Gestión Inteligente de Unidades de Estudio:
Biblioteca Nacional Mariano Moreno

Trabajo Final, Internet de las Cosas 2025

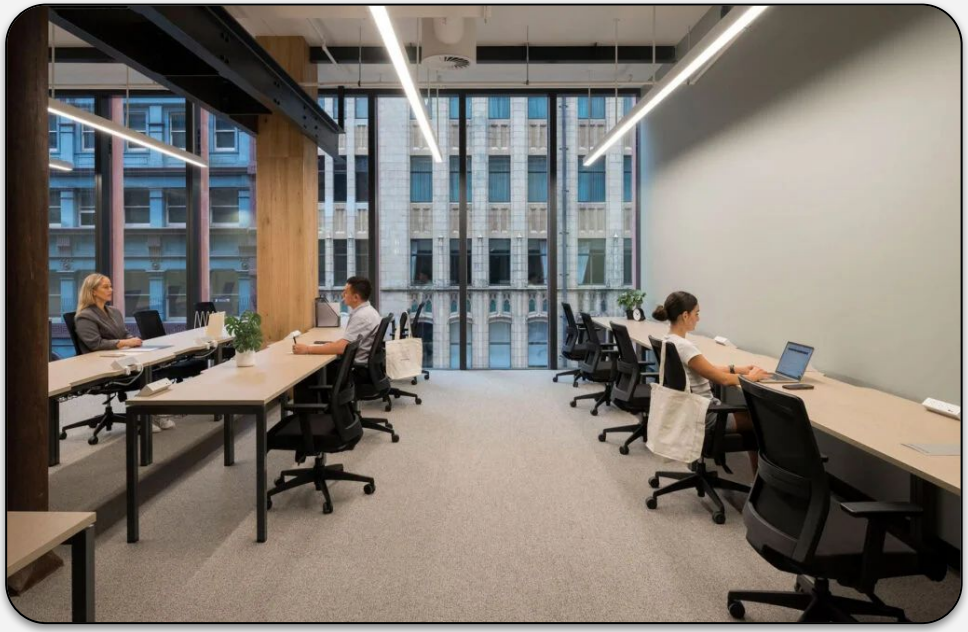
Por Juan Ignacio Acuña

Problema

- **Gestión ineficiente** de unidades de estudio o trabajo
- Necesidad de los usuarios de **conseguir una unidad disponible**
- **Falta de datos precisos** sobre el uso de las unidades

*A modo de ejemplo, se tomará como caso de estudio a la **Biblioteca Nacional Mariano Moreno (BNMM)**, teniendo en cuenta su inmensa cantidad de unidades de estudio.*

Salón Principal de Lectura de la Biblioteca Nacional Mariano Moreno



Oficina con disposición “hot desk”

Objetivo

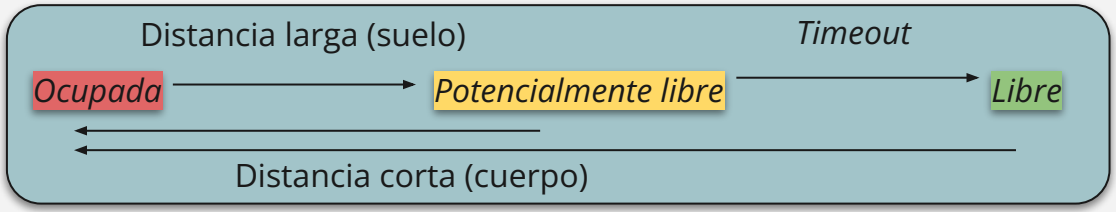
- **Mejorar la experiencia del usuario** con una solución autónoma e interactiva
- Proporcionar una manera de **recolectar información** en tiempo real y automáticamente
- **Optimizar el uso de los espacios**
- **Mejorar la toma de decisiones administrativas**
- Sentar las bases de un **sistema escalable**

Solución: Los nodos

Sistema de nodos inteligentes ESP32 distribuidos a lo largo de las mesas de trabajo.

A cada nodo se le asignan **unidades**, para las cuales:

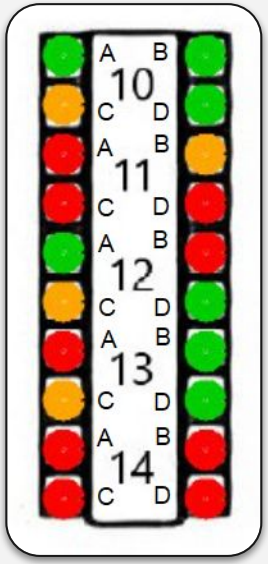
- Hace una **lectura de distancia** al suelo, utilizando un sensor de distancia (*Sensor Ultrasónico HC-SR04*) que se ubica en la parte inferior de la mesa
- Maneja su **estado** (*Libre*, *Potencialmente libre* u *Ocupada*)



Además, el nodo se encarga de enviar la información recolectada hacia un servidor central a través del **protocolo MQTT**

Adicionalmente, el nodo *hostea* una **interfaz web** que permite interactuar con él localmente. Es posible visualizar las lecturas de los sensores así como alterar el estado de cada unidad asignada.

Solución: Arquitectura



Disposición de nodos en las mesas del Salón Principal de Lectura

Nodo 10

Protocolo MQTT

Nodo 11

Nodo 12

Nodo 13

Nodo 14

mosquitto

CPU

Node-RED

InfluxDB

Grafana

Servidor central

Telegram

Entendiendo el funcionamiento de los nodos

Veamos por encima las partes más importantes del código de los nodos...

```ino
void setup() {
  Serial.begin(115200);
  setupWifi();
  setupMqtt();
  spiffsInit();
  setupServer();
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_SENSOR_A_TRIG, OUTPUT);
  pinMode(PIN_SENSOR_A_ECHO, INPUT);
  pinMode(PIN_SENSOR_B_TRIG, OUTPUT);
  pinMode(PIN_SENSOR_B_ECHO, INPUT);
  pinMode(PIN_SENSOR_C_TRIG, OUTPUT);
  pinMode(PIN_SENSOR_C_ECHO, INPUT);
  pinMode(PIN_SENSOR_D_TRIG, OUTPUT);
  pinMode(PIN_SENSOR_D_ECHO, INPUT);
}

void loop() {
  mqttVerifyReconnectReceive();
  readDistanceAndHandleUnits();
  mqttPublishData();
  delay(100); // Delay para no saturar el loop
}
```

```ino
// Cambiar estado de nodo A
server.on("/ocuparUnitA", HTTP_GET, [](AsyncWebServerRequest *request){
  stateUnitA = OCUPADO;
  request->send(SPIFFS, "/index.html", String(), false, processor);
});
server.on("/liberarUnitA", HTTP_GET, [](AsyncWebServerRequest *request){
  if (stateUnitA == OCUPADO) {
    stateUnitA = POTENCIALMENTE_LIBRE;
    unitALibreTimeout = millis();
  } else if (stateUnitA == POTENCIALMENTE_LIBRE) {
    stateUnitA = LIBRE;
  }
  request->send(SPIFFS, "/index.html", String(), false, processor);
});
server.on("/stateUnitA", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(200, "text/plain", stateToString(stateUnitA));
});
server.on("/distanceSensorA", HTTP_GET, [](AsyncWebServerRequest *request){
  if (distanceSensorA == -1) {
    request->send(200, "text/plain", "ERROR");
  } else {
    request->send(200, "text/plain", String(distanceSensorA));
  }
});
```

Setup(), Loop() y algo de setupServer()

```html
<h2>Unidad 37A</h2>
<p>Distancia (cm): <span id="distanceSensorA"></span></p>
<p>Estado: <strong id="stateUnitA">%STATE_UNIT_A%</strong></p>
<p>
  <a href="ocuparUnitA"><button class="button">OCUPAR</button></a>
  <a href="liberarUnitA"><button class="button button2">LIBERAR</button></a>
</p>
```

```javascript
<script>
  setInterval(function() {
    fetch('/distanceSensorA')
      .then(response => response.text())
      .then(data => {
        document.getElementById("distanceSensorA").innerHTML = data;
      });
    fetch('/stateUnitA')
      .then(response => response.text())
      .then(data => {
        document.getElementById("stateUnitA").innerHTML = data;
      });
```

```javascript
String processor(const String& var){
  Serial.println(var);
  if (var == "STATE_UNIT_A") {
    return stateToString(stateUnitA);
  }
```

Interfaz web

```
long getUnitDistance(const String& unitName, int trigPin, int echoPin) {
  long duration, distance;
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);

  if (duration ≤ 0) {
    return -1;
  }

  distance = (duration * SOUND_SPEED) / 2; // cm
  Serial.printf("Unidad %s Distance: %ld cm\n", unitName.c_str(), distance);
  return distance;
}
```

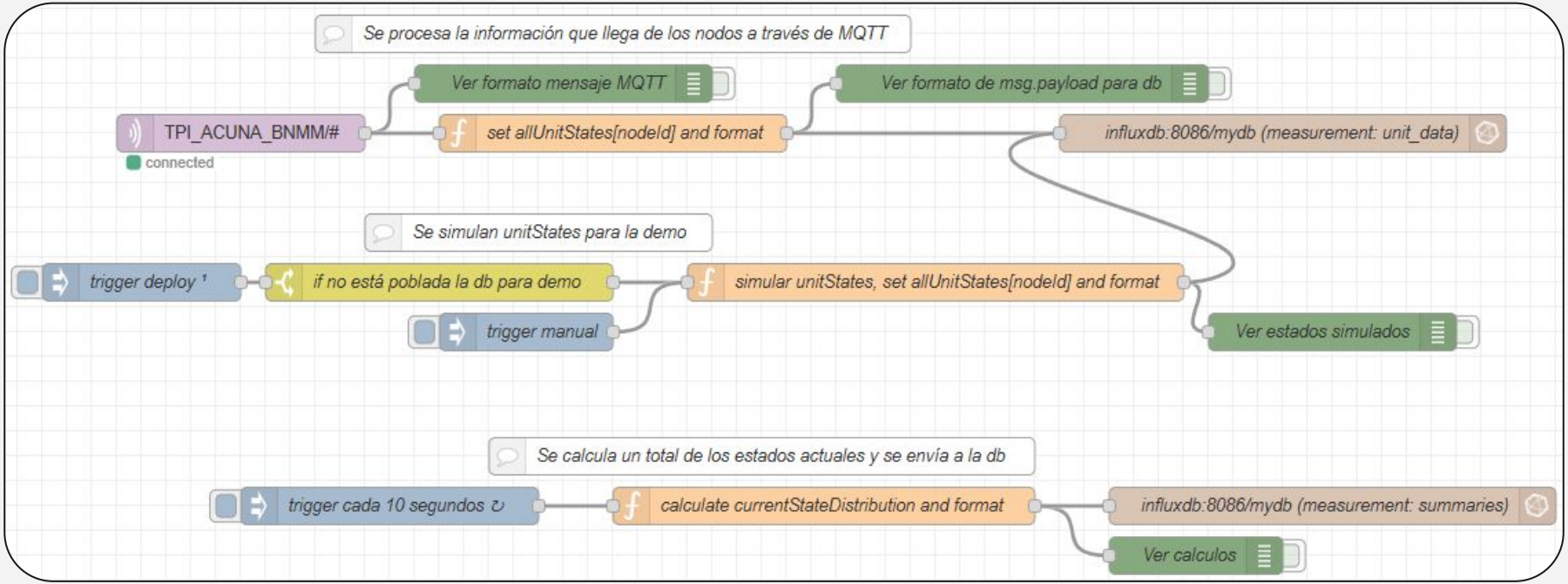
getUnitDistance()

readDistanceAndHandleUnit()...

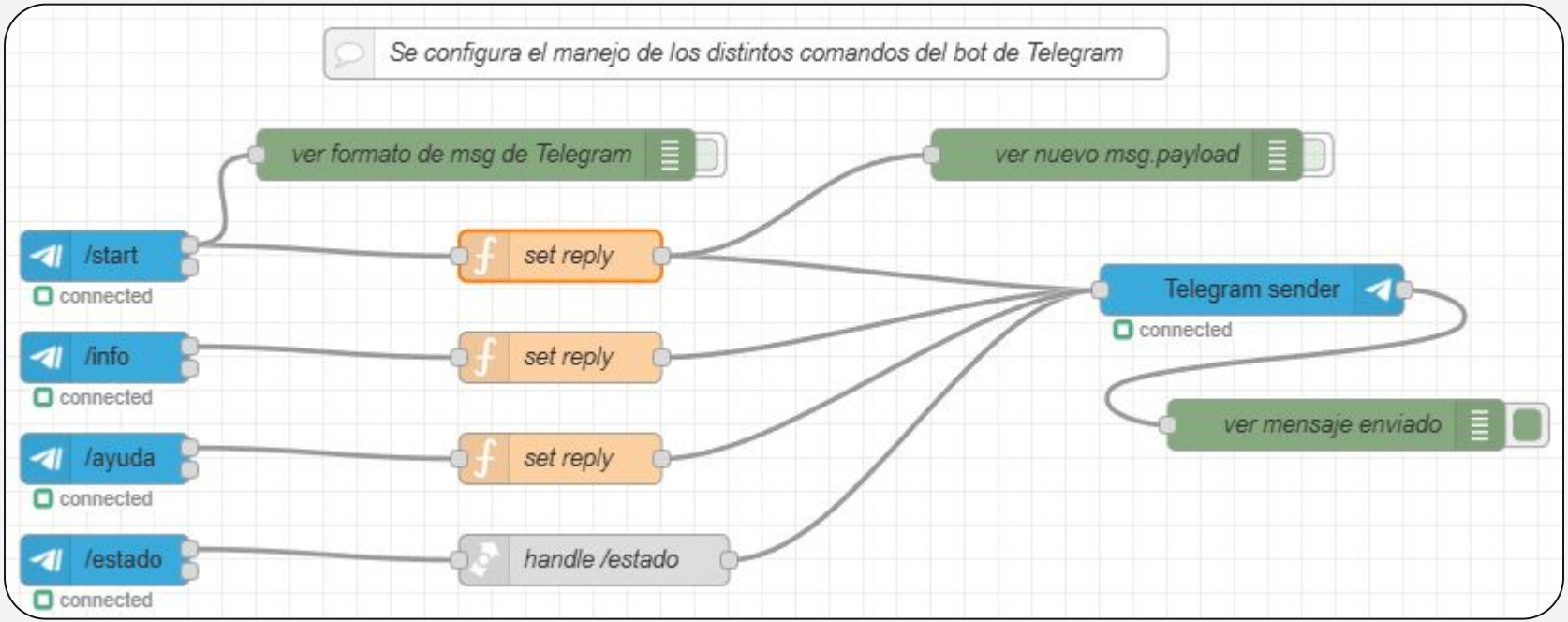
```
void readDistanceAndHandleUnit(const String& unitName, int trigPin, int echoPin, long& sensorDistance, UnitState& state, unsigned long& unitLibreTimeout) {
  sensorDistance = getUnitDistance(unitName, trigPin, echoPin);
  mqttClient.publish(("TPI_ACUNA_BNMM/" + String(NODE_ID) + "/" + unitName + "/distance").c_str(), String(sensorDistance).c_str());

  if (sensorDistance == -1) {
    Serial.printf("Error al leer datos de la unidad %s.\n", unitName.c_str());
    return;
  }

  if ((state == LIBRE) && (distanceMeansOcupado(sensorDistance))) {
    state = OCUPADO;
    Serial.printf("Unidad %s ocupada.\n", unitName.c_str());
    mqttClient.publish(("TPI_ACUNA_BNMM/" + String(NODE_ID) + "/" + unitName + "/state").c_str(), stateToString(state).c_str());
    return;
  }

  if ((state == OCUPADO) && (!distanceMeansOcupado(sensorDistance))) {
    state = POTENCIALMENTE_LIBRE;
    unitLibreTimeout = millis();
    Serial.printf("Unidad %s potencialmente libre.\n", unitName.c_str());
    mqttClient.publish(("TPI_ACUNA_BNMM/" + String(NODE_ID) + "/" + unitName + "/state").c_str(), stateToString(state).c_str());
    return;
  }

  if (state == POTENCIALMENTE_LIBRE) {
    if (distanceMeansOcupado(sensorDistance)) {
      state = OCUPADO;
      Serial.printf("Unidad %s sigue ocupada.\n", unitName.c_str());
      mqttClient.publish(("TPI_ACUNA_BNMM/" + String(NODE_ID) + "/" + unitName + "/state").c_str(), stateToString(state).c_str());
      return;
    }
    if ((millis() - unitLibreTimeout) ≥ POTENCIALMENTE_LIBRE_TIMEOUT) {
      state = LIBRE;
      Serial.printf("Unidad %s libre.\n", unitName.c_str());
      mqttClient.publish(("TPI_ACUNA_BNMM/" + String(NODE_ID) + "/" + unitName + "/state").c_str(), stateToString(state).c_str());
    }
  }
}
```

Comprendiendo los flujos de Node-RED y el bot de Telegram

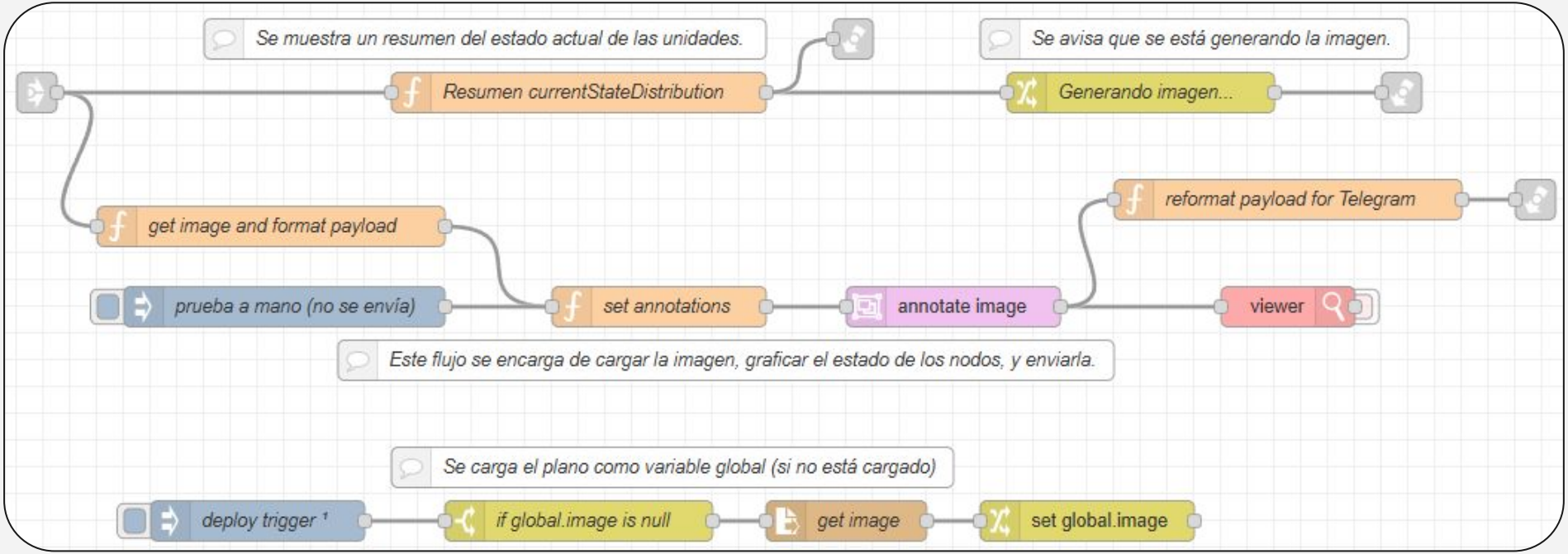
Ahora observemos los distintos flujos definidos en Node-RED y un ejemplo haciendo uso del comando fundamental del bot de Telegram...



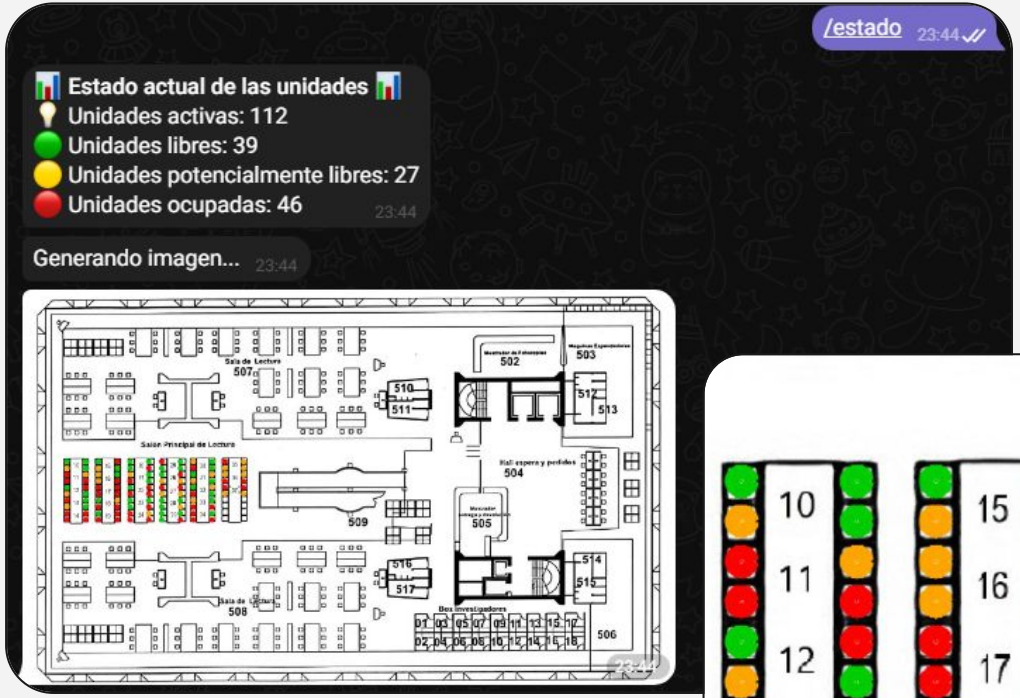
Flujo principal (MQTT, estados, DB)

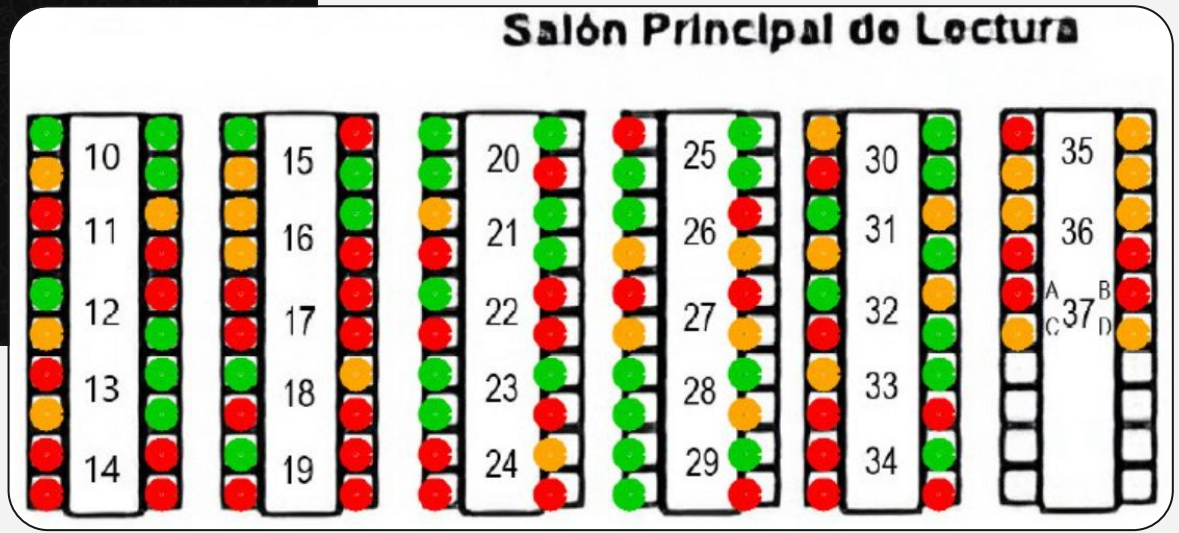


Bot de Telegram



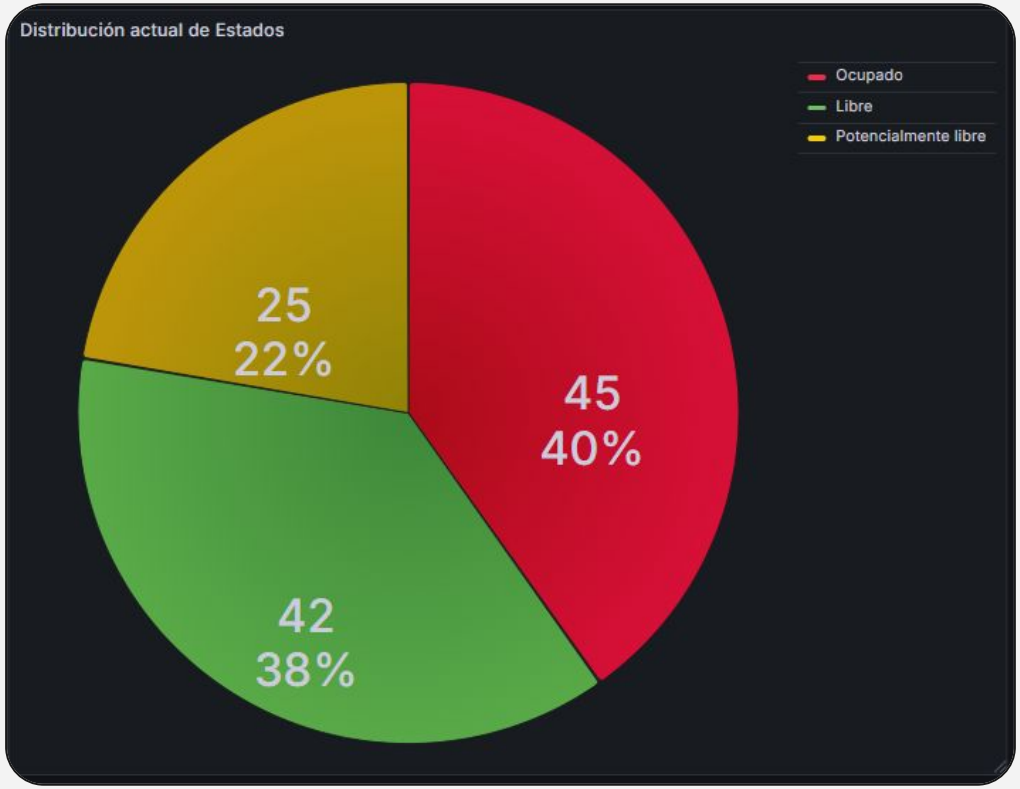
handle /estado





Consultando /estado por Telegram

Finalmente echemos un vistazo a la visualización por Grafana



Dashboard Principal

Conclusión: Próximos Pasos

- **Agregar funcionalidades al nodo**, especialmente si se utilizará el nodo en una sala cerrada como un *box investigador* o una sala de conferencias pequeña.
- **Complejizar** la lógica de estados con la ayuda de **nuevos sensores** (pulsador, otro sensor de distancia, cámara de video).
- **Extender** la solución a todas las salas de la biblioteca.
- Llevar la solución a **otros casos de estudio** (otra biblioteca, piso de oficina, aula).
- **Nuevos comandos** para Telegram que involucren control de nodos (requiere **autenticación**).
- **Dashboards más robustos**: horas pico de uso, salas o unidades más usadas, alertas de sala llena o vacía.

¡Muchas gracias!

Juan Ignacio Acuña (juanacunars@gmail.com)