
# TRABAJO FINAL DE GRADO

**TÍTULO DEL TFG: Programar notificaciones en Unity 3D para Plugin de CoAP-Californium**

**TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicación**

**AUTOR: Parra Sevillano, Sergio**

**DIRECTOR: Royo Vallés, M. Dolores**

**FECHA: 24 de octubre de 2018**

**Título:** Programar notificaciones en Unity 3D para Plugin de CoAP-Californium

**Autor:** Parra Sevillano, Sergio

**Director:** Royo Vallés, M. Dolores

**Data:** 24 de octubre de 2018

## Resumen

El avance en las nuevas tecnologías nos ha abierto un gran campo de posibilidades y de campos de investigación.

A día de hoy vivimos rodeados de dispositivos que están conectados a internet donde quiera que miremos. De entre ellos están los sensores, los cuales van a ser la base de este proyecto.

Podemos encontrar sensores en cualquier dispositivo de nuestro alrededor, estos tienen diferentes funcionalidades, en nuestro caso, para este proyecto, su funcionalidad será alertarnos de cualquier dato recolectado que este fuera de lo normal. Para ello dispondremos de una placa Arduino colocada estratégicamente en el entorno a monitorizar y a través de una aplicación móvil seremos capaces de monitorizar los valores de estos sensores remotamente.

Nuestra placa Arduino hará de servidor en nuestro caso, y el cliente será el usuario a través de la aplicación móvil.

En este proyecto se propondrán diferentes soluciones y con sus pros y sus contras y finalmente presentare la solución más apropiada que será la que finalmente apliquemos.

**Title:** Coding notifications in Unity 3D for CoAP-Californium Plugin

**Author:** Parra Sevillano, Sergio

**Director:** Royo Vallés, M. Dolores

**Date:** October 24, 2018

## Overview

Advances in new technologies have opened up a large field of possibilities and fields of research.

Today we live surrounded by devices that are connected to the internet wherever we look. Among them are the sensors, which will be the basis of this project.

We can find sensors in any device around us, they have different functionalities, in our case, for this project, its functionality will alert us of any data collected that is out of the ordinary. For this we will have an Arduino board placed strategically in the environment to be monitored and through a mobile application we will be able to monitor the values of these sensors remotely.

Our Arduino board will act as a server in our case, and the client will be the user through the mobile application.

In this project different solutions will be proposed with their pros and cons and finally I will present the most appropriate solution that will be the one we finally apply.

# Contenido

# Capítulo 1. INTRODUCCIÓN

A donde quiera que miremos la tecnología avanza a pasos agigantados y cada día que pasa llevamos los límites de lo posible un poco más allá. La velocidad del cambio es impresionante y se encuentra en continua aceleración. Cosas que hace unos años no podríamos ni llegar a imaginar, a día de hoy forman parte de nuestras vidas como, por ejemplo, el hecho de estar conectados en todo momento.

Un invento que ha revolucionado nuestra manera de vivir en la última década han sido los Smartphones. A día de hoy los *Smartphone* superan en número a la población mundial. Y cada día, estos se superan con nuevas funcionalidades y capacidades. Tenemos la facilidad de estar conectados al alcance de nuestras manos, de poder obtener información y comunicarnos con el resto del mundo desde cualquier punto. Tareas que antes podrían suponer gran trabajo o elevados costes, ahora se han simplificado de manera notable.

Se puede decir, que ha sido un invento que ha cambiado nuestras vidas y ha marcado un antes y un después en la historia tecnológica. Creando también, nuevos puestos de trabajo de desarrollo de aplicaciones y *startups.*

Así mismo, no solo los *Smartphone* están conectados a la red. A día de hoy vivimos rodeados de millones de dispositivos capaces de recolectar información del entorno y subir los datos a la red. Uno de estos dispositivos son los sensores que están teniendo un gran éxito en nuestra sociedad por ser básicamente un conjunto de dispositivos de recolección de información. Estos son conocidos por no consumir muchos recursos, en muchos de los casos se alimentan de baterías las cuales se recargan a través de placas fotovoltaicas, de manera que podemos situarlos en lugares recónditos y de difícil acceso debido a que no necesitan un constante mantenimiento.

Uniendo ambas tecnologías de los Smartphones y sensores, se pueden crear aplicaciones con muchas finalidades que pueden ahorrar tiempo y dinero en sectores empresariales e industriales, o incluso a nivel de uso personal.

## 1.1. Objetivos

La idea principal de este proyecto es crear una aplicación para *Smartphone* que sea capaz de recibir datos de sensores y avisar al usuario cuando estos sensores detecten alguna anomalía.

La intención es que estos sensores estén colocados en una habitación de manera que podamos monitorizar diferentes datos de esta misma como, por ejemplo, temperatura, detección de luz, detección de movimiento, humedad, etc. De manera que, en un caso práctico, estos sensores nos puedan avisar de la subida de temperatura en una habitación indicándonos de la posibilidad de incendio o detectar la presencia de un intruso, o simplemente la posibilidad de que nos hayamos dejado la luz encendida de nuestro salón.

Para llevar a cabo este proyecto vamos a utilizar diferentes tecnologías como Unity para crear la aplicación móvil que nos permitirá interactuar con los sensores y nos mostrará los datos por la pantalla de nuestro Smartphone, Android para crear los Plugins que permitirán conectarnos con el servidor y Arduino, donde estará alojado nuestro servidor que será el encargado de recoger los datos.

## 1.2. Planteamiento del proyecto

El presente documento se estructura de la siguiente manera:

Para empezar, explicare las diferentes tecnologías que se llevan a cabo para la realización de este proyecto y como están relacionadas entre ellas.

Después, presentare los entornos de los cuales partimos explicando detalladamente el funcionamiento.

Y para acabar, planteare posibles soluciones, como he llegado a ellas y cuál es la mejor a implementar.

## 1.3. Términos clave

A continuación, los términos clave imprescindibles junto a su definición correcta para una mejor compresión del documento.

- **open source**: Programas informáticos que permiten el acceso a su código por parte de programadores ajenos.

- **URI**: Son las siglas de *Uniform Resource Identifier* y sirve para identificar recursos en internet.

- **Multicast**: Envío de información a múltiples destinatarios simultáneamente

- **Protoboard:** Tablero con orificios interconectados eléctricamente entre si de manera interna en el cual se pueden insertar componentes electrónicos.

- **REST**: Se utiliza para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos.

- **Framework:** Entorno de trabajo o marco de trabajo

- **Thread:** Utilizado para llevar a cabo funciones simultaneas

# Capítulo 2. TECNOLOGÍAS

## 2.1. Internet de las cosas

### 2.1.1. Concepto

El IdC (*Internet de las Cosas*), o más comúnmente conocido como IoT (*Internet of things*), ha sido uno de los avances tecnológicos más relevantes en los últimos años. Este concepto hace referencia a un gran número de tecnologías que hacen posible obtener nuevas funcionalidades de objetos cotidianos, mejorando la productividad, disminuyendo costes, reduciendo riesgos, etc.

En esencia, es un sistema de máquinas u objetos de la vida cotidiana de los seres humanos capaces de recolectar datos con el objetivo de que estos datos puedan comunicarse unos con otros. Dicha comunicación entre objetos cotidianos implica la conexión de estos mediante internet, evolucionando esta última de una red de ordenadores interconectados a una red de objetos interconectados.

Hasta la aparición del termino IoT, la información era recolectada por las personas de manera manual, de manera que se experimentaban errores y retrasos. Sin embargo, si la información proviene del mismo objeto, en este caso, sensores, se puede hacer un seguimiento a tiempo real de su utilización, características, vida útil, batería, etc. Lo cual estamos hablando de aumentos de productividad y, por consiguiente, reducción de costes.

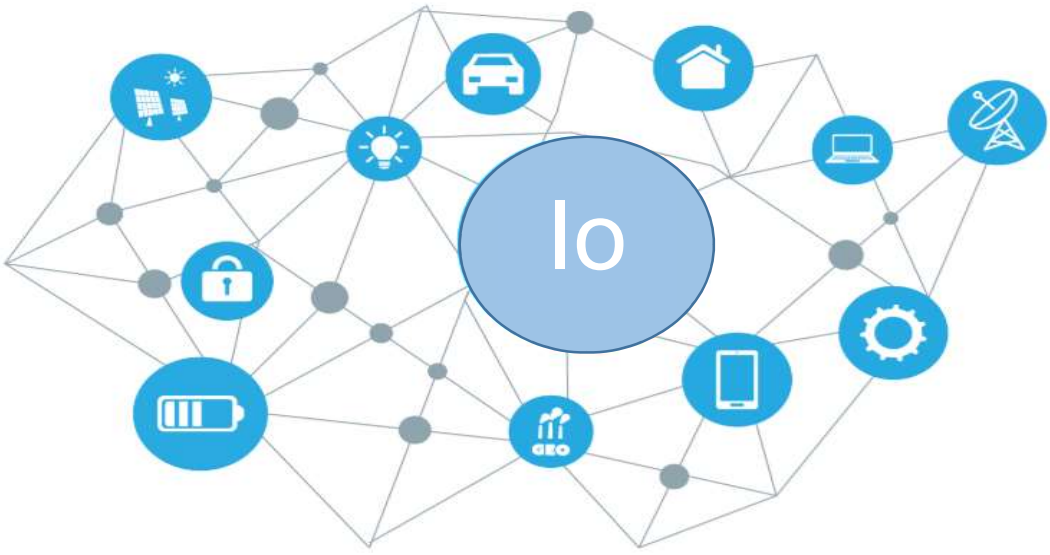


*Ilustración 1 Internet of Things [3]*

### 2.1.2. Historia

El Internet of things surgió hace 19 años [5], pero la idea de interconectar dispositivos ya se había planteado cerca de los años 70. La idea se planteó en un principio como "*embedded internet*" o "*pervasive computing*". Pero el termino actual "Internet of Things" fue concebido por Kevin Ashton en 1999. *Kevin Ashton* trabajo como "*supply chain optimization*" y quiso impresionar a su senior manager con una nueva tecnología llamada RFID. ÉI llamo a su presentación: *Internet of Things*

### 2.1.3. IoT en la actualidad

Como vemos en la siguiente figura basada en información proporcionada por la empresa Cisco, el crecimiento e implantación de dispositivos IoT crece de una forma exponencial, abriendo grandes posibilidades, no solo de negocio, sino de una mejora cualitativa de nuestra vida cotidiana

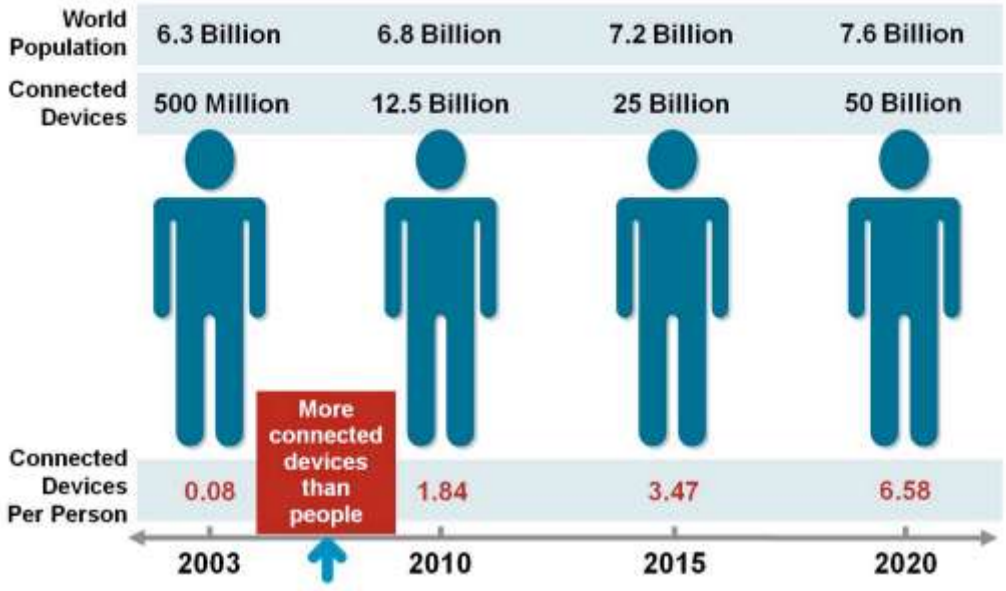


*Ilustración 2 Crecimiento de dispositivos IoT [6]*

Esta grafica nos indica en si misma que cada vez van a ser más los objetos con tecnología IoT de los cuales podremos obtener información de todo nuestro alrededor.

### 2.1.4. Aplicaciones

El ámbito del Internet of things se puede extrapolar a diferentes sectores, tanto comercial como industrial e incluso para el uso personal.

#### 2.1.4.1. Aplicaciones comerciales

El internet of things se ha instalado en muchas profesiones, una de ellas, la medicina, también conocida como "*Internet of Health Things*" [7], con la finalidad de recoger datos de los pacientes de forma más eficiente sin deteriorar la salud del paciente.

#### 2.1.4.2. Aplicaciones industriales

Como se ha comentado anteriormente, una de las ventajas del IoT es la velocidad y facilidad de recolección de datos. Aquí es donde interviene como ejemplo el sector de la Agricultura [8]. Gracias a los nuevos avances los agricultores pueden recolectar información acerca de sus cosechas remotamente con su Smartphone para detectar, por ejemplo, el estado de la tierra.

#### 2.1.4.3. Aplicaciones personales

En aplicaciones personales podemos incluir el objetivo de este proyecto, la monitorización de un espacio privado para controlar cualquier incidente o anomalía que pueda pasar en nuestro entorno, un término más conocido como "Smart Home".

Pero cuando hablamos de aplicaciones personales también nos estamos refiriendo a dispositivos que usamos día a día para monitorizar nuestras constantes vitales, ciclos de sueño, etc. Estamos hablando de una red PAN (*Personal Area Network*) es decir, una red de dispositivos centrados en el entorno más cercano de un individuo, como pueden llegar a ser, un reloj, un Smartphone, una Tablet, etc.

## 2.2. CoAP

CoAP (*Constrained Application Protocol*) es un protocolo software a nivel de aplicación pensado para ser usado en dispositivos electrónicos simples permitiendo que puedan comunicarse a través de Internet.

CoAP implementa la arquitectura *REST* en la que los recursos son controlados por el servidor mediante URIs y accedemos a ellos con las peticiones *GET, POST, PUT y DELETE*.

Se podría decir que CoAP es un modelo de HTTP el cual ha sido modificado teniendo en cuenta que está orientado a dispositivos simples de bajo consumo como, por ejemplo, los sensores, pero incluyendo otros requisitos como *multicast* y simplicidad, que son muy importantes para el Internet de las cosas [1]. En la figura siguiente se puede ver el protocolo de HTTP junto al de CoAP.

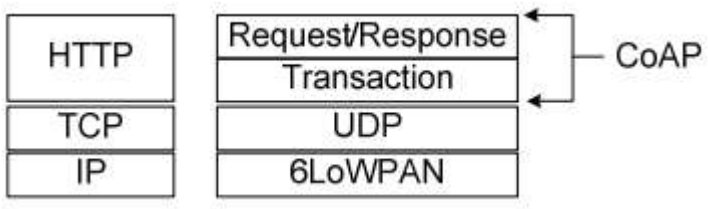


*Ilustración 3 Modelos HTTP y CoAP [10]*

### 2.2.1. Interacción del modelo

Un intercambio de mensajes CoAP es enviado por un cliente para solicitar una acción en un recurso, identificado por un URI, en un servidor.

A diferencia de HTTP, CoAP trata estos intercambios de forma asíncrona a través de un transporte UDP con soporte para interacciones de unidifusión y multidifusión [9]. Esto se logra utilizando los siguientes mensajes de transacción

- **Confirmable**: Mensaje el cual necesita confirmación por parte del servidor.

- **Non-Confirmable**: Mensaje el cual no necesita confirmación por parte del servidor, se debe a peticiones simples como por ejemplo la medición de un sensor.

- **Acknowledgment (ACK)**: Respuesta al mensaje confirmable para afirmar que el mensaje ha llegado correctamente, suele ir ligado a la información de respuesta.

- **Reset**: Mensaje enviado por parte del servidor para informar que la petición no ha llegado correctamente o no puede procesarla.

### 2.2.2. Modelo de mensajes

El modelo de mensajes de CoAP está basado en el intercambio de mensajes UDP entre nodos finales.

CoAP utiliza un encabezado de longitud fija (4 bytes) [2] que viene seguido de una carga útil (*payload*). Este formato de mensaje es compartido por los menajes de *request* / *response*. Cada mensaje contiene un ID de mensaje utilizado para detectar duplicados y para fiabilidad opcional.

La fiabilidad se proporciona al marcar un mensaje como Confirmable (CON). El mensaje confirmable se retransmite utilizando un tiempo de espera predeterminado y retroceso exponencial (*exponential back-off*) entre retransmisiones, hasta que el receptor envía un mensaje ACK con el mismo ID de mensaje (en este ejemplo, 0x7d34); ver la ilustración 4. Cuando un servidor no es capaz de procesar un mensaje confirmable, es decir, ni siquiera es capaz de proporcionar una respuesta de error adecuada, responde con "*Reset message*" (RST) en lugar de un ACK.

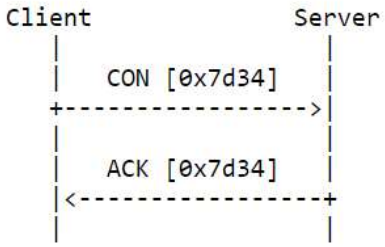


*Ilustración 4 Transmisión de mensaje confirmable [2]*

Hay mensajes que no necesitan de cierta fiabilidad. Este puede enviarse como mensaje "*Non-confirmable message"* (NON). Estos no necesitan de un mensaje ACK de respuesta, pero aun así tienen un ID de mensaje para la detección de duplicados (en este ejemplo,0x01a0), ver la siguiente ilustracion. Cuando un destinatario no puede procesar un Non-confirmable message, puede responder con un mensaje de reinicio (RST)

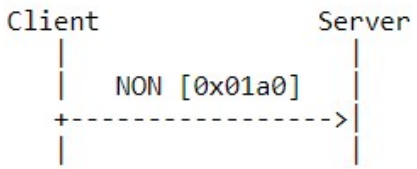


*Ilustración 5 Transmisión de mensaje no confirmable [2]*

### 2.2.3. Respuesta asíncrona

La única diferencia entre los mensajes de *request* y *response* es que las respuestas pueden ser asíncronas, es decir, para según que peticiones el servidor requiere de un tiempo para procesar la información. Entonces el servidor desvincula la respuesta del mensaje de ACK, enviando un ACK sin ninguna respuesta para que el cliente deje de enviar peticiones constantemente.

Una vez el servidor tenga la respuesta, se la enviara al cliente con el mismo Token o ID que el request (para saber que la respuesta corresponde con el request) tal y como se muestra en la ilustración 6.

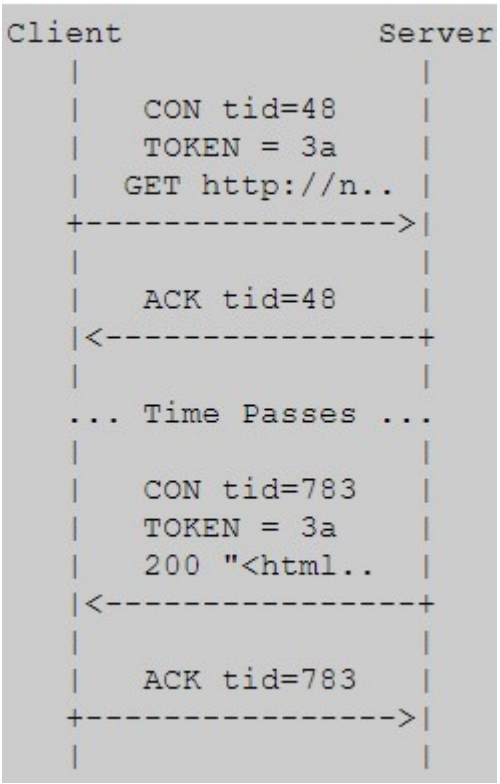


*Ilustración 6 Respuesta asíncrona [2]*

### 2.2.4. Modelo de mensajes Request/Response

La semántica de Request/Response de CoAP está en los mensajes, que incluyen un Method Code o Response Code, respectivamente. Este código nos da información opcional (o predeterminada), como por ejemplo la URI y el tipo de payload.

CoAP utiliza los métodos GET, PUT, POST y DELETE [1].

| Petición | Descripción |
|---|---|
| Get | Recupera la información del recurso identificado por el URI. |
| Put | Solicita que el recurso identificado por el URI se actualice. |
| Post | Se utiliza para solicitar al servidor que cree un nuevo recurso bajo el URI principal solicitado |
| Delete | Solicita que se elimine el recurso identificado por el URI de solicitud |

*Tabla 1 Métodos CoAP*

## 2.3. Californium

Californium (Cf) [6] es una implementación de *open source* de CoAP. Está escrito en Java y se dirige a entornos sin restricciones, como infraestructuras de servicios (por ejemplo, servidores proxy, directorios de recursos o servicios en la nube) y entornos menos restringidos, como dispositivos integrados que ejecutan Linux (por ejemplo, controladores inteligentes de fábrica/hogar). En particular, Cf se centra en la escalabilidad del servicio para aplicaciones del Internet of Things a gran escala.

## 2.4. Android

Una parte imprescindible del proyecto va a estar formada por los Plugins, estos van a ser los encargados de comunicar el cliente con el servidor.

### 2.4.1. Plugins

Podemos decir que un Plugin es básicamente un añadido al código que nos proporciona funcionalidades nuevas. En nuestro caso los Plugins son usados

básicamente para interacción con Android para llamar métodos escritos en java a partir de los scripts de Unity en C#.

Para llamar a un Plugin a partir de un script necesitaremos inicializar una variable como objeto de Android, esta variable la inicializaremos con el valor del paquete del mismo Plugin.

```
private AndroidJavaObject coap = new AndroidJavaObject("com.coap.client.CoapClientManager");
```

*Ilustración 7 Inicialización objeto*

Acto después, solo tendremos que llamar a la función del Plugin a partir del método "*Call*" con los parámetros que necesite la misma función, tal y como se muestra en la siguiente ilustración.

```
coap.Call("doPut", uri, data);
```

*Ilustración 8 Llamada función Plugin*

Donde en este caso la función "doPut" sería la encargada de enviar la petición al servidor

En este proyecto se va a usar un Plugin que reúna las funciones de poder intercambiar información con el servidor y, por otro lado, un Plugin que programe notificaciones en nuestro dispositivo Android cuando lo deseemos.

## 2.5. Unity

Para la parte de la interfaz vamos a usar Unity. Este software nos permite crear libremente elementos que nos van a ayudar a interactuar con la aplicación y tener un control de los datos proporcionados por el servidor.

Una parte imprescindible en el código del cliente van a ser los eventos o *"Events"*

### 2.5.1. Events

Los eventos son una manera de delegar funciones a clases cuando queremos alertar de que algo ha sucedido. Cuando esta situación específica ocurra, en

nuestro caso, una respuesta del servidor, invocaremos nuestro evento y este llamara a los métodos de las clases que tiene suscritas.

A continuación, la función del evento formado por la clase que va a contener la respuesta del servidor.

```
public event EventHandler<ResponseReceivedEventArgs> ResponseReceivedHandler;
```

*Ilustración 9 Función evento*

Esta será la clase del evento, es decir, una vez llamemos al evento, serán del formato de la clase asignada

```
public class ResponseReceivedEventArgs : EventArgs
{
    public string Resource { get; set; }
    public string Data { get; set; }
}
```

*Ilustración 10 Clase asignada al evento*

Por otro lado, tendremos otra clase, la cual al inicializarse suscribirá una función de esta clase al evento.

```
void Start()
{
    coapManager.ResponseReceivedHandler += ResponseReceived;

}

public void ResponseReceived(object sender, ResponseReceivedEventArgs e)
{
    label.text = "UiManager "+e.Resource + " : " + e.Data;
    Debug.Log("La respuesta es: " + e.Resource + " : " + e.Data);
}
```

*Ilustración 11 Suscripción de método*

# Capítulo 3. ENTORNOS

## 3.1. Cliente/Servidor

Para realizar este proyecto partimos de la base de dos entornos, un cliente y un servidor.

### 3.1.1. Cliente

El cliente, una aplicación para Smartphone, estará hecho en Unity. El usuario podrá interactuar con la aplicación a través de una UI (*User Interface*) que nos permitirá de forma fácil e intuitiva interactuar con los sensores y obtener los datos recolectados. El cliente será el encargado de activar las peticiones (request) que nos van a dar la información del servidor.

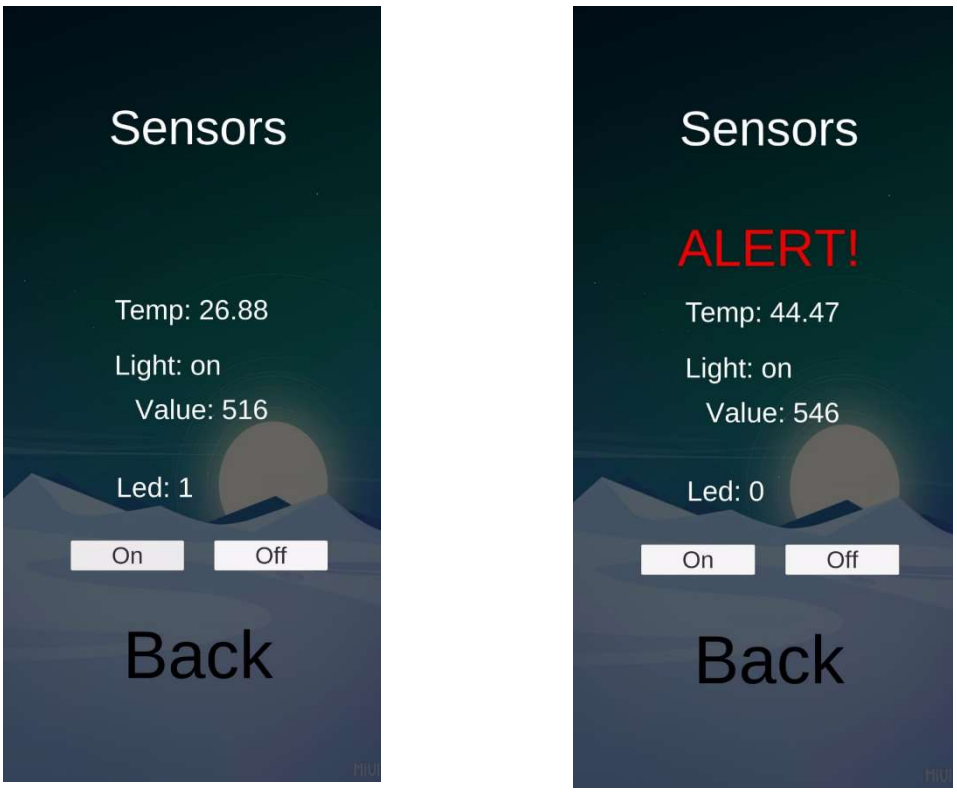


*Ilustración 12 UI Aplicación*

Para poder conectar con el servidor, Unity se va a ayudar de los Plugins hechos en Java los cuales se ejecutarán en el Smartphone y harán posible la conexión con el dispositivo y el servidor. Este Plugin será un framework de CoAP comentado anteriormente (Californium). Este Plugin dispondrá de los diferentes métodos comentados anteriormente (PUT, GET, POST, DELTE), que serán los encargados de mandar las peticiones al servidor y obtener la información.

### 3.1.2. Servidor

El servidor estará formado por un Arduino donde estará alojado nuestro servidor MicroCoAP, una versión de CoAP para Arduino. Además, entrarán en juego todos los sensores a monitorizar, los cuales estarán conectados directamente a una placa protoboard y esta directamente al Arduino.

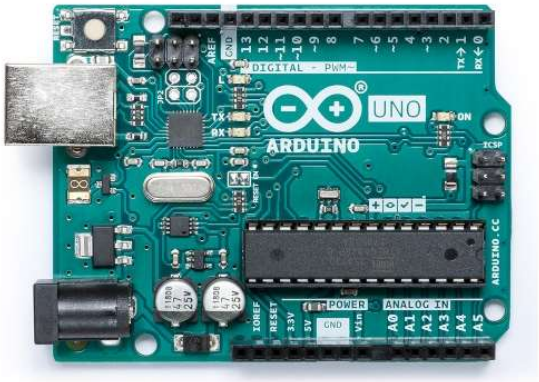


*Ilustración 13 Arduino UNO*

El Arduino estará posicionado en la habitación a monitorizar. El servidor será el encargado de recibir las peticiones y procesarlas para controlar los sensores.

Los sensores estarán conectados a la placa de la siguiente manera:

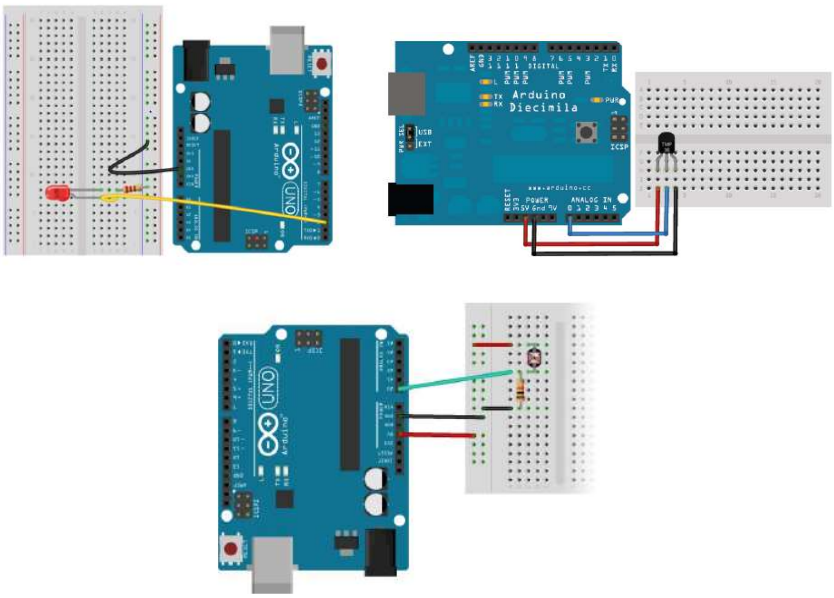

*Ilustración 14 Conexión sensores*

Antes de ponernos a desarrollar la aplicación, tenemos que asegurarnos que los sensores están bien instalados y el servidor recolecta correctamente los datos de los sensores. Para ello vamos a utilizar un Plugin de Google Chrome que nos permite testear el servidor de CoAP.

#### 3.1.2.1. Cooper

Como bien he comentado anteriormente, Cooper es un Plugin que nos permite testear nuestro servidor de CoAP, para ello necesitaremos instalar el Plugin en Google Chrome y añadir la ip de nuestro servidor

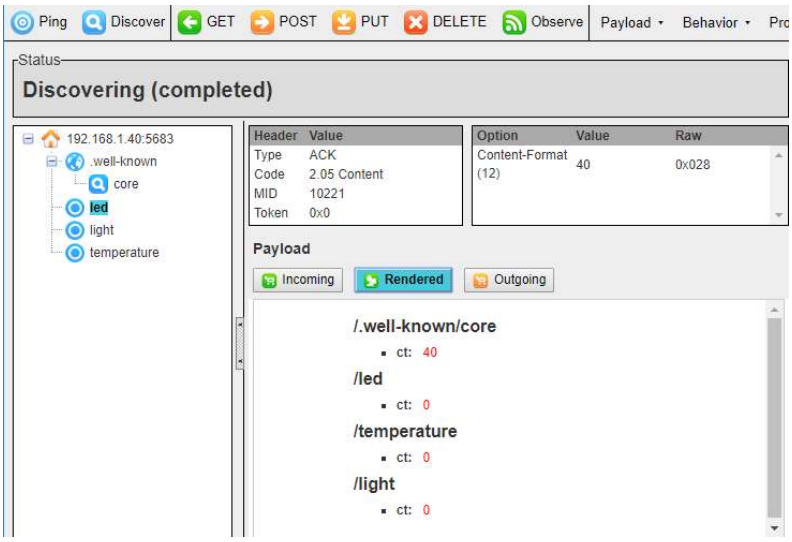


*Ilustración 15 Cooper*

En este menú nos aparecerán todos los sensores instalados en nuestro servidor, y nos permitirá interactuar con ellos a partir de los métodos ya conocidos como PUT, GET, POST, DELETE.

Es así como podremos testear de manera rápida que nuestro servidor esta recolectando los datos de manera correcta.

## 3.2. Entorno de partida

Partimos de una base que combina los dos sistemas anteriormente comentados, cliente/servidor. Gracias al Plugin de Californium podremos poner en contacto ambos entornos a través de peticiones.

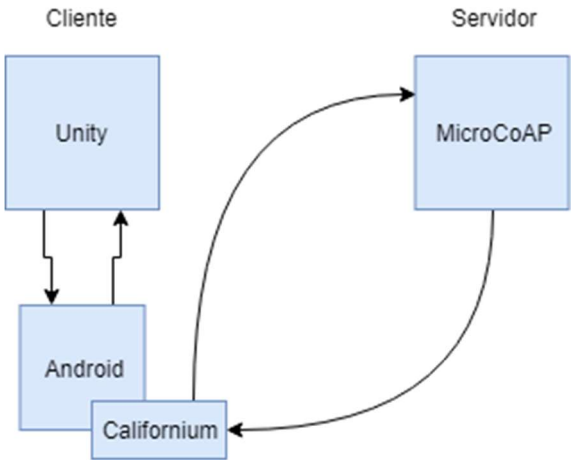


*Ilustración 16 Esquema de los sistemas*

Este sistema nos permite enviar y recoger datos desde el servidor hacia nuestro Smartphone.

Para probar el sistema se ha instalado el led en el Arduino y hemos podido comprobar como desde la aplicación podemos encender y apagar el led y a su vez pedir al servidor que nos indique el estado de este.

# Capítulo 4. DESARROLLO DEL PROYECTO

El objetivo principal de este proyecto es crear un sistema que permita leer datos constantemente del servidor y que la aplicación nos alerte con una notificación a nuestro dispositivo cuando alguno de los sensores alcance un valor determinado de manera que podamos detectar la subida de temperatura de la habitación en un caso de incendio, o la activación de un sensor de movimiento al detectar una persona en nuestra habitación.

Partiendo del sistema base planteado en el capítulo tres, nos encontramos con tres objetivos a cumplir para completar el sistema:

- Añadir notificaciones al sistema.
- El sistema tiene que leer constantemente los datos del servidor.
- El sistema tiene que funcionar aun cuando la aplicación se encuentre en background.

Para ello se va a solucionar cada objetivo por separado y finalmente se pondrán en conjunto.

## 4.1. Notificaciones

En un principio se estudió la posibilidad de generar las notificaciones desde Unity sin tener que acceder a Android de manera que podríamos simplificar bastante el sistema, pero estudiando la posibilidad se descartó debido a que si Unity se iba a encargar de programar las notificaciones debería estar la aplicación abierta. Ya que la intención de este proyecto es añadir notificaciones al sistema no tendría ningún sentido que nos alerten las notificaciones si estamos viendo el menú principal.

La opción escogida fue crear un Plugin de Android que pudiera ejecutarse incluso cuando la aplicación estuviera cerrada.

Para ello se diseñó una aplicación aparte en Unity capaz de llamar a funciones de un Plugin para programar notificaciones simples o repetición de notificaciones.

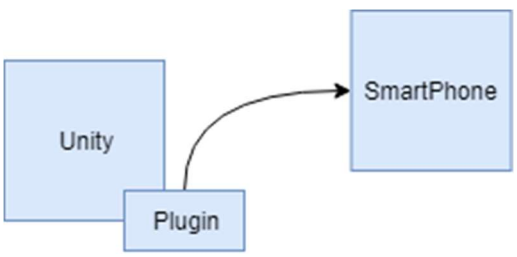


*Ilustración 17 Sistema de notificación*

Unity le pasa al Plugin, a partir de funciones vistas anteriormente (ver apartado 2.4), diferentes parámetros como, por ejemplo, el título de la notificación, el tiempo desde que se hace la petición hasta que surge la notificación, sonido, vibración, luces, etc.

Entonces el Plugin de Android programa dicha notificación con la siguiente clase.

```
AlarmManager am = (AlarmManager)currentActivity.getSystemService(Context.ALARM_SERVICE);
```

*Ilustración 18 Alarm manager*

Esta clase nos proporciona acceso a las alarmas del sistema que más adelante nos permitirá programar cuando los sensores recojan valores fuera de lo común.

Para esto utilizaremos la función “setExact” de la clase AlarmManager.

```
public void setExact (int type,
                      long triggerAtMillis,
                      PendingIntent operation)
```

*Ilustración 19 función setExact*

## 4.2. Lectura constante de datos

La intención de este objetivo es que la aplicación nos proporciones datos a tiempo real. Para ello tendremos que programarlo de alguna manera para que la aplicación se encargue de hacer peticiones GET al servidor constantemente.

Una de las posibles soluciones contempladas es hacer que la misma aplicación de Unity pida constantemente datos al servidor. De esta manera, Unity estará mandando peticiones de Get hacia el Plugin de Californium y conseguir así actualizar los datos de la aplicación constantemente.

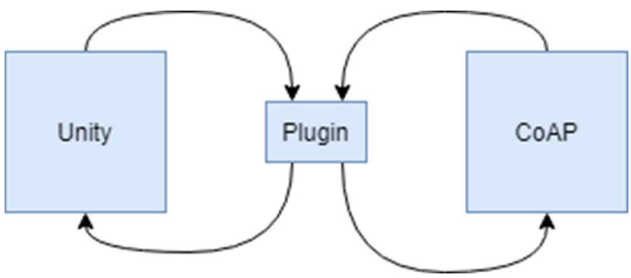


*Ilustración 20 Bucle Get en Unity*

Para conseguir esto se va a crear una función que sea la encargada de hacer las peticiones en bucle. Ver la siguiente figura:

```
// Usamos esta funcion al inicializar
IEnumerator Start()
{
    Debug.Log("sensors  Start");
    yield return new WaitForSeconds(0);
    // Llamamos a la funcion que va a entrar en bucle
    StartCoroutine("TestTemperature");

}

public IEnumerator TestTemperature()
{
    //Llamamos una primera vez a los sensores para que nos den la informacion
    string uritemp = coapManager.GetUri(IP, "temperature");
    coapManager.DoGet(uritemp);
    yield return new WaitForSeconds(1);
    string urilight = coapManager.GetUri(IP, "light");
    coapManager.DoGet(urilight);
    yield return new WaitForSeconds(1);
    string uriled = coapManager.GetUri(IP, "led");
    coapManager.DoGet(uriled);
    yield return new WaitForSeconds(1);
    for (; ; )
    {   //iniciamos el bucle para recolectar los datos constantemente
        uritemp = coapManager.GetUri(IP, "temperature");
        coapManager.DoGet(uritemp);
        yield return new WaitForSeconds(2);
        urilight = coapManager.GetUri(IP, "light");
        coapManager.DoGet(urilight);
        yield return new WaitForSeconds(2);
    }

}
```

*Ilustración 21 Función bucle Unity*

Es así como vamos a conseguir los datos del servidor constantemente, dejando un tiempo entre peticiones para poder recibir la respuesta y poder procesarla por parte de la UI y mostrarla por pantalla.

El principal problema de esta solución es que la aplicación deberá estar en foreground para poder hacer los request.

## 4.3. Aplicación en background

Como bien hemos comentado antes, hemos llegado a la solución de tener nuestra aplicación haciendo peticiones al servidor de CoAP y mandarnos notificaciones cuando alguno de los datos se saliera de un rango previamente establecido. El problema de esta solución es que la aplicación debía estar en foreground.

La intención de este objetivo es añadir a nuestra aplicación una solución para que esta funcione cuando este en background.

Hasta ahora Unity, era el encargado de hacer constantemente las peticiones, es por eso que en cuanto ponemos nuestra aplicación en background Unity se queda en pausa y perdemos la obtención de datos a tiempo real.

Para aplicar una solución a este sistema, vamos a trasladar el bucle encargado de hacer la petición de datos al Plugin, de manera que sea el Plugin el que haga todas las peticiones en background.

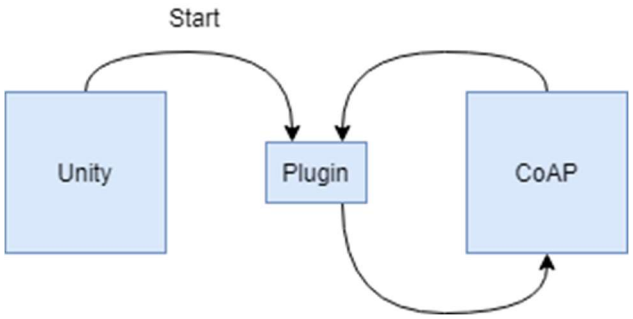


*Ilustración 22 Bucle Get en Plugin*

Al iniciar la aplicación será Unity el encargado de llamar directamente a la función que se va a encargar de manejar las peticiones en modo bucle. Es así como el Plugin cargara con todo el peso del sistema recolectando los datos y activando las notificaciones cuando sea preciso.

El inconveniente de esta solución es que, una vez hecha la petición inicial en Unity hacia el Plugin, Unity se queda bloqueado esperando a que termine un bucle infinito en el Plugin. Es por eso que cuando el Plugin le devuelve los datos a Unity para poder mostrarlos por pantalla, este no responde debido a que se encuentra bloqueado con la petición inicial.

Es por eso que al bucle programado en el Plugin deberemos formarlo con un Thread de manera que el Plugin ejecute de manera simultánea el bucle y no bloquee a Unity con la petición.

```
public void loop(final String ip, final String resource) {
    Thread th = new Thread(new Runnable() {
        public void run() {

            for(;;) {

                Log.d("unity", "inside loop " +resource);
                String uriparam = getUri(ip,resource);
                doGet(uriparam);
                try {
                    Thread.sleep(5000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

        }
    });
    th.start();
}
```

*Ilustración 23 Loop en Plugin con Thread*

Así es como finalmente damos con la solución para nuestra aplicación y cumplimos con todos los objetivos propuestos

Disponemos de una aplicación capaz de mostrarnos los valores de los sensores instalados en el servidor y capaz de alertarnos con una notificación siempre que estos sensores recolecten dato con un valor anómalo.

## 4.4. Posibles mejoras

Como posibles mejoras, se podrían añadir nuevos sensores como asimismo nuevas funcionalidades, de entre ellas, que la aplicación completamente apagada alertara cuando lea un dato fuera de lo común sin necesidad de iniciarla por primera vez.

Por otro lado, se podría añadir que leyera simultáneamente información de diferentes servidores de manera que podamos monitorizar diferentes entornos de una sola vez y/o poder elegir que sensores queremos que nos alerte.

# CONCLUSIONES

Día a día surgen nuevas ideas e inventos que nos abren puertas hacia lo desconocido y campos donde explorar. Sin duda es un mundo abierto donde se pueden desarrollar nuevas ideas y nos permiten crear aplicaciones como las de este proyecto.

A lo largo de este proyecto se han introducido nuevas tecnologías como el internet of things y softwares como CoAP y su funcionamiento. A su vez, también se han descrito diferentes entornos de trabajo con los que hemos podido llevar a cabo los objetivos del proyecto, como Unity, para la creación de una interfaz de una aplicación móvil, y Android, para la creación de los Plugins.

Gracias a estas tecnologías hemos podido llevar a cabo la aplicación de nuestro proyecto. Una aplicación capaz de recolectar datos de un servidor situado en una habitación remota y poder monitorizar los diferentes datos que nos pueden proporcionar los sensores instalados en este, teniendo así, visibilidad y capacidad de reacción cuando nos alerte ante un peligro como, por ejemplo, un incendio.

No cabe duda que el mundo de las aplicaciones está en auge, esta tecnología tan reciente está avanzando a pasos agigantados y como hemos visto, ahora está al alcance de todo el mundo. Sin darnos cuenta nos hemos visto envueltos de objetos capaces de conectarse a internet, donde hace uno 20 años era inimaginable, encontrar objetos los cuales se pueden conectar a internet está a la orden del día.

Es por eso que aplicaciones como las de este proyecto pueden ser de gran provecho, es sin duda un sector de mercado muy amplio el cual hemos visto que se podría aplicar tanto a uso doméstico, a uso personal o incluso a nivel industrial pasando desde el ámbito de la medicina hasta el sector agrícola.

# REFERENCIAS

[1] RFC 7252, Constrained Application Protocol (CoAP)
https://tools.ietf.org/html/rfc7252

[2] RFC 7252, Constrained Application Protocol draft-ietf-core-cap-03 (CoAP)
https://tools.ietf.org/id/draft-ietf-core-coap-03

[3] IOT
http://geekoders.com/index.php/areas-transversales-claves-para-el-exito-del-iot/

[4] The Internet of Things (IoT) – essential IoT business guide
https://www.i-scoop.eu/internet-of-things-guide/

[5] Ashton, K. That 'Internet of Things' Thing.
https://www.rfidjournal.com/articles/view?4986

[6] Dave Evans. The Internet of Things: How the Next Evolution of the Internet Is Changing Everything
https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[7] Costa, CA; Pasluosta, CF; Eskofier, B; da Silva, DB; da Rosa Righi, R Internet of Health Things: Toward intelligent vital signs monitoring in hospital wards.
https://www.sciencedirect.com/science/article/pii/S0933365717301367?via%3Dihub

[8] Meola, A. Why IoT, big data & smart farming are the future of agriculture.
https://www.businessinsider.com/internet-of-things-smart-agriculture-2016-10?IR=T

[9] RFC 6690, Constrained RESTful Environments (REST)
https://tools.ietf.org/html/rfc6690

[10] Colitti, W; Steenhaut, K; De Caro, N. Integrating Wireless Sensor Networks with the Web.
http://hinrg.cs.jhu.edu/joomla/images/stories/IPSN_2011_koliti.pdf

[11] RFC 3986, Uniform Resource Identifier (URI)
https://tools.ietf.org/html/rfc3986

[12] Californium
https://projects.eclipse.org/projects/technology.californium

# ANEXOS

## CODIGO SERVIDOR ARDUINO

### Función led

```
static const coap_endpoint_path_t path_led = {1, {"led"}};
static int handle_get_led(coap_rw_buffer_t *scratch,
const coap_packet_t *inpkt, coap_packet_t *outpkt, uint8_t id_hi, uint8_t id_lo) {
    return coap_make_response(scratch, outpkt,
    (const uint8_t *)&lightLed, 1, id_hi, id_lo, &inpkt->tok, COAP_RSPCODE_CONTENT,
    COAP_CONTENTTYPE_TEXT_PLAIN);
}

static int handle_put_led(coap_rw_buffer_t *scratch, const coap_packet_t *inpkt,
coap_packet_t *outpkt, uint8_t id_hi, uint8_t id_lo) {
    if (inpkt->payload.len == 0) {
        return coap_make_response(scratch, outpkt, NULL, 0, id_hi, id_lo, &inpkt->tok,
        COAP_RSPCODE_BAD_REQUEST, COAP_CONTENTTYPE_TEXT_PLAIN);
    }
    if (inpkt->payload.p[0] == '1') {
        lightLed = '1';
        digitalWrite(ledPin, HIGH);
        return coap_make_response(scratch, outpkt, (const uint8_t *)&lightLed, 1, id_hi,
        id_lo, &inpkt->tok, COAP_RSPCODE_CHANGED, COAP_CONTENTTYPE_TEXT_PLAIN);
    } else {
        lightLed = '0';
        digitalWrite(ledPin, LOW);
        return coap_make_response(scratch, outpkt, (const uint8_t *)&lightLed, 1, id_hi,
        id_lo, &inpkt->tok, COAP_RSPCODE_CHANGED, COAP_CONTENTTYPE_TEXT_PLAIN);
    }
}
```

### Función temperatura

```
static const coap_endpoint_path_t path_temp = {1, {"temperature"}};
static int handle_get_temp(coap_rw_buffer_t *scratch, const coap_packet_t *inpkt,
coap_packet_t *outpkt, uint8_t id_hi, uint8_t id_lo) {

    float sensorValue = analogRead(A2);
    //sprintf("Sensor value: %d", sensorValue);

    float sensorVoltage = (sensorValue*5000)/1023;

    float temperatureDegrees = sensorVoltage/10;

    char charVal[10];
    int num = (int)temperatureDegrees;

    float num2 = (temperatureDegrees - num) * 100;
    int num2int = (int)num2;
    sprintf(charVal, "%d.%d", num, num2int);

    return coap_make_response(scratch, outpkt, (const uint8_t *)&charVal, strlen(charVal),
    id_hi, id_lo, &inpkt->tok, COAP_RSPCODE_CONTENT, COAP_CONTENTTYPE_TEXT_PLAIN);
}
```

## Función light

```
static const coap_endpoint_path_t path_light = {1, {"light"}};
static int handle_get_light(coap_rw_buffer_t *scratch, const coap_packet_t *inpkt,
coap_packet_t *outpkt, uint8_t id_hi, uint8_t id_lo) {
    int sensorValue = analogRead(A1);
    char charVal[10];
    sprintf(charVal, "%d", sensorValue);

    return coap_make_response(scratch, outpkt, (const uint8_t *)&charVal, strlen(charVal),
    id_hi, id_lo, &inpkt->tok, COAP_RSPCODE_CONTENT, COAP_CONTENTTYPE_TEXT_PLAIN);
}
```

# CÓDIGO CLIENTE UNITY

## CoAP Manager

```
public class ResponseReceivedEventArgs : EventArgs
{
    public string Resource { get; set; }
    public string Data { get; set; }


    public void Awake()
    {
        Application.runInBackground = true;
    }
}
public class CoapManager : MonoBehaviour
{

    public void Awake()
    {
        Application.runInBackground = true;
    }

    public event EventHandler<ResponseReceivedEventArgs> ResponseReceivedHandler;
    private AndroidJavaObject coapClient;

    void Start()
    {
        try
        {
            coapClient = new AndroidJavaObject("com.coap.client.CoapClientManager");
        }
        catch (Exception e)
        {
            Debug.LogError(e.ToString());
        }
    }

    public string GetUri(string ip, string resource)
    {
        string res = coapClient.Call<string>("getUri", ip, resource);
        return res;
    }
```

```
public string GetUri(string ip, string resource)
{
    string res = coapClient.Call<string>("getUri", ip, resource);
    return res;
}

public void DoPut(string uri, string data)
{
   coapClient.Call("doPut", uri, data);

}

public void DoGet(string uri)
{
    coapClient.Call("doGet", uri);
}

public void GetResponse(string response)
{
    Debug.Log("GetResponse ");
    ResponseReceivedEventArgs args = new ResponseReceivedEventArgs();

    if (response != "Error")
    {
        string[] msg = response.Split('/');

        args.Resource = msg[0];
        args.Data = msg[1];
        OnResponseReceived(args);
    }
    else
    {
        args.Resource = "Error";
        args.Data = "Error";
        OnResponseReceived(args);
    }
}

protected virtual void OnResponseReceived(ResponseReceivedEventArgs e)
{
    EventHandler<ResponseReceivedEventArgs> handler = ResponseReceivedHandler;

    Debug.Log("OnResponseReceived ");
    if (handler != null)
    {
        handler(this, e);
    }
}
```

# CÓDIGO PLUGIN ANDROID

## CoAP

```
public void doGet(final String uriParam) {
    Log.d("unity", "do Get Plugin " + uriParam);
    Thread th = new Thread(new Runnable() {
        public void run() {
            URI uri = null;

            try {
                uri = new URI(uriParam);
            } catch (URISyntaxException var9) {
                Log.d("coap", "Invalid URI: " + var9.getMessage());
            }

            Request request = new GETRequest();
            Log.d("coap", "Get ");
            request.setURI(uri);
            Log.d("coap", "request ");
            request.enableResponseQueue(true);
            Log.d("coap", "execute ");

            try {
                request.execute();
            } catch (Exception var8) {
                Log.d("coap", "Failed to execute request: " + var8.getMessage());
            }

            try {
                Response response = request.receiveResponse();
                if (response != null) {
                    Log.d("coap", response.getOptions() + "");
                    Log.d("coap", response.getPayloadString());
                    String res = response.getPayloadString();
                    int pos = uriParam.lastIndexOf("/");
                    String resource = uriParam.substring(pos + 1);

                    UnityPlayer.UnitySendMessage("CoapManager", "GetResponse",
                    resource + "/" + res);
                    Log.d("unity", "Sensor: " + resource + " result: " + res);
                    float result = Float.parseFloat(res);
                    //Log.d("unity", "Sensor: " + resource + " result: " + result);
                    if(resource.equals("temperature") && result >= 32)
                    {   Log.d("unity", "Activate Notification");
                        UnityNotificationManager.SetNotification();
                    }

                } else {
                    UnityPlayer.UnitySendMessage("CoapManager", "GetResponse", "Error");
                    Log.d("coap", "No response received.");
                }
            } catch (InterruptedException var7) {
                UnityPlayer.UnitySendMessage("CoapManager", "GetResponse", "Error");
                Log.d("coap", "Receiving of response interrupted: " + var7.getMessage());
            }

        }
    });
    th.start();
```

```
public void doPut(final String uriParam, final String payload) {
    Log.d("unity", "do Put Plugin " +uriParam);
    Thread th = new Thread(new Runnable() {
        public void run() {
            URI uri = null;

            try {
                uri = new URI(uriParam);
            } catch (URISyntaxException var9) {
                Log.d("coap", "Invalid URI: " + var9.getMessage());
            }

            Request request = new PUTRequest();
            request.setURI(uri);
            request.setPayload(payload);
            request.enableResponseQueue(true);

            try {
                request.execute();
            } catch (IOException var8) {
                Log.d("coap", "Failed to execute request: " + var8.getMessage());
            }

            try {
                Response response = request.receiveResponse();
                if (response != null) {
                    Log.d("coap", response.getOptions() + "");
                    Log.d("coap", response.getPayloadString());
                    String res = response.getPayloadString();
                    int pos = uriParam.lastIndexOf("/");
                    String resource = uriParam.substring(pos + 1);

                    //UnityNotificationManager.SetNotification();
                    UnityPlayer.UnitySendMessage("CoapManager", "GetResponse", resource + "/" + res);

                    Log.i("coap", "Message sent");
                } else {
                    UnityPlayer.UnitySendMessage("CoapManager", "GetResponse", "Error");
                    Log.d("coap", "No response received.");
                }
            } catch (InterruptedException var7) {
                UnityPlayer.UnitySendMessage("CoapManager", "GetResponse", "Error");
                Log.d("coap", "Receiving of response interrupted: " + var7.getMessage());
            }

        }
    });
    th.start();
}

public String getUri(String ip, String resource) {

    return "coap://" + ip + ":5683/" + resource;
}
```

## Notificación

```
public static void SetNotification()
{
    ArrayList<NotificationAction> actions = new  ArrayList<NotificationAction>();
    int id = 1;
    long delayMs = 1000;
    String title = "CoAPP";
    String message = "Sensor Alert";
    String ticker = "Sensor Alert";
    int sound = 1;
    String soundName = null;
    int vibrate = 1;
    int lights = 1;
    String largeIconResource = null;
    String smallIconResource = "notify_icon_small";
    int bgColor = 16729156;
    String bundle = "com.serparsev.mynotificationtest";
    String channel = null;

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        if (channel == null)
            channel = "default";
        createChannelIfNeeded(channel, title, soundName, lights == 1, vibrate == 1, bundle);
    }

    Activity currentActivity = UnityPlayer.currentActivity;
    AlarmManager am = (AlarmManager)currentActivity.getSystemService(Context.ALARM_SERVICE);
    Intent intent = new Intent(currentActivity, UnityNotificationManager.class);

    intent.putExtra("ticker", ticker);
    intent.putExtra("title", title);
    intent.putExtra("message", message);
    intent.putExtra("id", id);
    intent.putExtra("color", bgColor);
    intent.putExtra("sound", sound == 1);
    intent.putExtra("soundName", soundName);
    intent.putExtra("vibrate", vibrate == 1);
    intent.putExtra("lights", lights == 1);
    intent.putExtra("l_icon", largeIconResource);
    intent.putExtra("s_icon", smallIconResource);
    intent.putExtra("bundle", bundle);
    intent.putExtra("channel", channel);
   Bundle b = new Bundle();
   b.putParcelableArrayList("actions", actions);
   intent.putExtra("actionsBundle", b);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
        am.setExact(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + delayMs,
    PendingIntent.getBroadcast(currentActivity, id, intent, PendingIntent.FLAG_UPDATE_CURRENT));
    else
        am.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + delayMs,
    PendingIntent.getBroadcast(currentActivity, id, intent, PendingIntent.FLAG_UPDATE_CURRENT));
}
```