Actividad 3

Una vez instalado el SDK de Moddable trata de realizar los siguientes programas que utilizan el protocolo MQTT.

1. **Hacer un programa que se conecte a un broker MQTT para mostrar los mensajes que se publiquen en todos los tópicos debajo de la ruta `ESP32/mqtt/example`. Además publicará la fecha y un número aleatorio bajo `ESP32/<id>` donde `<id>` será un número único para el dispositivo.**

Necesitamos la clase `MQTT` y por lo tanto el módulo `mqtt`, además de establecer una conexión de red. Dejaremos que el módulo `net` se encargue de establecer la conexión WiFi. Por ello el fichero `manifest.json` deberá contener:

```
{
  "include": [
    "$(MODDABLE)/examples/manifest_base.json",
    "$(MODDABLE)/examples/manifest_net.json",
    "$(MODDABLE)/modules/network/mqtt/manifest.json"
  ],
  "modules": {
    "*": [
      "./main"
    ]
  },
  "config": {
    "ssid": "MiWiFi",
    "password": "secreto"
  }
}
```

Para obtener un identificador único para el dispositivo podemos basarnos en la dirección MAC de su interfaz WiFi. Esta se puede obtener mediante NET.get("MAC").

Un posible código que soluciona el ejercicio es:

```
import Net from "net";
import Client from "mqtt";
import Timer from "timer";

const mac = Net.get("MAC");
const id = `ESP32_${mac.replaceAll(":","").slice(-6)}`;
const mqtt  = new Client({
  host: "10.209.2.232",
  timeout: 60000,
  id,
});

mqtt.onReady = function() {
```

```
14   this.subscribe("ESP32/mqtt/example/#");
15
16   this.timer = Timer.set(() => {
17     this.publish(`ESP32/${id}/date`, Date());
18     this.publish(`ESP32/${id}/random`, Math.random());
19    }, 0, 5000);
20 };
21 mqtt.onMessage = function(topic, body) {
22   trace(`received "${topic}": ${String.fromArrayBuffer(body)}\n`);
23 };
24 mqtt.onClose = function() {
25   trace('lost connection to server\n');
26   if (this.timer) {
27     Timer.clear(this.timer);
28     delete this.timer;
29   }
30 };
```

Una vez importadas los módulos (líneas 1 a 3) sabemos que tenemos conexión del red ya que `net` se encarga es su inicialización.

Generamos el identificador único para el dispositivo (líneas 5 y 6) basándonos en la dirección MAC de su interfaz WiFi. Esta se puede obtener mediante `NET.get("MAC")`.

Luego procedemos a realizar la conexión al servidor MQTT, que en nuestro caso está en la dirección IP `10.209.2.232`. A continuación definimos las acciones a realizar cuando se produzcan los distinto eventos de la conexión MQTT: conexión, recepción de mensaje, cierre de la conexión:

- `onReady`: una vez conectados nos suscribimos a los tópicos de interés (línea 14): todos los que están bajo `ESP32/mqtt/example`. Además lanzamos temporizador repetitivo que publique la fecha (línea 17) y un número aleatorio (línea 18) en los tópicos indicados cada 5 segundos.
- `onMessage`: al recibir cualquier mensaje de los tópicos a los que nos hemos suscritos (en la línea 14) enviamos por la serial la ruta del tópico y su contenido como cadena de caracteres (línea 22).
- `onClose`: avisamos por la serial (línea 25) y desactivamos y borramos el temporiza que se había creado en la línea 16 para evitar que trate de enviar mensaje sobre una conexión cerrada.

2. **Hacer programa que publique en broker MQTT el estado de un pulsador (en `ESP32/<id>/boton`) y un potenciómetro (en `ESP32/<id>/poten`). Además cambien el estado de un led rojo y uno verde según los valores que se publiquen en `ESP32/<id>/leds`.**

Para resolver este ejercicio debemos, no solo usar MQTT, sino también acceso a entrada y salida digital (para pulsador y leds) y entrada analógica (para leer potenciómetro). Por ello `manifest.json` debe incluir todos esos módulos:

```
{
  "include": [
    "$(MODDABLE)/examples/manifest_base.json",
    "$(MODDABLE)/examples/manifest_net.json",
    "$(MODDABLE)/modules/network/mqtt/manifest.json",
    "$(MODULES)/pins/digital/manifest.json",
    "$(MODULES)/pins/digital/monitor/manifest.json",
    "$(MODDABLE)/modules/pins/analog/manifest.json"
  ],
  "modules": {
    "*": [
      "./main"
    ]
  },
  "config": {
    "ssid": "MiWiFi",
    "password": "secreto"
  }
}
```

Combinando lo que se vio en las actividades 1 con lo utilizado en la solución del ejercicio anterior, un posible programa solución a codificar en `main.js` sería:

```
import Net from "net";
import Client from "mqtt";
import Timer from "timer";
import Digital from "pins/digital";
import Monitor from "pins/digital/monitor";
import Analog from "pins/analog";

const PIN_ADC = 0;
const BUTTON_PIN = 0;

trace(`Botón en el pin ${BUTTON_PIN}\n`);

const bot0 = new Monitor({
  pin: BUTTON_PIN,
  mode: Digital.InputPullUp,
  edge: Monitor.Rising | Monitor.Falling,
});
```

```
18
19
20 const rojo = new Digital({
21     pin: 2,
22     mode: Digital.Output
23 });
24 rojo.estado = 0;
25 rojo.write(rojo.estado);
26
27 const verde = new Digital({
28     pin: 16,
29     mode: Digital.Output
30 });
31 verde.estado = 0;
32 verde.write(verde.estado);
33
34 const mac = Net.get("MAC");
35 const id = `ESP32_${mac.replaceAll(":","").slice(-6)}`;
36 const mqtt  = new Client({
37   host: "10.209.2.232",
38   timeout: 60000,
39   id,
40 });
41
42 let mqttReady = false;
43
44 const topicoLeds = `ESP32/${id}/leds`
45
46 mqtt.onReady = function() {
47   mqttReady = true;
48   trace(`Servidor MQTT listo\n`);
49
50   this.subscribe(topicoLeds);
51
52   this.timer = Timer.repeat(() => {
53     let valor = Analog.read(PIN_ADC);
54     trace(`Valor leído ${valor}\n`);
55     mqtt.publish(`ESP32/${id}/poten`, valor);
56   }, 1000);
57 };
58
59 mqtt.onMessage = function(topic, body) {
60   const bodyStr = String.fromArrayBuffer(body);
61   trace(`received "${topic}": ${bodyStr}\n`);
62   if(topic == topicoLeds) {
63     // invertimos el led indicado
64     if (bodyStr == 'rojo') {
65       rojo.estado = ! rojo.estado;
66       rojo.write(rojo.estado);
```

```
67      }
68      if (bodyStr == 'verde') {
69        verde.estado = ! verde.estado;
70        verde.write(verde.estado);
71      }
72    }
73  };
74
75  mqtt.onClose = function() {
76    trace('lost connection to server\n');
77    mqttReady = false;
78    if (this.timer) {
79      Timer.clear(this.timer);
80      delete this.timer;
81    }
82  };
83
84  let contador = 0;
85
86  bot0.onChanged = function() {
87    trace(`Botón estado ${this.read()} → ${++contador}`
88      + ` → ^ ${this.rises} v ${this.falls}\n`);
89    if (mqttReady) {
90      mqtt.publish(`ESP32/${id}/boton`, `${this.read()} → ${++contador}`
91      + ` → ^ ${this.rises} v ${this.falls}`);
92    }
93  }
```

Las principales novedades son:

- Ahora nos suscribimos al tópico `ESP32/${id}/leds` (líneas 44 y 50). En `onMessage()` estudiamos el contenido del mensaje. Si éste contiene `'rojo'` cambiamos el estado del led rojo y si contiene `'verde'` cambiamos el estado del led verde (líneas 62 a 72).
- En el temporizador leemos el valor de la entrada analógica y la publicamos en `ESP32/${id}/leds` cada segundo (líneas 52 a 56).
- Finalmente usamos el `onChange()` del monitor de la entrada digital conectada al pulsador para emitir en el tópico `ESP32/${id}/boton` un texto que dice el estado del pulsador y el número de flancos que se llevan producidos (línea 90 y 91).