro del mismo se creó la siguiente estructura de directorios:

/Bitacora	Bitácora del proyecto en formato <i>markdown</i>
/Bitacora/imagenes	Conexiones de SIM800L
/Front	Programa del frontend
/GPRS	Programa de pruebas SIM800L Arduino
/app	Scripts de prueba en Docker
/arduinoClient	Programa del sistema de hardware
/backend	Programa del backend (local y remoto)
/docs	Documentación (informes)
/emqx	Programas de prueba EMQX + capturas
/mysql	Programas de prueba MySQL
/videos	Videos de funcionamiento del proyecto

Tabla 5. Estructura de directorios del proyecto

Los videos subidos al directorio correspondiente representan cada una de las partes del proyecto individualmente: un video muestra el frontend, otro el backend y otro el circuito con Arduino que realiza las mediciones y transmite los datos.

Apéndice A: Materiales 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 6. Presupuesto de los materiales obtenidos (Arduino UNO, DHT11, GPRS SIM 908C: MercadoLibre, octubre 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.