Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA TELEMÁTICA

**Trabajo Fin de Grado**

# Desarrollo de sistema domótico y aplicación de gestión Android, conectados mediante plataforma IoT.



**Autor:** Víctor Ibarra Moreno

**Tutor/es:** Bernardo Alarcos Alcázar

2016

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

**GRADO DE INGENIERÍA TELEMÁTICA**

Trabajo Fin de Grado

# Desarrollo de sistema domótico y aplicación de gestión Android, conectados mediante plataforma IoT.

**Autor:** Víctor Ibarra Moreno

**Director/es:** Bernardo Alarcos Alcázar

**TRIBUNAL:**

**Presidente:** Antonio García Herráiz

**Vocal 1º:** David Fernández Barrero

**Vocal 2º:** Bernardo Alarcos Alcázar

**CALIFICACIÓN**: .....
**FECHA**: .....

# AGRADECIMIENTOS

Me gustaría dedicar este espacio a todas las personas que me han ayudado y mostrado su apoyo en la realización de este proyecto.
En primer lugar a mi familia, que tanto me ha apoyado durante este tiempo y a la que le debo todo. A mi madre por interesarse continuamente sobre el progreso de este proyecto atendiendo atentamente a mis explicaciones aunque no entendiese la mitad. A mi padre, por darme la libertad todos estos años para investigar, muchas veces con desastroso resultado, y que me ha permitido llegar hasta donde he llegado. Gracias por dejarme desmontar toda la habitación. Y a mi hermano, que ha soportado también mil y una pruebas fallidas de la aplicación, y me ha aportado muchas buenas ideas para incorporar en este proyecto.
También agradecer a Bernardo su gran interés mostrado durante el desarrollo de todo el proyecto, siempre ha estado ahí para darme su ayuda y consejo. Ha sido un gusto trabajar con él.
Agradecer a Miguel su inestimable ayuda con los problemas que han ido surgiendo en torno a la electrónica. Gracias a su ayuda ha sido posible arrancar definitivamente el desarrollo del sistema al comienzo complicado.
Agradecer también a Alicia y Elena sus consejos entorno al diseño y marketing del proyecto. Para el recuerdo queda el paseo nocturno por Madrid reflexionando sobre cuál sería el nombre definitivo de la aplicación.
Y por último, a mis amigos que me han ayudado a desconectar cuando era necesario, y que también han estado dispuestos a escuchar mis demostraciones del proyecto con la misma ilusión con la que yo lo hacía.

Muchas personas han puesto su granito de arena para llegar hasta aquí, a todas ellas GRACIAS.

# ÍNDICE

# ÍNDICE DE IMAGENES

## RESUMEN

Este proyecto desarrolla el escenario de una casa domótica, en la cual se recoge información a través de diferentes sensores, y permite la modificación de ciertos parámetros a través de actuadores. Todos estos datos son subidos a un servidor. Para ello emplearemos ThinkSpeak, una de las múltiples plataformas del Internet de las Cosas existentes en la actualidad (servidores IoT). Posteriormente, con el escenario ya creado, se desarrolla una aplicación en el sistema operativo Android para el uso generalista de todos los usuarios de esta plataforma. Su interfaz gráfica es sencilla intuitiva y flexible, con el objetivo que esta aplicación sea válida en múltiples escenarios.

## PALABRAS CLAVE

Aplicación, Android, Internet of the things, Domótica, Thinkspeak.

## SUMMARY

This project develops a scenery of a domotic home. On it, some sensors take information on different parameters, and then this parameters can be changed through actuators. All this data are stored in a server. In this project the server is ThinkSpeak, one of the many Internet of the Things platforms available now a days (Server IoT).
Once the scenario is created, an Android application is developed, for the general use of the users of this platform. With a friendly and flexible interface in order to be valid in multiples scenarios.

## KEYWORDS

Application, Android, Internet of the things, Domotic, Thinkspeak.

.

# RESUMEN EXTENDIDO

Con el desarrollo de las redes de comunicación y los dispositivos móviles, en la actualidad podemos obtener, almacenar y procesar gran cantidad de información a través de nuestros dispositivos móviles. Además, podemos dotar de cierta inteligencia a elementos que antes no la tenían, como es el caso de los edificios y hogares, así, podemos conseguir que estos se comporten de una manera u otra de forma óptima, y que se comuniquen con otros dispositivos, es el llamado "Internet de las cosas (IoT)".

El objetivo principal de este proyecto es la gestión y visualización ordenada de esta cantidad ingente de información, mostrándola al usuario de forma clara y simple, para que este pueda visualizar la información que realmente necesita.
Aquí radica el espíritu del proyecto, por un lado, demostrar la cantidad de información que podemos obtener, almacenar y acceder de forma remota, así como el uso que puede tener dicha información, para dotar de inteligencia a edificios y hogares. Por otro, dotar a un usuario medio de una interfaz clara para la visualización y programación de su hogar.

Por ello, el proyecto se puede dividir en dos partes claramente diferenciadas:

- Creación de un escenario que cuente con un sistema electrónico de recolección, modificación y subida de parámetros al servidor IoT.
- Desarrollo de una aplicación configurable y flexible para el uso de todos los usuarios de la plataforma IoT, que muestre de forma clara el estado de todos los parámetros. Además esta plataforma permitirá la ejecución de ciertas acciones y eventos.

## SISTEMA ELECTRONICO

Está compuesto por un microcontrolador Arduino Nano acompañado del módulo wifi ESP-8266 en su versión ESP1. A través de este podremos establecer las conexiones con el servidor IoT y subir los datos que el sistema vaya obteniendo. Estos datos se tomarán mediante múltiples sensores que estarán controlados por el propio Arduino. Sensores como DHT11, PIR, Sensores de efecto Hall, LDR… obtendrán los datos de los diferentes parámetros tales como humedad, temperatura, apertura de puertas y ventanas, movimiento….
Además como hemos dicho, el sistema podrá modificar el valor de ciertos parámetros, para ello, se emplearan algunos relés o leds IR.

En la actualidad existen múltiples servidores que ofrecen servicios de almacenamiento IoT, por ello en este proyecto se hace un pequeño estudio de algunos de ellos, valorando su popularidad, servicios, precios y la existencia o no de un ecosistema con aplicaciones para poder obtener, mostrar o modificar parámetros de dicho servidor.

Una vez realizado el estudio, se llega a la conclusión que **el servidor a emplear será ThingSpeak**, por lo que se adaptarán a él los diferentes envíos con los parámetros que realice nuestro microcontrolador.

Una vez creado el escenario y definido el servidor a emplear, se procederá al desarrollo de la aplicación en el sistema operativo Android, y que será denominada como: **DOMOTE**.

## DESARROLLO APLICACIÓN

Como decimos, la aplicación debe ser clara, simple y sencilla. Por lo que se introduce el concepto de **Sala** que busca añadir un punto de abstracción entre el funcionamiento de Thingspeak con Canales y Campos, y el razonamiento básico del usuario el cual no quiere más que controlar diferentes parámetros de sus salas en un edificio.
Con esta filosofía el usuario solo tendrá que introducir sus credenciales de Thinkspeak en la aplicación, crear las salas que desee y asignar los diferentes parámetros de su cuenta a estas salas.

Una vez configurada la aplicación, el usuario solo tendrá que pulsar sobre la sala que desee (Las cuales puede identificar mediante imágenes de la misma) para observar el valor actual de sus parámetros, así como gráficas que mostrarán el comportamiento de estos a lo largo del tiempo.

Además de visualización, el usuario podrá crear sus propios botones que pueden modificar estos parámetros. Los botones estarán basados en peticiones http que el usuario definirá, y que serán realizadas cuando este los pulse.
Puesto que las necesidades del usuario pueden ser muy diversas, se han definido tres tipos de botones:

- Botón simple
- Interruptor
- Barra de progreso

La aplicación contará con una sección de alarmas y notificaciones que permitirá al usuario definir ciertos eventos (como lanzar notificaciones en su dispositivo móvil o pulsar de forma automática botones ya creados) cuando se cumplen ciertas condiciones (Temporales, de conectividad o valores en parámetros).

Por último se establecerá un asistente de voz a la aplicación para que el usuario pueda interactuar con ella de una forma fácil y natural. Acciones como obtener el valor de un parámetro, mostrar la gráfica de este, mostrar salas o pulsar botones, serán posibles con solo decirlo.

# CAPÍTULO 1:

# INTRODUCCIÓN Y PLANTEAMIENTO DE OBJETIVOS

# CAPÍTULO 1: INTRODUCCIÓN Y PLANTEAMIENTO DE OBJETIVOS

## *1.1 Introducción*

Con el desarrollo de la tecnología, podemos dotar de una cierta inteligencia a elementos que antes no la tenían. Además podemos establecer una comunicación entre estos elementos (es el llamado internet de las cosas). Todo ello, con el objetivo de que estos elementos se comporten de una forma optima, y hacer la vida más cómoda.
Uno de los elementos que puede ser más interesante dotar de esta inteligencia, son los hogares y edificios. Este proyecto busca dotarlos de inteligencia mediante un sistema domótico. El sistema se encargará de **controlar y obtener los datos a través de sus sensores y actuadores**. Cada uno de estos datos (Temperatura, humedad, apertura de puertas, movimiento…) son los llamados parámetros del sistema, y serán subidos a una de las múltiples plataformas actuales del internet de las cosas (Plataforma IoT), para que estos puedan ser consultados posteriormente por los dispositivos que lo necesiten.

Llegados a este punto, surge el segundo objetivo de este proyecto. Actualmente la mayoría de personas disponen de un dispositivo móvil, nuestro objetivo es la creación de una aplicación para el sistema operativo móvil Android, que sea capaz de **obtener los datos que el sistema domótico** está subiendo a la plataforma IoT, además debe de permitir la creación de **botones, alertas y eventos** que permitan la interacción entre la aplicación y el sistema domótico.
Esta aplicación está pensada **para el uso de todos los usuarios** de plataforma IoT, por lo que debe ser muy flexible e intuitiva para que se adapte a las necesidades de cada uno de ellos.

Como vemos, el proyecto está claramente dividido en dos apartados, y del mismo modo se estructura esta memoria, en donde los dos primeros capítulos se expondrá el desarrollo del sistema domótico:

- Capitulo 2: Hardware del Sistema Domótico.
- Capitulo 3: Software del Sistema Domótico.

Posteriormente, se procederá a la exposición del desarrollo de la aplicación en Android, la cual se dividirá en tres capítulos:

- Capitulo 4: Domote, Descripción General.
- Capitulo 5: Interfaz de usuario.
- Capitulo 6: Estructura interna.

A continuación se procederá al análisis en profundidad de los objetivos que se marcan en cada apartado.

## *1.2 Objetivos y planteamiento de trabajo*

Como ya ha sido comentado, este trabajo se marca dos objetivos claros:
En primer lugar, el diseño e implementación tanto a nivel de hardware como de software, de un sistema domótico que realice las siguientes tareas:

- ✓ Recolección de datos y eventos que sucedan en el inmueble
- ✓ Modificación de algunos del estado de estos mediante diferentes actuadores.
- ✓ Envío de datos recogidos a plataforma IoT.
- ✓ Gestión de servicio que permita una comunicación bidireccional con el sistema.

Una vez desarrollado este sistema, ya tendremos el escenario preparado para nuestro segundo objetivo, el desarrollo de una aplicación en el sistema operativo móvil Android, para el uso generalista de todos los usuarios de la plataforma IoT. Por ello esta aplicación se marca los siguientes objetivos:

- ✓ Mostrar el estado actual de los parámetros
- ✓ Mostrar el estado anterior de los parámetros mediante gráficas configurables
- ✓ Permitir el establecimiento de Alarmas y Eventos.
- ✓ Permitir la creación de botones que permitan modificar el estado del sistema.
- ✓ Disponer de un asistente de voz, que permita el uso de la aplicación a través de comandos vocales
- ✓ Interfaz clara, agradable e intuitiva
- ✓ Flexibilidad en su configuración.

Llegados a este punto, se hace necesario la selección de la plataforma IoT (Internet of the Things) a utilizar, para posteriormente, analizando sus características, establecer los requisitos de una manera exacta que deberá cumplir nuestro sistema.

## *1.3 Selección de servidor IoT.*

Será el lugar donde nuestro sistema subirá y almacenará las muestras de los parámetros que se vayan tomando a lo largo del tiempo.
En la actualidad existen gran cantidad de servidores de este tipo, algunos de pago y otros gratuitos, o parcialmente gratuitos.
En nuestro caso, debemos de pensar en el posterior desarrollo de nuestra aplicación en Android y su posible público. Nos interesa un servidor orientado al gran público, con gran cantidad de usuarios y que no cuente en la actualidad con una aplicación de control y gestión en el sistema operativo Android.

Con estas condiciones, se han valorado muchas alternativas, y se ha podido comprobar como una gran cantidad de estos servidores, como puede ser Temboo[2] o TempoIQ[3] están orientados a un uso más industrial en los cuales el usuario final no estará en contacto con la plataforma, por lo que sus usuarios no parecen un público potencial para nuestra aplicación.

Por otro lado, se han descartado todos los servidores de pago, puesto que se considera que de nuevo este tipo de servidores, no serán los empleados por los potenciales usuarios de nuestra aplicación. De esta forma, quedan descartados importantes servidores como Xively[4] o mNubo[5].

También se ha valorado el desarrollo en la plataforma Beebotte[6], que si bien es cierto que ofrece gran cantidad de opciones y una interfaz sencilla, ha sido descartado debido a su escasa popularidad en comparación con otros servidores.

Finalmente, con estas condiciones se ha decidido el uso de la plataforma **Thinkspeak**[1]. Puesto que se trata de uno de los servidores más populares. Su interfaz es bastante sencilla, y cuenta con gran cantidad de guías paso a paso sencillas de realizar, lo cual puede facilitar la incorporación de usuarios que son a su vez usuarios potenciales de nuestra aplicación. Además es completamente gratuita, cuenta con una API abierta, y sistemas de procesado y muestra de datos, que posteriormente pueden ser usados por nuestra propia app. Por último, Thinkspeak tiene gran potencial en el análisis de datos, gracias a que cuenta con la herramienta de software matemático Matlab.

A pesar de todos estos puntos fuertes, también se han encontrado algunos puntos a mejorar y que deben ser comentados, tales como la limitación de un tiempo mínimo entre muestras de 15 segundos y la poca información que este aporta sobre los tipos de datos que almacenan sus campos.

Por otro lado, si analizamos en el mercado de aplicaciones de Android (Google Play[22]), podemos encontrar algunas aplicaciones ya desarrolladas para Thinkspeak como “ThingView” o “IoT ThingSpeak Data Monitor”, pero si analizamos su funcionamiento en profundidad, estas dejan mucho que desear, y su funcionalidad quedan muy por detrás de los objetivos que se marcan en Domote.

Por estos motivos, el servidor elegido para nuestro proyecto es **Thinkspeak.**

## *1.4 Requisitos del sistema*

Una vez definidos los objetivos y el orden de implementación, se procede a describir algunos de los requisitos del sistema:

SISTEMA ELECTRÓNICO:

- Constará de uno o varios microcontroladores que tomarán los datos de los sensores y se encargarán de transmitir esta información al servidor. Estos microcontroladores, estarán conectados a internet mediante la conexión a una red wifi.

- El sistema deberá recoger a través de sus sensores los siguientes datos (parámetros del sistema):

  - ✓ Temperatura
  - ✓ Humedad
  - ✓ Presencia
  - ✓ Apertura y cierre de puertas y ventanas
  - ✓ Luminosidad
  - ✓ Nivel de apertura de persianas

- Por otro lado, el sistema deberá realizar las siguientes acciones:

  - ✓ Encendido y apagado de luces
  - ✓ Apertura y cierre de persianas
  - ✓ Control de temperatura mediante sistema de Aire Acondicionado/Bomba de calor.
  - ✓ Lanzamiento de alarmas.

- El sistema domótico tomará los datos y los subirá al servidor Thinkspeak (Como se muestra en el esquema general en la imagen 1.4-1). Esta plataforma, nos permite almacenar el estado de gran cantidad de parámetros a sus servidores de forma gratuita.

En este punto, es donde surge el segundo objetivo, la creación de una aplicación en el sistema operativo Android para el uso de todos los usuarios de la plataforma Thinkspeak, que será llamada Domote.

DOMOTE:

- Esta aplicación deberá ajustarse a las necesidades de todos los usuarios, es decir la **flexibilidad en su configuración** debe ser una de sus principales cualidades.

- Para obtener esta flexibilidad, se establecerá la figura de las **salas**, que serán creadas por el usuario para añadirle los parámetros de su cuenta de Thinkspeak que él considere necesarios, y una imagen de la sala para crear una interfaz intuitiva. De esta forma, se añade una capa de personalización entre el estricto orden de Thinkspeak de campos y canales, y las necesidades del usuario.

A continuación se describen los puntos más importantes que la aplicación debe tener:

- FASE DE REGISTRO Y CONFIGURACIÓN

Domote contará con una primera fase de registro y configuración que será necesario realizar solo la primera vez que el usuario ejecute la aplicación y que posteriormente será almacenado en la memoria del sistema. Esta fase contará con 5 sencillos pasos:

1. Registro: Es la pantalla inicial del la aplicación. El usuario deberá introducir su nombre de usuario en Thinkspeak y su ApiKey.
2. Selección de canales: La aplicación obtendrá los canales y parámetros del usuario, y le serán mostrados, para que este seleccione los parámetros que desea utilizar en su configuración.
3. Creación de salas: En esta fase, simplemente se definirán las salas que deseen, y se le asignará una foto con la que será identificada de forma clara.
4. Asignación de parámetros: En esta fase, el usuario debe asignar los parámetros a cada sala
5. Configuración de parámetros: Se muestran los parámetros asignados por salas, en esta pantalla el usuario debe introducir el tipo de dato que se trata (Analógico o Digital), así como sus unidades.

- PANTALLA PRINCIPAL

Tras la sencilla configuración, y una animación de carga de la nueva configuración, el usuario accede a la pantalla principal. En ella, se observan las imágenes de las diferentes salas creadas, para acceder a los detalles de estas, así como dos icono en la barra de estado superior para la configuración de alarmas y el inicio del asistente de voz.

- PANTALLA DE SALA

Cuando el usuario pulsa sobre una de las salas en la pantalla principal, accederá a esta pantalla. En ella, se muestra el estado actual de los parámetros de la sala, si se pulsa sobre alguno de ellos, se despliega una lista con más detalles del parámetro, tales como

la fecha de la última muestra, el valor, o una gráfica con la evolución del parámetro a lo largo del tiempo.
Por otro lado, dentro de la pantalla de la sala, habrá un menú llamado Botones, en el cual el usuario podrá agregar los botones que considere necesarios.

- BOTONES

Hay diferentes tipos de botones, todos ellos están basados en peticiones http. El usuario podrá definir y configurar, de forma que el comportamiento de estos se adapten a sus necesidades.

- ALARMAS Y NOTIFICACIONES

El usuario puede a acceder a la pantalla Alarmas desde el menú principal de la aplicación. Allí, podrá crear todas las alarmas que desee. La estructura principal de las alarmas será la de Condición y Evento.

- ASISTENTE DE VOZ

Se trata de uno de los factores diferenciales de la aplicación. Desde cualquier parte de la aplicación, el usuario podrá pulsar, y decir la acción que desea.
Pedir el estado de un parámetro, solicitar gráficas, actualizaciones o modificar botones, se podrá realizar con tan solo pulsar el botón del asistente y pedirlo.

En la siguiente imagen, se puede apreciar el esquema general de funcionamiento de todo el proyecto:

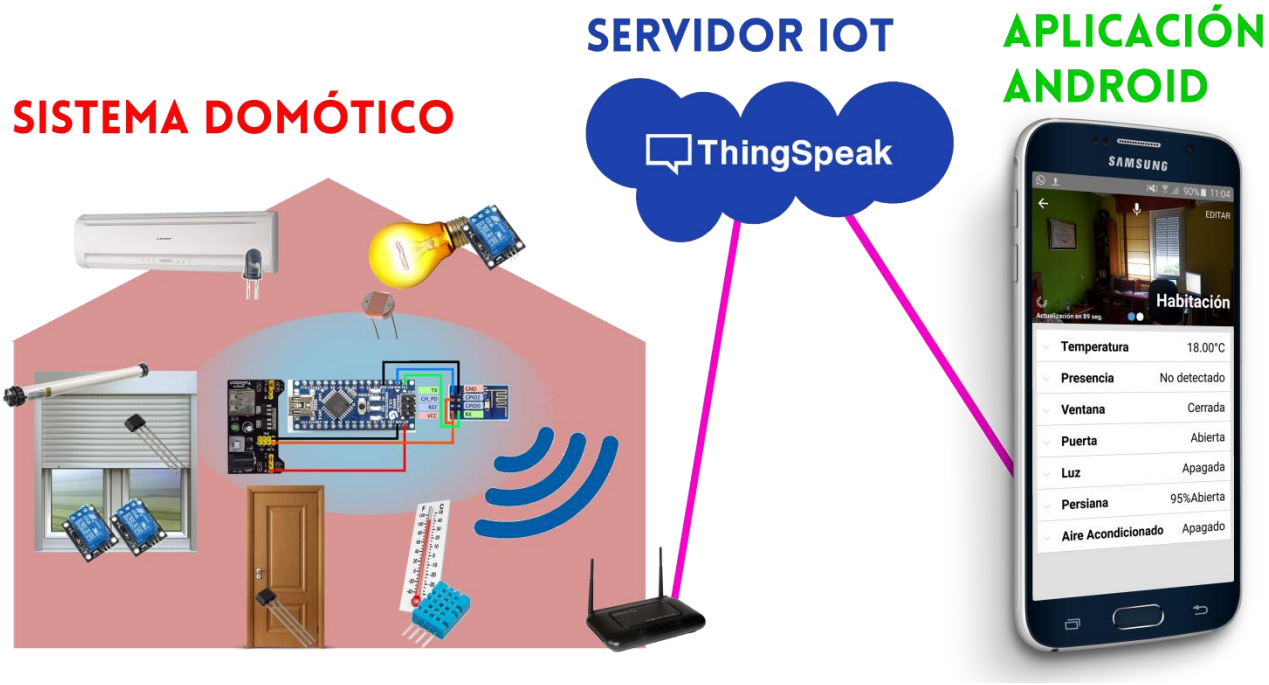



Imagen 1.4-1 Esquema General del proyecto

# CAPÍTULO 2:

# HARDWARE DEL SISTEMA DOMÓTICO

# CAPÍTULO 2: HARDWARE DEL SISTEMA DOMÓTICO

## 2.1 Introducción

En este capítulo se definirán de forma razonada todos los elementos físicos escogidos para realizar las labores que se han especificado en la introducción. Además, se mostrará el conexionado de todos ellos, para posteriormente poder desarrollar el software del sistema domótico.
Para comenzar, se comenzará con la definición del núcleo del sistema, el cual está compuesto por el microcontrolador y el módulo wifi. Para posteriormente definir los sensores y actuadores del sistema para obtener y modificar los parámetros necesarios. Por último, se definirá la conectividad entre todos los elementos del sistema.

## 2.2 Núcleo del sistema domótico

El primer elemento a determinar es el microcontrolador que se empleará, ya que este es el núcleo del sistema. En este caso la elección es clara, será un Arduino Nano. Este ha sido elegido por varios motivos, el principal es la facilidad de programación de la plataforma Arduino, y sus múltiples librerías. De esta forma se puede desarrollar el código de forma fácil y en un tiempo relativamente corto.
Dentro de los diferentes micros que nos ofrece la plataforma Arduino, se empleará la placa Arduino Nano[7], ya que con un uso optimizado, sus 2k bytes de memoria RAM son más que suficientes. De esta forma, sin un sobredimensionamiento excesivo del sistema, se consigue un microcontrolador a un precio bastante bajo[1].

A esta placa Arduino Nano, le acompañará el módulo wifi ESP-8266 en su versión ESP1. De nuevo, un módulo con características bastante simples y económicas que encajan con las necesidades del sistema domótico, y que permite reducir los costes[2].

En la imagen 2.2-1 se muestra el conexionado entre el modulo ESP-8266 y la placa Arduino Nano. Como se puede observar, se comunica mediante el puerto serie, los cuales se conectan cruzando sus pines RX y TX. Por otro lado, el módulo ESP recibe el nivel de tierra, así como alimentación a 3,3V. Por último el pin de chip-select del módulo (CH_PD) también es alimentado con 3,3V.

A través de esta comunicación serie, modulo y placa intercambian comandos AT. Es en este punto donde se puede observar una de las ventajas del uso de la plataforma Arduino. No será necesario el control de estos comandos AT ya que podemos emplear, una de las múltiples librerías existentes para el uso de estos módulos. Estas librerías nos permiten la abstracción a un nivel superior mediante el uso de funciones, las cuales se encargarán internamente del intercambio de estos comandos.

[1] Ver apartado PRESUPUESTO
[2] Ver apartado PRESUPUESTO

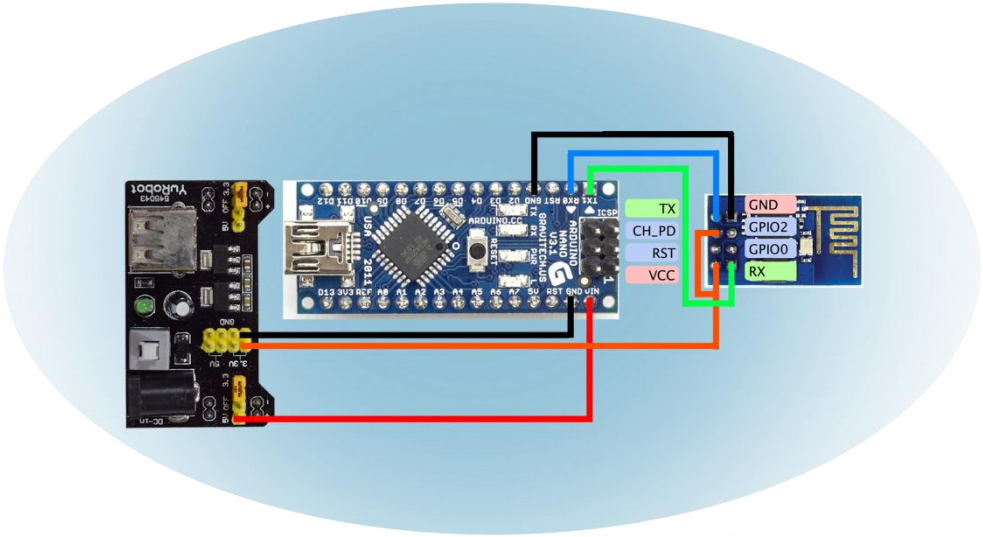



**Imagen 2.2-1 Conexionado Arduino-Esp**

Esta configuración que hemos elegido de Arduino Nano + ESP8266, es la elegida por el departamento de Automática para la realización de un sistema prototipado[3]. Este sistema cuenta con un Arduino Nano y el módulo ESP8266 ya conectados mediante la PCB del sistema. Además cuenta con algunos leds, switches y una pequeña placa de prototipado, como se puede ver en la Imagen 2.2-2.

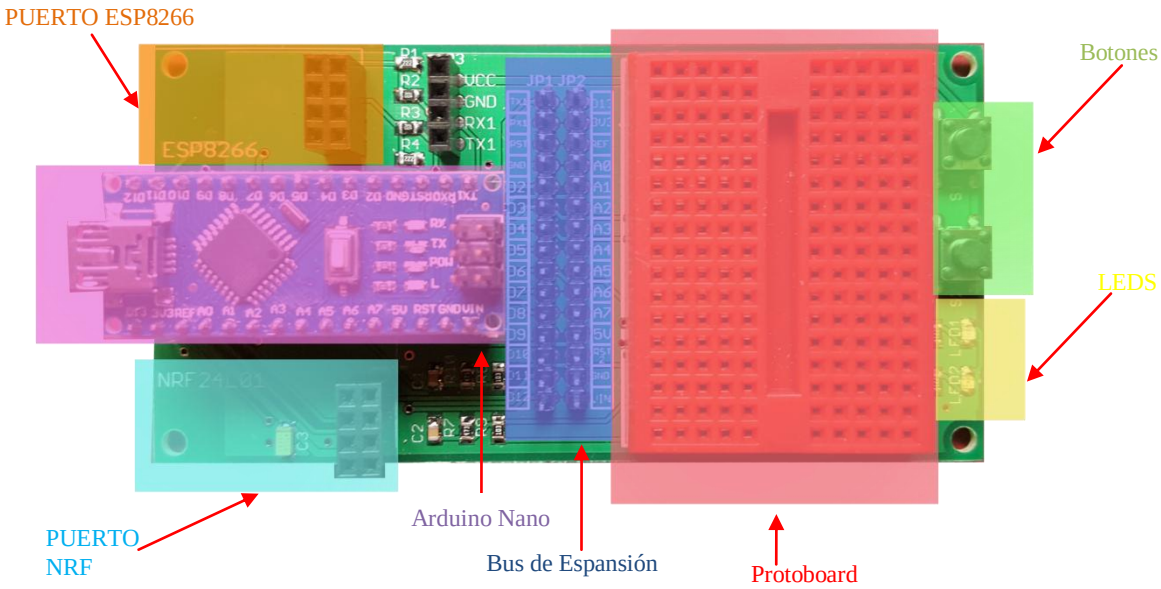



**Imagen 2.2-2 Sistema de prototipado del departamento de Automática**

[3] Ver esquemático del SISTEMA DE PROTOTIPADO en ANEXO

Puesto que este sistema se ajusta perfectamente a nuestras necesidades, se ha empleado este sistema de prototipado para el desarrollo del sistema domótico.

Como es lógico, siempre pueden surgir problemas una vez implementado el sistema, y que deben ser depurados. A continuación, se resumen algunos de los problemas que se han encontrado en esta primera versión del sistema de prototipado:

- Una vez montado, se ha podido observar que el módulo ESP-8266 requiere de una alimentación a 3,3V de hasta 215mA[8]. En un principio se intento de alimentar a través de la propia fuente que proporciona la placa Arduino a 3.3V como se puede ver en los esquemas de la placa[4]. Pero lamentablemente, esta no es capaz de dar toda la corriente que requiere el módulo ESP. Observándose una caída en la tensión y un apagado del módulo.

Como solución, se ha optado por la alimentación de la placa a través de una fuente externa. Si bien es cierto que estas no son excesivamente caras[5], si resultan incomodas a nivel de montaje del sistema, por ello se recomienda en futuras versiones del sistema de prototipado del departamento, la incorporación de un chip regulador de tensión capad de dar los 215mA a 3,3V.

- Otro problema detectado, es que como se puede ver en el esquema del sistema de prototipado[6], el módulo ESP se comunica con el Arduino directamente a través de su puerto serie hardware (TX0 y RX0). Si bien puede parecer lo correcto, pues este es el tipo de transmisión que se requiere. Surge un problema cuando intentamos reprogramar la placa. Para esta tarea, el programador se comunica por puerto serie (Pines RX0 y TX0) con la placa arduino, pines a los que ya está conectado el módulo ESP. De este modo, no es posible establecer una comunicación, pues el modulo ESP estará fijando unos niveles de tensión diferentes en el pin RX de la placa, lo cual hace imposible la transferencia de bits entre el programador y la placa, e impidiendo la reprogramación de esta. Este hecho, resulta muy incómodo en las labores de desarrollo del software del sistema puesto que en cada prueba se debe de desconectar el módulo ESP, programar la placa y volver a conectarlo. Además, se ha podido observar que con las continuas conexiones y desconexiones los pines del sistema se van dañando.

Se apunta para versiones futuras de la placa, no emplear el puerto serie hardware de la placa con el módulo ESP, y comunicarse mediante otros pines. Este hecho sería posible a través de la librería software serial. Esta solución ha podido ser probada con resultado positivo.
Otra posible solución sería el control del módulo ESP mediante el pin chip-select del mismo, de modo que este seria desactivado durante el proceso de reprogramación.

Todos estos problemas han sido notificados y las soluciones aquí expuestas serán desarrolladas en la siguiente versión del sistema de prototipado.

[4] Ver esquemático de sistema de prototipado en el apartado ANEXO
[5] Ver apartado PRESUPUESTO
[6] Ver esquemático de sistema de prototipado en el apartado ANEXO

## 2.3 Sensores

Una vez ya ha sido especificado el núcleo del sistema, es el turno de definir que sensores se van a emplear para la obtención de datos del sistema. Para ello, primero se listará la información a obtener, y que ha sido definida en las condiciones del sistema[7] como los parámetros del sistema domótico a controlar:

- Temperatura
- Humedad
- Presencia
- Apertura y cierre de puertas y ventanas
- Luminosidad
- Nivel de apertura de persianas

Una vez definidos los parámetros necesarios, a continuación se presentan las soluciones planteadas para obtener dicha información:

- **DHT11**: Se trata de un dispositivo digital de bajo coste[8] que cuenta con un sensor de **humedad** y de **temperatura**. Además cuenta con una biblioteca DHT en Arduino para una mayor simplicidad en su uso [11]. Su montaje es muy sencillo, siguiendo el esquema mostrado en su datasheet[10].

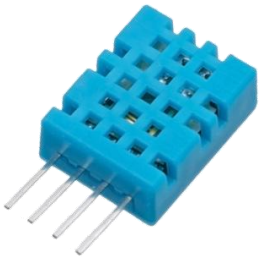


Imagen 2.3-1 Sensor DHT11

- **SENSOR PIR**: Se trata de un sensor de movimiento por infrarrojo. Un dispositivo digital de bajo coste[9] capaz de cubrir distancias de hasta 6 metros. Este dispositivo, devolverá un nivel alto si detecta **presencia**[12].

Imagen 2.3-2 Sensor PIR

[7] Ver apartado REQUISITOS DEL SISTEMA
[8] Ver apartado PRESUPUESTO.
[9] Ver apartado PRESUPUESTO.

- **SENSOR HALL U18:** Este dispositivo, se comporta como un biestable, devolviendo estados diferentes en función al campo magnético al que es expuesto[13]. Puede ser empleado para detectar **apertura de puertas y ventanas** solamente colocando un pequeño imán en la propia puerta, y el sensor en la posición cerrado.

Imagen 2.3-3 Sensor Hall

- **LDR**: Se trata de una fotorresistencia, la cual modifica su valor óhmico en función de la luminosidad que recibe[14]. Colocada en serie a una resistencia (como se muestra en el circuito de la imagen 2.3-5), el micro podrá medir la cantidad de luz que recibe solamente leyendo el voltaje que cae en la fotorresistencia a través de una de sus entradas analógicas.

Imagen 2.3-4 LDR

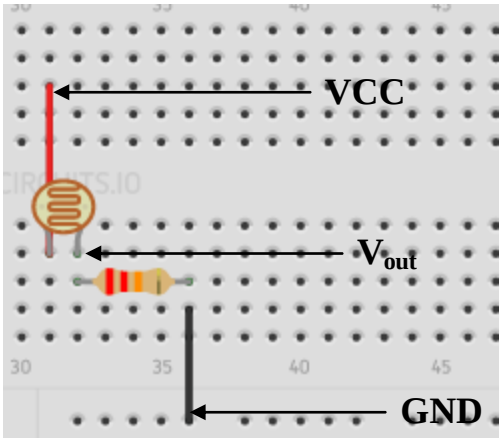



Imagen 2.3-5 Esquema conexión LDR

Por último, queda por definir la obtención del nivel de apertura de persiana, que será explicado con detenimiento más adelante[10].

[10] Ver apartado CONTROL DE PERSIANA

## 2.4 Actuadores

De nuevo, para definir los actuadores, se comenzará repasando los elementos a modificar requeridos en los requisitos del sistema[11]:

- Encendido y apagado de luces
- Apertura y cierre de persianas
- Control Aire Acondicionado/Bomba de calor.

En la mayoría de los casos, la solución es realizada por un mismo elemento:

- **Relés**: A través de este dispositivo, puede controlar la apertura y cierre de un circuito. Este funcionamiento básico puede ser empleado para **encender y apagar luces o controlar la alimentación de dispositivos**.


Imagen 2.4-1 Relé

Para el accionamiento del relé según su datasheet el sistema debe alimentarlo con una corriente de 90mA según su datasheet[15]. En este punto nos surge un problema, pues el microcontrolador tiene una excursión máxima de salida en sus pines de 40mA.
Para solucionar este problema, se ha incorporado un circuito amplificador como el que se muestra en la imagen 2.4-2:

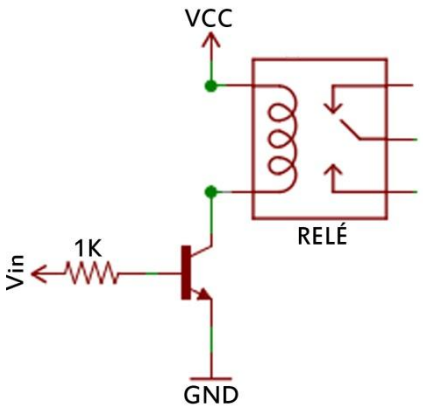



Imagen 2.4-2 Circuito amplificador

[11] Ver apartado REQUISITOS DEL SISTEMA

- **Led IR**: Mediante este diado, se realiza la misma función que tienen los mandos a distancia de muchos dispositivos. Con ello, podremos tomar el control de elementos como el **aire acondicionado/bomba de calor**[16]. Para su montaje se emplea una resistencia para fijar la corriente que circula por el circuito.

Imagen 2.4-3 Led IR

- **Motor persiana Jarolift SL-35**: Se trata de un pequeño motor que se introduce en el interior del tubo sobre el que se recoge la propia persiana. Su montaje y ajuste de finales de carrea es relativamente sencillo[19].

En función a su alimentación, este se moverá en un sentido de giro u otro como se observa en la figura 2.4-5-4.

Imagen 2.4-4 Jarolift SL-35

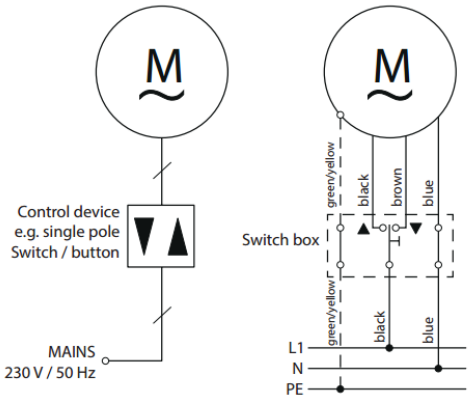



Imagen 2.4-5 Esquema Funcionamiento Motor

Por otro lado, para su control por parte del micro, se han empleado un par de relés simples, con el montaje mostrado en la imagen 2.4-6. El relé A será el encargado de la alimentación general del motor, y por otro lado, el relé B, será el encargado de seleccionar el sentido de giro del motor. De esta forma, el micro tiene control absoluto sobre la persiana.

Una vez obtenido el control de la persiana por parte del micro, este llevará la cuenta de en qué posición se encuentra la persiana, simplemente tiene que realizar una inicialización al empezar el sistema, por ejemplo, bajar completamente la persiana. A partir de este punto, se contara el tiempo que se emplea en cada movimiento, así se obtendrá la posición final de esta[12].

Por último, para un control manual de la apertura de persiana, se han empleado los dos botones con los que cuenta el sistema de prototipado del departamento que se está empleando en el sistema domótico.

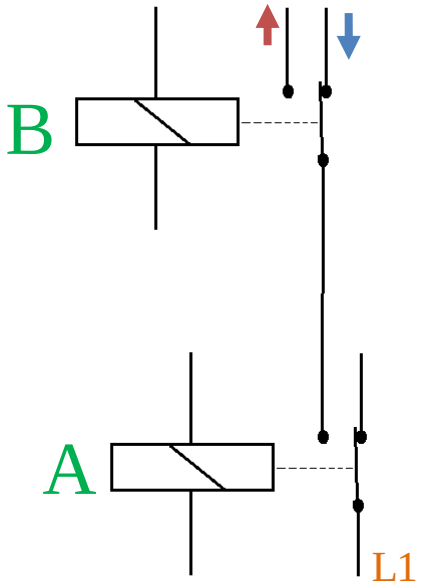



Imagen 2.4-6 Esquema Sistema de Control

## 2.5 Definición de entradas y salidas del sistema

Tras conocer los diferentes elementos de los que se disponen, se procede a listarlos, para posteriormente asignarle sus pines correspondientes en la placa Arduino:

| *Número* | *Elemento* | *Dispositivo* | *In/Out* | *Analog/Digital.* | *Pin Asignado* |
|---|---|---|---|---|---|
| **1** | Ventana | Sensor Hall | In | Digital | 12 |
| **1** | Puerta | Sensor Hall | In | Digital | A1 |
| **1** | Temp/Humedad | DHT11 | - | Digital | 9 |
| **1** | Luminosidad | LDR | In | Analógico | A0 |
| **1** | Presencia | PIR | In | Digital | 10 |
| **1** | Luces | Relé | Out | Digital | 11 |
| **2** | Persiana | Relé | Out | Digital | A3,A4 |
| **2** | Persiana | Pulsadores | In | Digital | 5,6 |
| **1** | Aire Acondicionado | Led IR | Out | Digital | 3 |

Imagen 2.5-1 Asignación Sensores y Actuadores

[12] Ver apartado CONTROL DE PERSIANA.

También por otro lado, se deben de tener en cuenta los pines que son necesarios para la comunicación y control del módulo ESP8266:

| *Número* | *Elemento* | *Pin Asignado* |
|---|---|---|
| **1** | Chip-Select | 2 |
| **2** | Comunicación | TX0, RX0 |

**Imagen 2.5-2 Asignación módulo ESP8266**

Por último, también serán necesarios el uso de algunos pines para extraer información sobre el estado en el que se encuentra el sistema domótico. Para ello, se han establecido los siguientes:

| *Número* | *Elemento* | *Pin Asignado* |
|---|---|---|
| **1** | Led Conexión TCP | 4 |
| **1** | Led Configuración | 3 |
| **2** | Monitor Serie | 7,8 |

**Imagen 2.5-3 Asignación de información**

En la imagen 2.5-4 se puede observar el conexionado final de todos los elementos empleados en el sistema domótico con los pines asignados a cada uno de ellos.

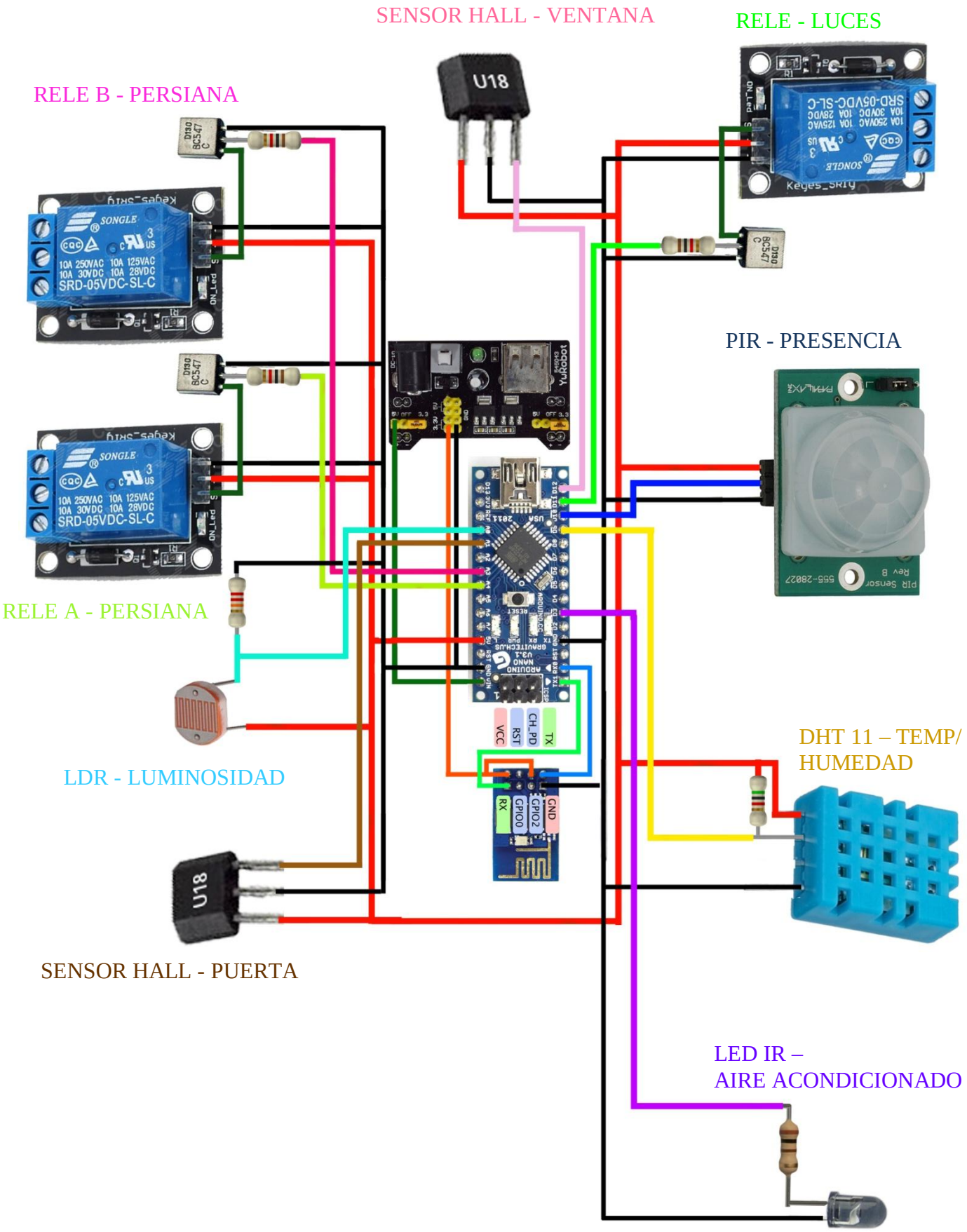



Imagen 2.5-4 Conexionado completo

## 2.6 Presupuesto

Todos los gastos de este proyecto han sido dedicados al hardware del sistema domótico. Por lo que resulta interesante, llegados a este punto, la realizar un pequeño resumen con todos los materiales que necesarios, para así hallar el coste de todo el sistema.
Los materiales electrónicos empleados para realizar este proyecto han sido relativamente económicos, si bien es cierto que se han optado por compras en comercios electrónicos chinos, que han ayudado a reducir notablemente el presupuesto[27].
Por otro lado, cabe destacar la utilización de elementos de hardware libre, que nos han ayudado a la reducción de costes tanto a nivel económico como de tiempo en el diseño del hardware del sistema domótico.

En la imagen 2.6-1 se detallan los elementos empleados, así como su precio:

| Producto | Cantidad | Precio Unitario | Precio Total |
|---|---|---|---|
| Arduino Nano | 1 | 2,41 € | 2,41 € |
| Modulo ESP-8266 1 | 1 | 6,56 € | 6,56 € |
| Relé | 3 | 1,05 € | 3,15 € |
| DHT11 | 1 | 1,23 € | 1,23 € |
| Sensor PIR | 1 | 1,23 € | 1,23 € |
| Sensor Hall U18 | 2 | 0,33 € | 0,66 € |
| Led IR | 1 | 0,80 € | 0,80 € |
| LDR | 1 | 0,80 € | 0,80 € |
| Motor Jarolift SL-35 | 1 | 45,00 € | 45,00 € |
| Rollo de cable 10m | 1 | 3,50 € | 3,50 € |
| Rollo Estaño | 1 | 1,00 € | 1,00 € |
| Cables Protoboard | 1 | 2,40 € | 2,40 € |
| Fuente de alimentacion 3,3V-5V | 1 | 1,03 € | 1,03 € |
| | | **Coste Total:** | **69,77 €** |

Imagen 2.6-1 Presupuesto del sistema domótico

Como vemos empleamos elementos bastante económicos, a excepción del motor de persiana. Obteniendo un coste final ajustado de 69,77€.

# CAPÍTULO 3:

# SOFTWARE DEL SISTEMA DOMÓTICO

# CAPÍTULO 3: SOFTWARE DEL SISTEMA DOMÓTICO

## 3.1 Introducción

En este capítulo, se explicará en profundidad el desarrollo de todo el software que controla la placa Arduino y todos los elementos hardware que han sido dispuestos en el capítulo anterior.
Toda la presentación, se realizará del mismo modo que este ha sido desarrollado. Comenzando por la comunicación básica entre el módulo ESP y la placa, además de configurar la plataforma Thingspeak de acuerdo a las necesidades del sistema.
Posteriormente se expondrá de forma detallada una versión básica del software, explicando sus funciones más importantes.
Por último, se añadirán elementos extra como el funcionamiento en modo servidor, control de persiana o aire acondicionado, que nos harán llegar al sistema finalmente implementado.

Para el desarrollo del código que controlará el sistema domótico, se ha empleado el IDE de Arduino en su versión 1.6.5. Un entorno de código abierto que si bien es cierto que resulta ligeramente limitado debido a la escasez de herramientas de las que dispone (Por ejemplo, se echa en falta la posibilidad de ocultar funciones dentro del código cuando la extensión de este comienza a ser considerable), resulta muy cómodo a la hora de cargar el código en la placa y depurar el funcionamiento de este gracias a su monitor serie.

## 3.2 Librería ESP-8266

Existen varias librerías ESP-8266, que realizan la misma tarea. En este proyecto se ha empleado la librería WeeESP8666 de Itead Studio[17]. Pues parece funcionar de forma correcta y el sistema se ha ido desarrollando con el uso de sus funciones.

Esta librería se encarga de la transmisión y recepción de comandos AT entre la placa Arduino y el módulo ESP. De esta forma, a través de una serie de funciones podemos indicar a la librería las acciones a realizar, y así abstraernos de los comandos AT intercambiados. Algunas de las funciones que emplearemos de esta librería serán:

- **bool joinAP (String ssid, String pwd)**

  Empleada para la conexión a un punto de acceso.

- **String getLocalIP ()**

  Devuelve la dirección ip actual del módulo.

- **bool enableMUX ()**

  Permite conexiones múltiples al módulo

- **bool createTCP (uint8_t mux_id, String addr, uint32_t port)**

  Establece conexión TCP con la dirección y puerto pasados.

- **recv (uint8_t mux_id, uint8_t *buffer, uint32_t buffer_size, uint32_t timeout=1000)**

  Esta función permite la recepción a través de una conexión TCP o UDP.

- **bool releaseTCP (uint8_t mux_id)**

  Finaliza una conexión TCP.

- **bool startServer (uint32_t port)**

  Inicia servidor TCP en el modulo, con el Puerto indicado.

Con el desarrollo del sistema, se ha podido comprobar cómo han surgido algunas limitaciones de memoria. Puesto que esta librería se encarga de la recepción y envió de mensajes a través de cadenas de caracteres (Comandos AT) esta puede llegar a consumir gran cantidad de memoria del micro, que precisamente, como se indicaba en el apartado de hardware[13] es el recurso más escaso de nuestro Arduino.
Es por ello, que ha sido necesario la optimización de la librería, para evitar el consumo excesivo de la memoria. Se ha podido comprobar como muchos objetos eran pasados por valor entre funciones, en lugar de por referencia, ocupando la memoria de forma inecesaria. Es por ello que se han modificado estos métodos, pasando los parámetros por referencia, optimizando así la cantidad de memoria empleada. A continuación se muestra un ejemplo:

```
bool ESP8266::createTCP(uint8_t mux_id, String addr, uint32_t port)
{
    return sATCIPSTARTMultiple(mux_id, "TCP", addr, port);
}
```

**Imagen 3.2-1 Código librería ESP8266 sin optimizar**

```
bool ESP8266::createTCP(uint8_t mux_id, String addr, uint32_t port)
{
    return sATCIPSTARTMultiple(mux_id, "TCP", &addr, port);
}
```

**Imagen 3.2-2 Código librería ESP8266 optimizada**

[13] Ver apartado NÚCLEO DEL SISTEMA.

## 3.3 Definición de canales y campos en ThingSpeak

Thingspeak trabaja por canales y campos. Un usuario puede crear múltiples canales y en cada canal puede definir hasta 8 campos. Cada uno de estos campos contendrá un parámetro del sistema domótico.
Como hemos visto anteriormente, es necesario almacenar 10 parámetros, por lo que es necesario definir dos canales. Esto no supondrá ninguna limitación como se podrá observar más adelante. En la imagen 3.3-1 se muestra el reparto de campos será el siguiente:

| *CANAL 1* | *CANAL 2* |
|---|---|
| *Luminosidad* | *Persiana* |
| *Humedad* | *ACTemperatura* |
| *Temperatura* | *AireAcondicionado* |
| *Presencia* | |
| *Ventana* | |
| *Puerta* | |
| *Luz* | |
| *SistemaLuzPresencia* | |

**Imagen 3.3-1 Reparto de canales y campos**

## 3.4 Filosofía de funcionamiento

En unas primeras versiones de la aplicación, se ha implantado un **sistema básico de funcionamiento**, en el que el sistema únicamente actualiza de forma periódica el valor de todos los parámetros en el servidor.
Este modo es el más sencillo en su programación. En algunos parámetros puede ser una buena opción tomar valores con una periodicidad constante para hallar posteriormente su media (Por ejemplo: la temperatura). En otros casos resulta altamente ineficiente, puesto que algunos de estos parámetros no varían a lo largo de mucho tiempo (Por ejemplo: apertura de puerta). Por lo cual no tiene sentido subir una gran cantidad de actualizaciones todas ellas con el mismo valor.

Por ese motivo, se ha pasado a un sistema de **funcionamiento por eventos**. Con esta filosofía, el sistema subirá la información de los parámetros cuando estos cambian. De esta forma, se reducen drásticamente el número de actualizaciones subidas, además se consigue una actualización casi inmediata del servidor, que por otro lado, repercutiendo en un mejor funcionamiento en la aplicación móvil que posteriormente será desarrollada.

A continuación se analizarán los parámetros del sistema para determinar cuáles deben de funcionar por eventos y cuáles por actualización periódica. Para ello se han definido dos categorías de parámetros:

- **Parámetros analógicos**: Son parámetros que no registran eventos como tal, sino que son continuos en el tiempo y su variación se produce lentamente. Además, resulta interesante obtener una gran cantidad de muestras de dicho parámetro para posteriormente obtener su media y poder así refinar su resultado final.
  Este es el caso de los parámetros: Luminosidad, temperatura y humedad.

En las imágenes 3.4-1 y 3.4-2 se muestran dos gráficas del parámetro temperatura con los mismos datos de origen, la segunda de ellas aplicando la media. En esta última, se puede apreciar de una forma mucho más clara la evolución del parámetro a lo largo del tiempo.

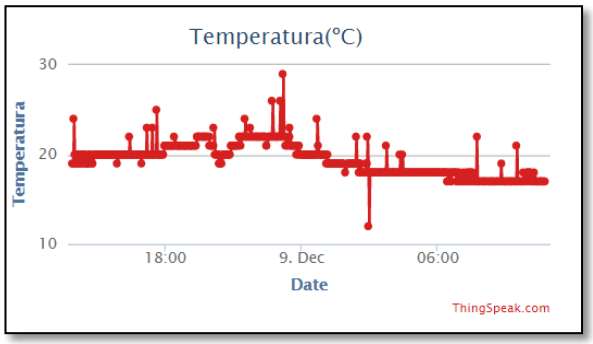



Imagen 3.4-1 Grafica de temperatura sin media

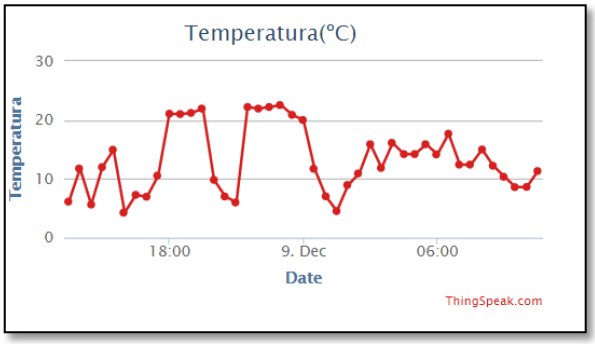



Imagen 3.4-2 Grafica de temperatura con media

- **Parámetros digitales:** (Presencia, ventana, luz, persiana, ACTemperatura y AireAcondicionado) Su comportamiento no es continuo en el tiempo, existen unos eventos claros en los cuales su valor cambia. Si estos eventos son detectados, y en dicho instante se actualiza el valor del parámetro, estará actualizado en todo momento el parámetro. Y con el menor número de muestras posible

## *3.5 Funcionamiento básico*

Una vez definida la filosofía de trabajo, se procederá a explicar el comportamiento básico del bucle del sistema.

Como se aprecia en la imagen 3.5-1, el bucle se divide en dos partes. Una primera dedicada a los parámetros analógicos que requieren una actualización periódica. Y una segunda parte, en la cual se chequea si se ha producido cualquiera de los eventos que

condicionan el estado de los parámetros digitales, si es así, la propia función de comprobación procederá a la actualización del mismo en Thingspeak.
En la imagen 3.5-1 se muestra el fragmento del código con el bucle básico.

```
void loop() {
   if(millis()-lastUpdateAnalogicosTime>updateValoresAnalogicosInterval){
   //comprobamos si el tiempo de actualizacion
  //de nuestros valores analogicos ha pasado
     if(millis()-lastEnvio>tiempoEntreEnvios){
    //comprobamos si el tiempo minimo entre envios ha sido superado
       lastUpdateAnalogicosTime=millis();//actualizo tiempo
       lastEnvio=millis();//actualizo tiempo
       enviarValoresAnalogicos();//enviamos valores analogicos
     }
   }
   //-------------CHECK DE EVENTOS
   comprobarPuerta();//chequeo si se ha producido evento en puerta
   comprobarVentana();//chequeo si se ha producido evento en ventana
   comprobarLuces();//chequeo si se ha producido evento en luces
   comprobarPresencia();//chequeo si se ha producido evento en presencia
}
```

Imagen 3.5-1 Bucle básico

A continuación se pasará a describir el funcionamiento de algunas de las funciones empleadas en este bucle básico.

## 3.6 Chequeo de eventos

Existe una serie de funciones que se dedican a chequear si un evento se ha producido. Cada una de ellas dedicadas a un parámetro en concreto. Su estructura es muy similar, es por ello que en este apartado, sé describirá solo una de ellas. De esta forma será posible el entendimiento de todas ellas si se consulta el código el código completo[14].

Como podemos observar en la imagen 3.6-1 se trata de un código muy sencillo. En primer lugar se consulta si ha pasado el tiempo mínimo entre actualizaciones del servidor. (15 segundos, tiempo que como se ha dicho en apartados anteriores, viene dado por ThingSpeak).

Si este tiempo ha pasado, se comprueba si existe diferencia entre el valor anterior del parámetro y el valor actual, si esto es así, quiere decir que se ha producido un evento, y se procede a la actualización del campo tanto en el propio sistema como en el servidor.

[14] Ver CÓDIGO SISTEMA ELECTRÓNICO en apartado ANEXO.

El resto de código está dedicado a la preparación de las variables para la función updateParametroTS(int * numCampo, String* valor, int cantidadCampos, int canal), que será la encargada de actualizar el servidor. Esta función será explicada en profundidad en el siguiente apartado.

```
void comprobarVentana(){
  if(millis()-lastEnvio>tiempoEntreEnvios){
  //compruebo que han pasado el tiempo subiciente
  //para un nuevo envio
    if(estadoVentana!=digitalRead(VENTANA)){
    //compruebo si ha variado el estado
      lastEnvio=millis();//actualizo tiempo de envio
      estadoVentana=digitalRead(VENTANA);//actualizo estado
      int parametro[]={5};//se trata del campo 5
      if(estadoVentana==HIGH){//si ha sido abierta envio un uno
        String* textos[]={&uno};
        mon.println(F("Ventana abierta"));//muesto evento en monitor
        updateParametroTS(parametro,textos,1,1);
        //funcion para actualizar servidor
      }else{//ha sido cerrada
        String* textos[]={&cero};//envio 0
        mon.println(F("Ventana cerrada"));//muestro evento en monitor
        updateParametroTS(parametro,textos,1,1);//funcion para actualizar
      }
    }
  }
}
```

**Imagen 3.6-1 Fragmento de código, función comprobarVentana().**

## 3.7 Función updateParametroTS()

Esta función, se encarga de preparar el mensaje HTTP completo (Cabecera + Cuerpo) que será posteriormente enviado al servidor.
Como se puede ver en la imagen 3.7-1, para realizar esta tarea recibe como parámetros un puntero a entero con los números de los campos que se van a actualizar, un puntero a String con los valores de estos parámetros, un entero con el número de campos a enviar, y por último un entero indicando el número de canal.

Con todo ello, la función comienza construyendo el cuerpo del mensaje que tendrá la forma que se muestra a continuación:

```
field1=73&field3=1&field5=12
```

Una vez obtenido el cuerpo del mensaje, se comienza a añadir al buffer (en el cual se almacenará todo el mensaje) las cabeceras del mensaje HTTP, mientras que se va calculando la longitud total que tendrá dicho mensaje.

Una vez construido el mensaje, se llama a la función enviarHttp(int Longitud), que se encargará de su envío y que será comentado en el siguiente apartado.

A continuación se muestra el código de la función updateParametrosTS(), y un ejemplo del mensaje HTTP resultante:

```
POST /update HTTP/1.1
Host: api.thingspeak.com
Connection: close
X-THINGSPEAKAPIKEY: [redacted]
Content-Type: application/x-www-form-urlencoded
Content-Length: 50

field1=781&field2=50.00&field3=22.00&headers=false
```

Imagen 3.7-1 Mensaje resultante

```
int updateParametrosTS(int numerosCampos[], String* valores[],int canal,
                       int cantidadParametros){
    String cuerpo="";//string con el cuerpo del mensaje
    for(int i=0;i<cantidadParametros;i++){
      if(i!=0){
        cuerpo=cuerpo+"&";//añadimos caracter "&" entre parametros
      }
      cuerpo=cuerpo+F("field");
      cuerpo=cuerpo+String(numerosCampos[i]);
      cuerpo=cuerpo+F("=");
      cuerpo=cuerpo+*valores[i];
    }
    cuerpo=cuerpo+F("&headers=false");
    //se solicita no recibir cabeceras en la respuesta
    //estas son las cabeceras del mensaje
    String mensaje=F("POST /update HTTP/1.1\nHost: api.thingspeak.com\n
                     Connection: close\nX-THINGSPEAKAPIKEY: ");
    String mensaje2=F("\nContent-Type: application/x-www-form-urlencoded\n
                      Content-Length: ");
    String mensaje3=F("\n\n");
    String mensaje4=F("\n");
    String numero1=String(cuerpo.length());
    //en la cabecera debe de constar la longitud del cuerpo
    int longitudMensaje=0;//esta variable sera la longitud total del mensaje
    //a continuación se porcede a ir añadiendo cada una de las partes
    //del mensaje al buffer y a calcular su longitud total
    anadirABuffer(longitudMensaje,&mensaje);
    longitudMensaje=longitudMensaje+mensaje.length();
    if(canal==1){//añadimos apikey
      anadirABuffer(longitudMensaje,writeAPIKeyCH1,16);
      //el tamano de apikey es 16
      longitudMensaje=longitudMensaje+16;
    }else{
      anadirABuffer(longitudMensaje,writeAPIKeyCH2,16);
      longitudMensaje=longitudMensaje+16;
    }
    anadirABuffer(longitudMensaje,&mensaje2);
    longitudMensaje=longitudMensaje+mensaje2.length();
    anadirABuffer(longitudMensaje,&numero1);
    longitudMensaje=longitudMensaje+numero1.length();
    anadirABuffer(longitudMensaje,&mensaje3);
    longitudMensaje=longitudMensaje+mensaje3.length();
    anadirABuffer(longitudMensaje,&cuerpo);
    longitudMensaje=longitudMensaje+cuerpo.length();
    anadirABuffer(longitudMensaje,&mensaje4);
    longitudMensaje=longitudMensaje+mensaje4.length();
    mon.print(F("Enviando datos...."));//mostramos informacion en monitor
    return enviarHTTP(longitudMensaje);//procedemos al envio del mensaje ya formado
}
```

**Imagen 3.7-1 Fragmento de código función updateParametroTS()**

## 3.8 Función enviarHTTP()

Esta función es la encargada de gestionar el envió del mensaje HTTP al servidor.
Como se puede ver en la imagen 3.8-1, esta función comienza abriendo una conexión TCP con el servidor ThingSpeak. Si esta conexión se realiza perfectamente, se procede al envío del mensaje almacenado en el buffer.
Posteriormente, pasa a la espera de recibir un mensaje de confirmación del envío. Si este es recibido, se puede afirmar que la conexión se ha realizado perfectamente.
En caso de que se haya producido un error, se incrementa en uno el valor del contador de errores.
Por último, como método de seguridad, esta función comprueba el numero de errores seguidos que se han producido, si este número llega a 10, el programa llama a la función conectarARedWifi(), para reiniciar la conexión.

```
//Esta funcion se encarga del envio del mensaje HTTP a thingspeak
int enviarHTTP(int longitud)
{
  uint32_t len=-1;
  if (wifi.createTCP(1,thingSpeakAddress, 80))//establecemos conexion TCP con TS
  {
    digitalWrite(LEDTCP, HIGH);//encendemos el led TCP
    if(wifi.send(1,(const uint8_t*)buffeer,longitud)){
      //enviamos contenidon del buffer
      numErroresContinuos=0;//reinicimos contador de errores
      mon.print(F("ENVIOK...."));//mostramos avance en monitor
      len = wifi.recv(1,buffeer, sizeof(buffeer), tiempoEsperaRespuesta);
      //procedemos a recibir la respuesta
      mon.print(F("RECIVOK...."));//mostramos avance en monitor
      wifi.releaseTCP(1);//liberamos conexion TCP
      mon.println(F("OK"));
    }else{//Si se produce un error en el envio del mensaje
      mon.println(F("Error al enviar el mensaje"));
    }
    digitalWrite(LEDTCP, LOW);//apago el led tcp
    return len;//devolvemos la longitud del mensaje recibido
  }else{//si se produce un error en el establecimiento de la conxion TCP
    numErroresContinuos++;//añadimos un error al contador
    mon.print(F("Parece que no hay conexion, fallo:"));
    mon.println(numErroresContinuos);
  }
  if(numErroresContinuos==10){
  //en caso de se hayan realizado 10 errores continuos
      conectarARedWifi();//se vuelve a establecer conexion wifi
      numErroresContinuos=0;//y se reinicia el contador
      String tweetInicio=F("Reestablecimiento de conexión");
      enviarTweet(&tweetInicio);//se envia un tweet como log del error
  }
  return -1;
}
```

Imagen 3.8-1 Fragmento de código, función enviarHTTP()

Una vez explicado el funcionamiento básico del sistema, vamos a proceder a añadirle al sistema la funcionalidad de servidor, de forma que este pueda recibir ciertos comandos y posteriormente realizar ciertas acciones con ellos.

## *3.9 Servidor*

El modo servidor se pondrá a la escucha en el puerto 80 del módulo (pudiendo ser otro si se prefiere). A este puerto podrán conectarse clientes para ejecutar ciertos comandos en el sistema.
A continuación se detallan los comandos con los que funcionará el sistema:

- **"Luces on-off"**: comando para encender o apagar las luces
- **"Luzpresencia: ON"**: comando que activa el modo Luz-presencia en el sistema
- **"Luzpresencia: OFF"**: comando que desactiva el modo Luz-presencia en el sistema
- **"Persiana:[numero]"**: comando que modifica el nivel de apertura de la persiana al número pasado
- **"AC on-off"**: comando que enciende o apaga el aire acondicionado
- **"ACTemp:[numero]"**: comando que modifica la temperatura objetivo del aire acondicionado

Para comenzar a establecer el servidor del sistema, el primer paso es solicitar al módulo ESP8266 que active su modo servidor TCP en un puerto determinado. Esto se realizará mediante la función:

```
startTCPServer(80);
```

Esta llamada se realizará en la función setup()[15] que será ejecutada al inicio del sistema.
Una vez iniciado el modo servidor del módulo, se comprobará periódicamente si ha llegado un nuevo cliente, esta comprobación se encargará de realizarla la función: *comprobarPeticionesServidor()* que será comentada a continuación.

### 3.9.1 Función comprobarPeticionesServidor()

Como se puede observar en la Imagen 3.9.1-1, lo primero que comprueba esta función es si ha llegado un nuevo cliente. Para ello, se llama a la función recv(), esta devolverá la longitud de la cabecera del mensaje recibido, siendo esta 0 si no ha llegado ningún cliente.

[15] Ver CÓDIGO DEL SISTEMA ELECTRÓNICO en el apartado ANEXO

```
//Esta funcion comprueba si se ha producido la entrada de algun cliente en nuestro
//servidor TCP del modulo si esto es así, se atiende dicha entrada
void comprobarPeticionesServidor(){
  uint32_t len;//longitud de mensaje recibido
  uint8_t mux_id;//canal por el que se recibe
  int8_t resultado=-1;//resultado final de recepcion
  //Nos ponemos a la escucha durante un tiempo para ver si ha llegado algun cliente
  len = wifi.recv(&mux_id, buffeer, sizeof(buffeer), tiempoEsperaConexiones);
  if (len > 0) {//si el cliente ha llegado, recibiremos la cabecera de su mensaje
    len = wifi.recv(&mux_id, buffeer, sizeof(buffeer), tiempoEsperaRecepcionMensaje);
    //procedemos a recibir el cuerpo del mensaje
    if(len>0){//si la del cuerpo del mensaje recibido es mayor a 0
      mon.print(F("LLega peticion canal:"));//mostramos estado en monitor
      mon.print(mux_id);
      mon.print(F("-------"));
      if(millis() - lastConnectionReceived > (long)tiempoFiltroConexiones){
        //filtramos por limitacion de tiempo entre conexiones
        mon.print(F("---Tiempo-OK,"));
        lastConnectionReceived = millis();//nuevo timestamp del filtro
        String comando=getString(len);
        //llamamos a la funcion getString que nos devolverá el cuerpo del mensaje
        resultado=ejecutarComandos(&comando);
      //posteriormente enviamos dicho cuerpo al la funcion ejecutarComandos
      //si se ha recibido un comando correcto, nos será devuleto un numero mayor que 0
        if(resultado>0){//mostramos evolucion en monitor
          mon.println(F("-EXITO"));
        }else{
          mon.println(F("-ERROR"));
        }
      }else{//en caso de que no pase la limitacion temporal, es bloqueado
        mon.println(F("Peticion Bloqueada por Tiempo"));
      }
      wifi.releaseTCP(mux_id);//la conexion tcp es cerrada, liberando nuestro servidor
      //en algunos casos es necesario enviar a thingspeak un cambio de estado
      if(resultado==2){
        enviarSistemaLuzPresencia();
        //se ha modificado el estado del sistema Luz-Presencia
      }else if(resultado==3){//se ha encendido o apagado el aire acondicionado
        enviarIR();//se envia señal IR
        enviaracEncendido();//se envia cambio a TS
      }else if(resultado==4){//se ha modificado la temperatura del aire
        enviarIR();//se envia señal
        enviarACTemp();//sen envia cambio a TS
      }
    }
  }
}
```

Imagen 3.9.1-1 Fragmento de código, función comprobarPeticionesServidor()

En caso de que la longitud sea mayor que 0, continúa con el proceso de recepción del cliente, con la solicitud del cuerpo del mensaje, de nuevo con la llamada a la función recv, y de nuevo se comprueba su longitud para que esta sea mayor a 0.

Una vez obtenido el cuerpo del mensaje, que ha sido almacenado en el buffer, se pasa por un filtro temporal que ha sido fijado para evitar un número excesivo de conexiones en el sistema. Si esta conexión cumple con la condición impuesta por el filtro, se procede a convertir el cuerpo de mensaje contenido en un array de caracteres, a una variable String (pues será más cómodo su manipulación en futuras operaciones). Posteriormente es llamada la función int ejecutarComando(String*) que se encargará de comprobar el comando recibido para luego ejecutarlo, y que será explicada en el siguiente apartado. Esta función devolverá un entero con el resultado del comando que ha sido enviado.
Tras la ejecución del comando, se procede al cierre de la conexión TCP con el cliente mediante la función releaseTCP(int).
Por último, la ejecución de algunos de estos comandos requieren de actualización de los parámetros del servidor, por lo que se comprueba si se trata de alguno de esos casos y de ser así, se envía su actualización

### 3.9.2 Función ejecutarComandos()

Como ya ha sido adelantado, esta función se encarga de reconocer el mensaje que se ha recibido, y comprobar si se ajusta a la forma de uno de los comandos definidos en el sistema, si esto es así, se lanzan las acciones de dicho comando.

Como se puede ver en el código de la imagen 3.9.2-1, cada comando produce una acción:

- "Luces on-off": llama a la función cambiar luces, la cual cambiará el estado del relé
- "Luzpresencia: ON": establece el parámetro sistemaLuzPresencia como activo
- "Luzpresencia: OFF": establece el parámetro sistemaLuzPresencia como desactivo
- "Persiana: [numero]": Calcula el tiempo que la persiana debe de estar en movimiento y la dirección del mismo. Posteriormente inicial el movimiento de la persiana en modo automático. (El funcionamiento de la persiana será visto en profundidad más adelante)[16].
- "AC on-off": cambia de valor la variable acEncendido.
- "AC temp: [numero]": establece el valor [número] a la variable acTemperatura.

Tras la ejecución del comando, la función devolverá un número indicando el comando que ha sido ejecutado. Como hemos visto en el apartado anterior este valor es empleado para realizar ciertas acciones tras el cierre de la conexión con el cliente, tales como la actualización del servidor, o el envió de señales IR.

[16] Ver apartado CONTROL DE PERSIANA.

```
//Esta funcion recibe el cuerpo de los mensajes recibidos
//y determina si se trata de alguno de los comandos definidos
int ejecutarComandos(String* comando){
  mon.print(F("Comando:"));
  //Vamos comparando el comando recibido con los definidos
  //si se trata de alguno de ellos, entrará en su condicion
  if(*comando==F("Luces on-off")){//Comando para encender/apagar luces
    mon.print(F("LUCES"));
    cambiarLuces();
    return 1;
  }else if(*comando==F("Luzpresencia: ON")){
    //Comando para activar el sistema de Luz-presencia
    sistemaLuzPresencia=true;
    mon.print(F("LUZ PRESENCIA ON"));
    return 2;
  }else if(*comando==F("Luzpresencia: OFF")){
    //Comando para desactivar el sistema de Luz-presencia
    sistemaLuzPresencia=false;
    mon.print(F("LUZ PRESENCIA OFF"));
    return 2;
  }else if(comando->substring(0,9)==F("Persiana:")){
  //Comando para modificar la apertura de persiana ejm:"Persiana:24"
    int posicion=comando->substring(9).toInt();//obtenemos posicion solicitada
    long tiempo=6250+(long)posicion*183;//calculamos el tiempo relativo solicitado
    if(tiempo>tpersiana+100){
    //si el tiempor relativo es mayor  al actual, debemos de subir la persiana
      modoAuto=true;//ponemos en modo auto
      bajada=false;//seleccionamos sentido de movimiento
      tautomatico=millis()+tiempo-tpersiana;
      //fijamos hasta cuando tiene que estar en movimiento
      tmovimiento=millis();//fijamos el tiempo inicial
      digitalWrite(MOTORDIRECCION, LOW);//estableciemos rele de subida
      digitalWrite(MOTORCONTROL, HIGH);//establecemos rele de alimentacion gral
    }else if(tpersiana>tiempo+100){
      //si el tiempor relativo es menor al actual, debemos de bajar la persiana
      modoAuto=true;//ponemos modo auro
      bajada=true;//seleccionamos sentido de movimiento
      tautomatico=millis()+tpersiana-tiempo;
      //fijamos hasta cuando tiene que estar en movimiento
      tmovimiento=millis();//fijamos el tiempo inicial
      digitalWrite(MOTORDIRECCION, HIGH);//establecemos rede de bajada
      digitalWrite(MOTORCONTROL, HIGH);//establecemos rele de alimentacion gral
    }
    moviendoPersiana=true;//se marca que la persiana se encuentra en movimiento.
    mon.print(F("MOVIENDO PERSIANA"));//Se muestra el estado en el monitor
    return 1;
  }else if(*comando==F("AC on-off")){
    //comando para encender y apagar el aire acondicionado
    acEncendido=!acEncendido;//cambiamos estado de variable
    mon.print(F("Aire on-off"));
    return 3;
  }else if(comando->substring(0,7)==F("ACTemp:")){
    //comando para modificar la temperatura de aire acondicionado  ejm:"ACTem:23"
    acTemperatura=comando->substring(7).toInt();//obtenemos temperatura solicitada
    mon.print(F("Aire temp"));
    return 4;
  }
  return -1;
}
```



Imagen 3.9.2-1 Fragmento de código función ejecutarComandos()

### 3.9.3 Configuración de router

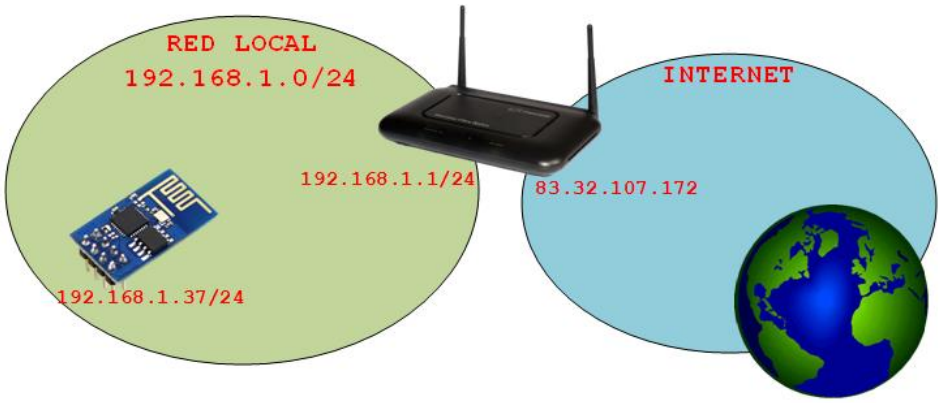



Imagen 3.9.3-1 Escenario de red

Llegados a este punto, surge un pequeño inconveniente. Si se observa la imagen 3.9.3-1, debido al NAT que realiza el router, todos los dispositivos de la red local tiene la misma dirección ip a nivel público. Por ello, si un cliente intenta realizar una nueva conexión desde la red externa al módulo ESP, el router no sabrá como encaminarlo, pues no es capaz de saber a qué dispositivo de la red interna va dirigido.

Para solucionar este problema, se ha establecido una nueva regla en la tabla del NAT del propio router. En ella se fija un puerto en el router (por ejemplo el:61111) y se le asigna como destino la dirección ip privada que le haya sido asignada al módulo ESP (192.168.1.37 en el ejemplo), en su puerto 80.(Como ha sido comentado en el apartado 3.9, se ha establecido el servidor TCP del módulo en el puerto 80).

| Server Name | External Port Start | External Port End | Protocol | Internal Port Start | Internal Port End | Server IP Address | RemoteHost IP Address | WAN Interface |
|---|---|---|---|---|---|---|---|---|
| | 61111 | 61111 | TCP | 80 | 80 | 192.168.1.40 | | ppp0.1 |

Imagen 3.9.3-2 Nueva regla NAT

De esta forma, el cliente externo solo tendrá que dirigirse al puerto 61111 del router para establecer la conexión con el servidor TCP del módulo ESP.

## *3.10 Control de Persiana*

Se ha desarrollado un control total de la persiana, tanto de forma remota con los comandos que recibe el servidor[17], como de forma física por parte del usuario pulsando los botones de subida y bajada.
Además el sistema permite al usuario subir o bajar parcial o completamente la persiana sin necesidad de mantener pulsado el botón de forma continua. Este sistema es llamado modo automático.

[17] Ver apartado FUNICION EJECUTARCOMANDOS().

Toda esta funcionalidad es llevada a cabo por la función comprobarPersiana() la cual es llamada dentro del bucle principal del sistema[18], y cuyo código es mostrado en la imagen 3.10-1.

De forma resumida, comprobarPersiana() comprueba el estado de los botones pasando a modo automático cuando el usuario ha pulsado el botón durante un tiempo superior a la constante *tiempoEntrarAuto* (que está fijada en 1,5segundos). En caso negativo, detiene el motor y envía la actualización del estado de la persiana al servidor.

Por otro lado, cuando entra en modoAutomatico, se calcula el tiempo necesario para realizar el movimiento automatizado, mediante la creación del timestamp *tautomatico*.
Cuando dicho tiempo es superado, la función detiene el motor y envía la actualización al servidor.
Como se ha visto anteriormente, este modo automático es activado, y tiempo de movimiento calculado cuando se recibe una petición de forma remota, a través de la función ejecutacomando()[19]. Posteriormente la función comprobarPersiana() será la encargada de detener el movimiento de esta, cuando el tiemplo esté cumplido.

Para observar más detalles del funcionamiento de esta función, ver el código que se muestra a continuación

```
//Esta es la funcion que gestiona el estado de la persiana, controlando el estado de los botones
//y el funcionamiento correcto del modo auto
void comprobarPersiana(){
  if(digitalRead(BOTONBAJAR)==HIGH){//comprobamos si boton bajar esta siendo pulsado
    if(modoAuto){//si viene en modo auto
      if(moviendoPersiana){
      //si se esta moviendo, quiere decir que el usuario quiere parar la persiana
        digitalWrite(MOTORCONTROL, LOW);//paramos control
        digitalWrite(MOTORDIRECCION, LOW);//paramos direccion
        actualizaPosicionPersiana();//actualizamos la posicion de la persiana
        moviendoPersiana=false;//dejamos de mover la persiana
      }
    }else{//si no viene en modo auto
      if(!moviendoPersiana){//y no se esta moviendo, el usuario lo ha pulsado para que baje
        moviendoPersiana=true;//indicamos el movimiento de la  persiana
        tpresion=millis();//creamos timestamp para el tiempo de presion del boton
        tmovimiento=millis();
        //creamos timestamp para el posterior calculo de la posicion de la persiana
        bajada=true;//indicamos que se encuentra bajando
      }
      digitalWrite(MOTORDIRECCION, HIGH);//rele de direccion en modo bajar
      digitalWrite(MOTORCONTROL, HIGH);//se activa el rele de control
    }

  }else if(digitalRead(BOTONSUBIR)==HIGH){//si se esta pulsando el boton subir
```

[18] Ver apartado FUNCIONAMIENTO BÁSICO.
[19] Ver apartado FUNCION EJETARCOMANDOS().

```
}else if(digitalRead(BOTONSUBIR)==HIGH){//si se esta pulsando el boton subir
  if(modoAuto){//si viene en modo auto
    if(moviendoPersiana){
    //y la ventana esta en movimiento, quiere decir que el usuario quiera pararla
      digitalWrite(MOTORCONTROL, LOW);//se desactiva el rele de control
      digitalWrite(MOTORDIRECCION, LOW);//se desactiva el rele de direccion
      actualizaPosicionPersiana();//actualizamos la posicion de la persiana
      moviendoPersiana=false;//indicamos el estado parado de la persiana
    }
  }else{//si no viene en modo auto
    if(!moviendoPersiana){
    //no se esta moviendo, quiere decir que el usuario quiere que la persiana suba
      moviendoPersiana=true;//indicamos moviento de persiana
      tpresion=millis();//creamos timestamp para el tiempo de presion del boton
      tmovimiento=millis();
      //creamos timestamp para el posterior calculo de la posicion de la persiana
      bajada=false;//indicamos que se encuentra subiendo
    }
    digitalWrite(MOTORDIRECCION, LOW);//indicamos direccion de subida con el rele
    digitalWrite(MOTORCONTROL, HIGH);//se activa el rele de control para alimentar
  }
}else{//si no se esta pulsando ningun boton
  if(moviendoPersiana){//y se esta moviendo la persiana
    if(modoAuto){//si esta en modoAuto
      if(millis()>tautomatico){//se comprueba si ha pasado del tiempo automatico
        modoAuto=false;//quitamos modo auto
        moviendoPersiana=false;//paramos movimiento
        digitalWrite(MOTORCONTROL, LOW);//se quita la alimentacion
        digitalWrite(MOTORDIRECCION, LOW);//vuelve el rele de direccion a su estado normal
        actualizaPosicionPersiana();//actualizamos la posicion de la persiana
      }
    }else{//si no viene en modo automatico, pero la parsiana se esta moviendo,
          //significa que el usuario ha dejado de pulsar uno de los botones
      if(millis()-tpresion>tiempoEntrarAuto){
      //se comprueba si el tiempo de presion en suficiente para entrar en en modo auto
        if(bajada){//si la persiana esta bajando
          tautomatico=millis()+tpersiana;//calculamos cuando la persiana estará bajada
        }else{//si esta subiendo
          tautomatico=millis()+(tiempoTotalPersianaSubida-tpersiana);
          //calculamos cuando la persiana ha subido completamente
        }
        modoAuto=true;//establecemos el modo auto como activo
      }else{//si el tiempo de pulsacion no ha sido suficiente para entrar en modo auto
        digitalWrite(MOTORCONTROL, LOW);//detenemos persiana
        digitalWrite(MOTORDIRECCION, LOW);//vuelve rele de direccion a su estado normal
        moviendoPersiana=false;//indicamos el estado parado de la persiana
        actualizaPosicionPersiana();//calculamos la nueva posicion de la persiana
      }
    }
  }else{//si la persiana no se encuentra en movimiento
    modoAuto=false;//aseguramos que el modo auto se encuentra desactivado
  }
 }
}
```

**Imagen 3.10-1 Fragmento de código, función comprobarPersiana()**

### 3.10.1 Cálculo de tiempos y posición

Para poder ubicar la posición de la persiana en todo momento, se cuenta con la variable tpersiana, que será incrementada o decrementada con los tiempos de cada desplazamiento. Puesto que sabemos cuál es el tiempo en el que la ventana está completamente abierta (25 segundos) y el tiempo en el que está completamente cerrada (0 segundos), nos es sencillo calcular el porcentaje de apertura.
Por otro lado, se le ha incorporado un pequeño offset, pues se ha observado que la existencia de un tiempo desde que la persiana está completamente bajada, hasta que esta comienza a mostrar la ventana.

El siguiente esquema el progreso de la variable tpersiana según va cambiando el porcentaje de apertura de la ventana.

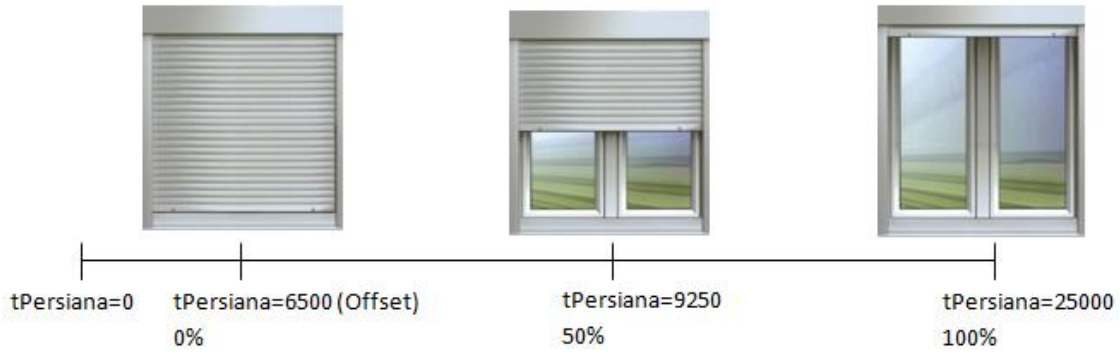



Imagen 3.10.1-1 Esquema Evolución parámetro tPersiana

## *3.11 Control Aire Acondicionado*

El sistema desarrollado es capaz de controlar el aire acondicionado/bomba de calor a través de un led infrarrojo, del mismo modo que lo haría su propio mando a distancia. Para poder generar las señales IR correspondientes, se ha empleado la librería IRremote.h[18] de Arduino.

La librería ofrece varias funciones de interfaz, para el uso de diferentes modelos de aire acondicionado por ejemplo: sendHvacPanasonic(), sendHvacMitsubishi() o sendSamsung(). En nuestro caso, contamos con un dispositivo de aire acondicionado Mitsubishi, luego emplearemos sendHvacMitsubishi().

A continuación se muestra la declaración de esta función:

```
void sendHvacMitsubishi(
    HvacMode        HVAC_Mode,       // Example HVAC_HOT
    int             HVAC_Temp,       // Example 21  (°c)
    HvacFanMode     HVAC_FanMode,    // Example FAN_SPEED_AUTO
    HvacVanneMode   HVAC_VanneMode,  // Example VANNE_AUTO_MOVE
    int             OFF              // Example false
  );
```

Como vemos, la función cuenta con los parámetros HVAC_Mode, HVAC_FanMode y HVAC_VanneMode, los cuales son enumeraciones definidas en la librería, que ofrecen un número limitado de opciones. En la imagen 3.11-1, se muestran sus definiciones:

```
typedef enum HvacMode {
  HVAC_HOT,
  HVAC_COLD,
  HVAC_DRY,
  HVAC_FAN, // used for Panasonic only
  HVAC_AUTO
}; // HVAC  MODE

typedef enum HvacFanMode {
  FAN_SPEED_1,
  FAN_SPEED_2,
  FAN_SPEED_3,
  FAN_SPEED_4,
  FAN_SPEED_5,
  FAN_SPEED_AUTO,
  FAN_SPEED_SILENT
};  // HVAC  FAN MODE

typedef enum HvacVanneMode {
  VANNE_AUTO,
  VANNE_H1,
  VANNE_H2,
  VANNE_H3,
  VANNE_H4,
  VANNE_H5,
  VANNE_AUTO_MOVE
};  // HVAC  VANNE MODE
```

Imagen 3.11-1 Enumeraciones librería IRremote.h

Una vez entendido el funcionamiento de la función sendHvacMitsubishi(), se procede a analizar la función enviarIR() del código del sistema, esta función realiza el control completo del sistema de aire acondicionado, y cuyo código se muestra en la imagen 3.11-2.

```
//Esta funcion se encarga de enviar la señal IR al aire acondicionado
void enviarIR(){
  if(acEncendido){//si el dispositivo esta encendido
    float t = dht.readTemperature();//tomo temperatura de la sala
    if(t<acTemperatura){//si la temperatura de la sala es menor a la objetivo
                        //enviamos señal con modo bomba de calor
      irsend.sendHvacMitsubishi(HVAC_HOT, acTemperatura, FAN_SPEED_AUTO
                                , VANNE_AUTO, false);
    }else{//si la temperatura de la sala es mayor a la de objetivo
          //enviamos señal con modo aire acondicionado
      irsend.sendHvacMitsubishi(HVAC_COLD, acTemperatura, FAN_SPEED_AUTO
                                , VANNE_AUTO, false);
    }
  }else{//si la variable encendido se encuentra desactivada se desea apagar el aire
        //enviamos señal de apagado
    irsend.sendHvacMitsubishi(HVAC_HOT, acTemperatura, FAN_SPEED_AUTO
                              , VANNE_AUTO, true);
  }
}
```

Imagen 3.11-2 Fragmento de código función enviar IR

En primer lugar se comprueba el estado del parámetro acEncendio. Si este se encuentra apagado, la función sendHvacMitsubishi() es llamada con la variable off con valor true. En caso contrario, se procede a tomar la temperatura de la sala y compararla con la temperatura objetivo del aire acondicionado (acTemperatura).
Si la temperatura de la sala es inferior a la temperatura objetivo, el dispositivo funcionará en modo bomba de calor. En caso contrario, en modo aire acondicionado.
Una vez realizada esta comprobación se procede a enviar las señales correspondientes.

De esta sencilla forma, es posible controlar el dispositivo de aire acondicionado/bomba de calor.

Por otro lado, se ha observado que la librería IRremote.h es muy extensa, ocupando una gran parte de la memoria de nuestro limitado micro Arduino Nano, que como se ha apuntado anteriormente, es un recurso limitado[20], provocando un comportamiento inestable del sistema. Por ello, se ha optado por la modificación de esta librería, eliminando todas las funciones y variables no empleadas, dejando solo las estrictamente necesarias para este proyecto. De esta forma, se ha conseguido reducir ostensiblemente el tamaño del programa, sin afectar a su comportamiento

[20] Ver apartado NÚCLEO DEL SISTEMA.

# CAPÍTULO 4:

# DOMOTE, DESCRIPCIÓN GENERAL

# CAPÍTULO 4: DOMOTE, DESCRIPCIÓN GENERAL

## 4.1 Introducción

Una vez desplegado el escenario, con el sistema domótico, se puede comenzar con el desarrollo de la aplicación en Android para su control y gestión. Esta aplicación será llamada **Domote**.

Tal y como se apuntó al comienzo del proyecto[21], esta aplicación se marca los siguientes objetivos a cumplir:

- ✓ Debe ser flexible para el uso de todos los usuarios de la plataforma ThingSpeak
- ✓ Permitir una fácil visualización de los parámetros
- ✓ Posibilidad de crear botones que modifiquen el valor de los parámetros
- ✓ Permitir al usuario establecer alarmas con gran flexibilidad en sus condiciones y eventos.
- ✓ Poder programar eventos, que modifiquen el valor de los parámetros.

En este capítulo estableceremos los datos y conceptos generales básicos de la aplicación. En posteriores capítulos, analizaremos la aplicación, tanto a nivel de usuario como interno:

- **Capítulo 5: Visión de Usuario.**
- **Capítulo 6: Estructura interna.**

## 4.2 Entorno de desarrollo y versiones Android

Domote es una aplicación nativa en el sistema operativo Android. Para su desarrollo se ha empleado el entorno de desarrollo Android Studio 1.2.2. Además, la aplicación ha sido compilada con la versión 5.1 de Android (Lollipop).

Por otro lado, otro de los aspectos importantes en la creación de la aplicación, es la versión mínima de Android necesaria para la ejecución de esta, pues se busca llegar al mayor número de usuarios posible, tener la mayor compatibilidad con todos ellos. De forma que todos los usuarios puedan ejecutar Domote sin tener como limitación la versión de Android que estos tengan en sus terminales. Es por ello, que se ha procurado el uso de librerías compatibles con las versiones más antiguas. Pudiendo así establecer versión mínima, en Android 2.2 (Froyo).

[21] Ver apartado REQUISISTOS DEL SISTEMA.

Si se observa la imagen 4.2-1 donde se muestra la dispersión de usuarios entorno a las diferentes versiones de Android[22] (A fecha de 2 de Mayo de 2016), se puede comprobar cómo se ha cumplido el objetivo de compatibilidad de Domote, pues la compatibilidad de este abarca prácticamente el 100% de los usuarios de Android.

| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.2 | Froyo | 8 | 0.1% |
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 2.2% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 2.0% |
| 4.1.x | Jelly Bean | 16 | 7.2% |
| 4.2.x | | 17 | 10.0% |
| 4.3 | | 18 | 2.9% |
| 4.4 | KitKat | 19 | 32.5% |
| 5.0 | Lollipop | 21 | 16.2% |
| 5.1 | | 22 | 19.4% |
| 6.0 | Marshmallow | 23 | 7.5% |

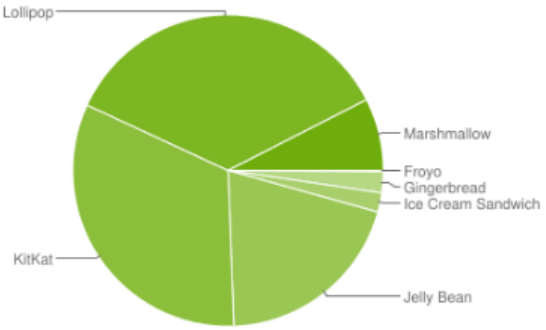



**Imagen 4.2-1 Dispersión de usuarios Android (a fecha 2 de Mayo del 2016)**

## 4.3 Introducción a la programación en Android

Este apartado es puramente descriptivo, y necesario para una mejor comprensión del funcionamiento interno y externo de la aplicación.
Para comenzar, se comenzará definiendo el elemento fundamental de las aplicaciones en Android, este son los Activities o Actividades. Un activity es cada una de las pantallas de las que está compuesta una aplicación. Estas clases Activity son implementadas como clase hija de "android.app.Activity", y se definen al menos sobrescribiendo la función onCreate(), que será llamada cuando la pantalla que se está definiendo sea mostrada en el dispositivo (Ver imagen 4.3-1).

[22] Fuente de los datos Google [21]

```
public class EjemploActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
    }
}
```

Imagen 4.3-1 Estructura básica de Activity.

Pero estas clases activities, no son el elemento gráfico que se muestra por pantalla como interfaz de usuario, sino que es la entidad que se encarga de mostrarlo, así como de controlar los eventos y acciones que ocurren sobre esta.

Los elementos que contienen la interfaz gráfica como tal son los llamados elementos Views o Vistas, los cuales pueden contener elementos contenedores como layouts, o elementos sobre los simples como por ejemplo texto, campos de texto o botones.
Como decimos, todos ellos, gestionados sus eventos por la clase activity.

Dado a esta clara separación entre interfaces de usuario, en esta memoria se presentarán de forma separada, por un lado las interfaces de usuario de cada una de las pantallas de las que se compone Domote (Capitulo 5), y por otro lado su funcionamiento y gestión interna (Capitulo 6).

## 4.4 Concepto de sala

Como se ha comentado, Domote tiene como objetivo el ofrecer una interfaz amigable e intuitiva al usuario. Para ello, se ha conseguido un punto de abstracción superior al existente en Thinkspeak, que estructura los campos en canales, y que como se ha mostrado anteriormente, este modelo es poco flexible y complicado[23].
Para solucionar este problema, Domote introduce el concepto de sala, más cercano al usuario, que al fin y al cabo quiere controlar un conjunto de parámetros en una o varias salas, abstrayéndole así de pensar en que canal se encuentran estos campos. Este solo tendrá que pulsar en la sala que quiere observar para comprobar el estado de todos sus campos, independientemente del canal al que pertenezcan estos campos.
Con esta abstracción, además se puede salvar la restricción que suponen los 8 campos por canal.

[23] Ver apartado Definición de canales y campos en Thinkspeak

# CAPÍTULO 5:
# VISIÓN DE USUARIO

# CAPÍTULO 5: VISIÓN DE USUARIO

## 5.1 Introducción

En este capítulo analizaremos en profundidad el interfaz de usuario de Domote.
La estructura de este interfaz está claramente diferenciada en dos fases:

- Fase de Configuración
- Fase de Visión general

A continuación se presentará el diagrama de flujo de la interfaz, para posteriormente, ir analizando pormenorizadamente las fases. Se irán recorriendo todas las pantallas de la aplicación, desde que el usuario inicia por primera vez la aplicación y la configura (Fase de configuración), hasta el uso más habitual de esta aplicación (Fase de Visión general).

## 5.2 Diagrama de flujo

En la imagen 5.2-1 se muestra el diagrama de flujo de la aplicación. En ella, se puede observar las dos fases en las que se divide, así como las diferentes pantallas (llamadas Acitivies en el sistema operativo Android) que las componen.

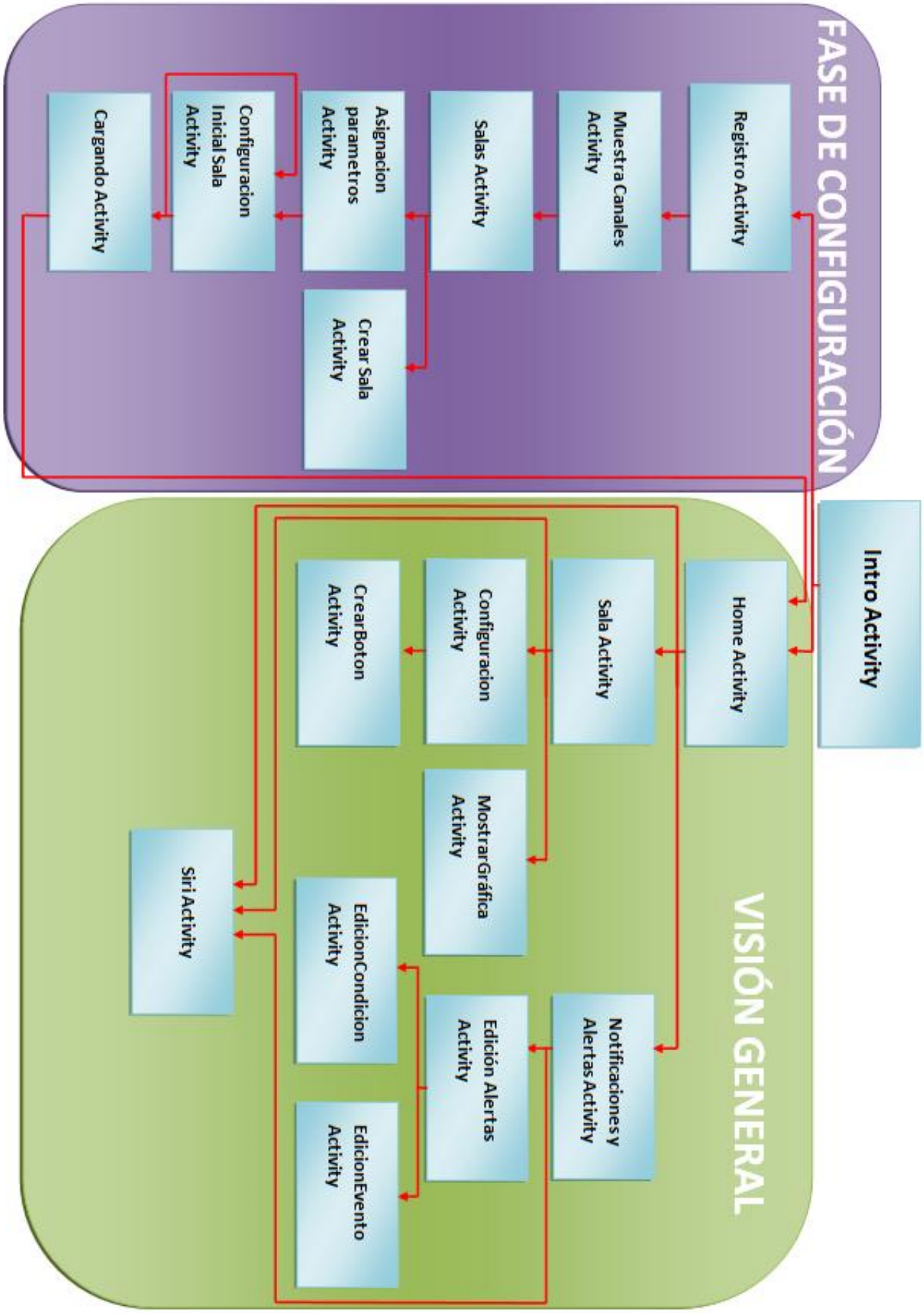



Imagen 5.2-1 Diagrama de flujo Domote

## 5.3 Fase de Configuración

Se trata de la primera fase de la aplicación, en la cual, el objetivo es tomar todos los datos necesarios para poder adaptar la aplicación a las necesidades del usuario.
Esta fase debe ser lo más simple e intuitiva posible.

Por ello, para la configuración inicial de la aplicación, el usuario debe de realizar los siguientes 5 sencillos pasos:

### 1. Registro (RegistroActivity):

Esta es la pantalla principal si el usuario no se ha registrado aun en la aplicación. En ella, se muestra dos cuadros de texto, en los cuales el usuario debe introducir su nombre de usuario en Thingspeak y su apikey de lectura general.
Una vez el usuario pulsa el botón "iniciar", la aplicación consulta los datos del usuario introducido en Thingspeak y toma los canales que este tiene creados, así como los campos de los canales que hayan sido declarados como públicos.
En caso de que se haya producido algún error, se mostrará al usuario un mensaje con el error producido.

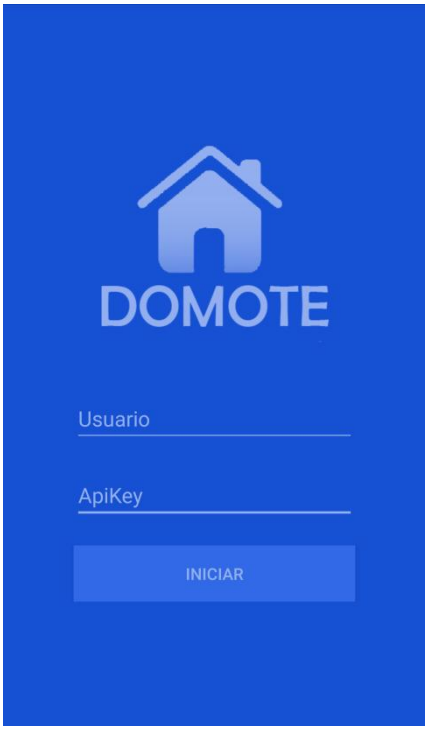



Imagen 5.3-1 Pantalla de registro

## 2. Selección de canales y campos (MuestraCanalesActivity):

En esta pantalla se muestran los canales y campos que el usuario ha definido en Thingspeak. En caso de que se trate de un canal privado, se muestra un campo de texto para que el usuario introduzca el apikey de lectura de dicho canal privado.
El usuario debe de seleccionar los campos que desea emplear en la aplicación. Una vez seleccionados, se puede acceder a la siguiente pantalla pulsando en la flecha derecha de la barra de navegación.

## 3. Salas (SalasActivity):

En esta pantalla se muestran las salas creadas mediante una imagen de los mismos, su nombre y dos botones para su edición o eliminación.
En la parte superior se muestra el botón de "+ Añadir nueva sala" para que el usuario cree sus salas.
Una vez creadas, se puede acceder a la siguiente pantalla pulsando en la flecha derecha de la barra de navegación.

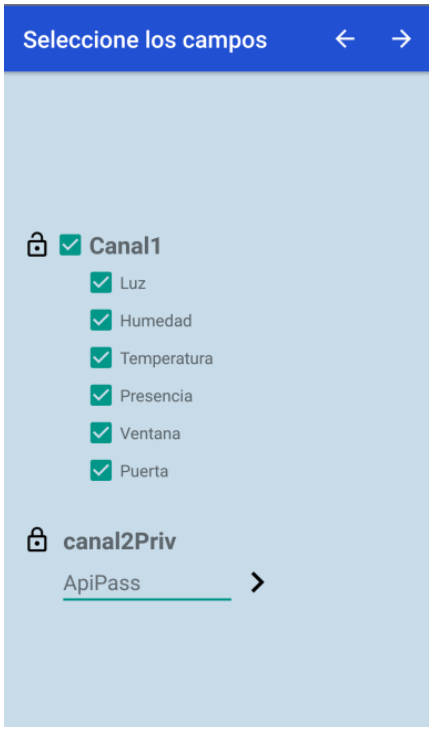



Imagen 5.3-2 Selección de campos

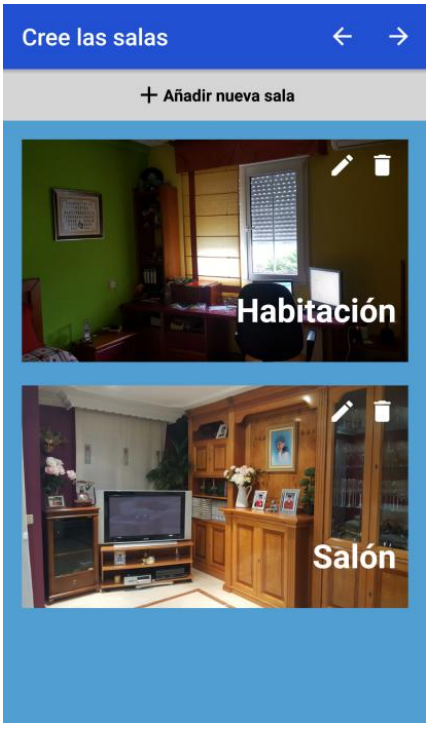



Imagen 5.3-3 Salas

### 3-1 Edición de Sala (CrearSalaActivity):

En esta pantalla el usuario puede configurar o editar una sala. Para ello se muestran los campos de texto Nombre y Descripción.
Además, para poder identificar y personalizar la nueva sala, se ofrece al usuario asignarle un color o una imagen de su biblioteca.
Por último, en la parte inferior de la pantalla se muestran dos botones para aceptar o cancelar la edición de dicha sala.
La sala solo será creada si el usuario introduce los datos correctamente, y no existe ninguna sala con el mismo nombre. En caso contrario, se muestra un mensaje al usuario con el error producido.

## 4. Asignación de parámetros (AsignacionParametrosActivity):

Una vez creadas las salas, se muestra en la parte superior de la pantalla, un listado con todos los campos que han sido seleccionados en el paso 2, y que todavía no tienen ninguna sala asignada.

Por otro lado, en la parte inferior se muestran otros dos listados. Y en la parte izquierda un listado con los nombres de las salas creadas, en la parte derecha un segundo listado en el cual se muestran los campos que han sido asignados a la sala que ha sido seleccionada en el listado derecho.
Entre ambos listados, se muestran dos botones para añadir o quitar los campos a la sala seleccionada.
Una vez asignados los campos a las salas, se puede acceder a la siguiente pantalla pulsando en la flecha derecha de la barra de navegación.

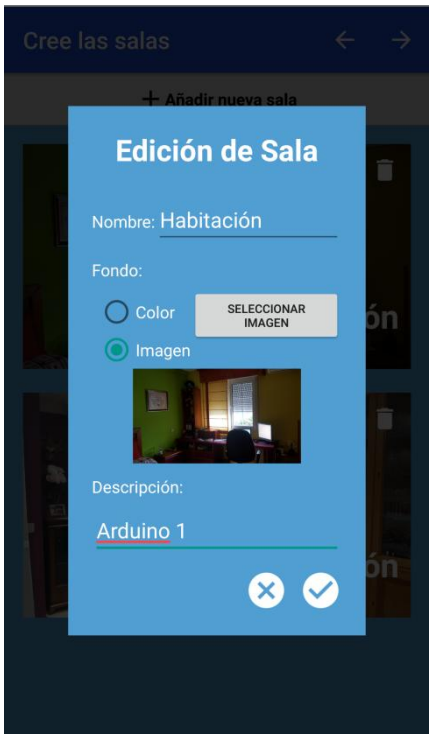



Imagen 5.3-4 Edición de sala

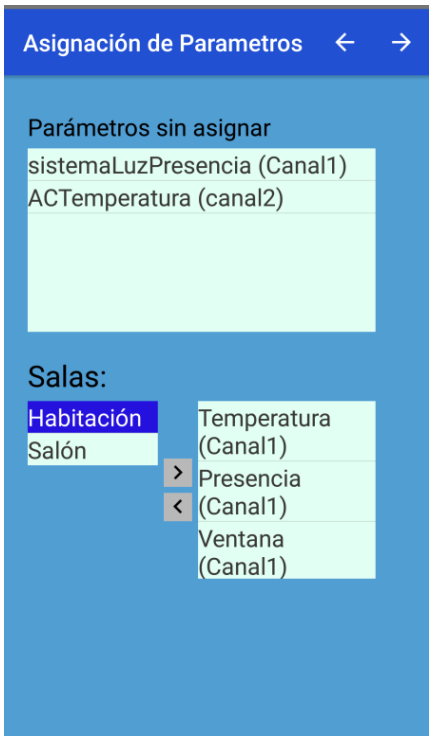



Imagen 5.3-5 Asignación de parámetros

## 5. Configuración inicial de salas (ConfiguraciónInicialSalaActivity):

Tras la asignación de campos a las salas. Se mostrará una serie de pantallas (tantas como salas) en las cuales se configura la sala y los campos que la integran para que la visión de estos se adapte a las necesidades del usuario.

En la parte superior de la pantalla se muestra un campo de texto con el tiempo de actualización de los datos cuyo valor mínimo puede ser 30 segundos.
En la parte inferior de la pantalla, se muestra una lista desplegable en la que se muestran los campos asignados de la sala a configurar.
Si el usuario pulsa sobre uno de los campos, se despliega una lista con sus parámetros de configuración, como se muestra en la imagen 5.3-6.

En esta lista, el usuario debe de seleccionar inicialmente el tipo de dato del que se trata: Digital (El campo puede tener solo dos valores: 0 y 1) o Analógico (El campo puede tener múltiples valores).
En caso de se trate de un campo digital, el usuario puede asignar un nombre a los valores 0 y 1(Por ejemplo: Encendido-Apagado). En caso de se trate de un campo analógico, el usuario puede asignar las unidades de dicho campo (Por ejemplo: ºC).

Tras seleccionar el tipo de dato, el usuario puede seleccionar que información desea visualizar:

- ✓ Hora última muestra
- ✓ Fecha última muestra
- ✓ ID última muestra
- ✓ Gráfica


Imagen 5.3-6 Configuración campos de sala

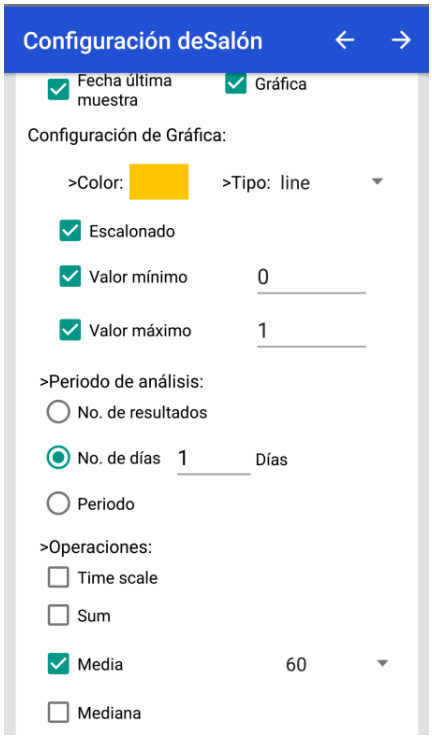



Imagen 5.3-7 Configuración de gráfica

Por último, en caso de que el usuario seleccione la opción mostrar la gráfica, se mostrarán las opciones de visualización de gráfica como se muestra en la imagen 5.3-7, con los siguientes parámetros:

- ✓ Color de grafica
- ✓ Tipo de gráfica
- ✓ Valor mínimo
- ✓ Valor máximo
- ✓ Periodo de análisis (Numero de resultados, número de días o periodo)
- ✓ Operaciones con conjuntos de muestras (TimeScale, suma, media, mediana)

Como será comentado más adelante[24], estas gráficas son generadas por Thingspeak, y las opciones que aquí se muestran son las opciones que su api nos ofrece[26].

Tras configurar todos los campos de la sala, se puede acceder a la configuración de la siguiente sala pulsando en la flecha de navegación derecha.

Tras configurar todas las salas se muestra una pequeña animación de carga de la configuración (Ver imagen 5.3-8) en la cual se crea una expectación en el usuario, además de establecer una clara separación entre la fase de registro, y la visión general.

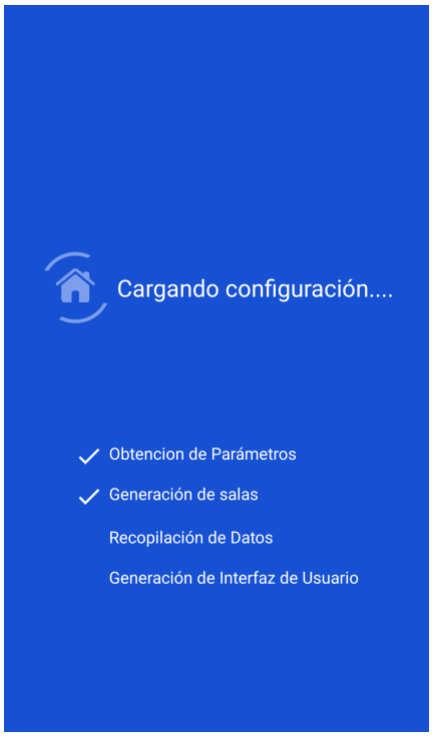



**Imagen 5.3-8 Animación de carga**

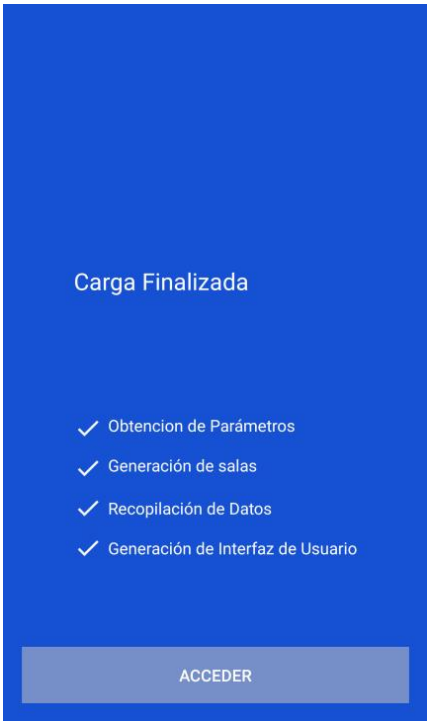



**Imagen 5.3-9 Fin animación de carga**

Finalmente, tras la animación, se muestra el botón acceder (Ver imagen 5.3-9), que el usuario deberá pulsar para acceder a la visión general.

[24] Ver apartado OBTENCIÓN DE GRÁFICAS

## *5.4 Visión general*

Esta es la sección principal de la aplicación. Una vez el usuario se ha registrado, se muestra la información ya adaptada a las necesidades del usuario. En ella se pueden distinguir cuatro pantallas principales, con sus diferentes submenús.

- Menú principal
- Visión de sala
- Notificaciones y alertas
- Asistente de voz

A continuación, se describen estas pantallas principales y sus diferentes submenús que las componen:

### 1. Pantalla principal (HomeActivity)

Esta pantalla se muestran las salas creadas mediante una imagen, su nombre y descripción, de esta forma el usuario puede identificar de forma intuitiva la sala a la que desea acceder.
En la parte superior, se muestra la barra de navegación con el nombre de usuario Thingspeak, un botón para acceder al asistente de voz, un botón para acceder a la sección de notificaciones y alertas, y un último botón de más opciones con las opciones configurar (volver a la fase de configuración) y salir (volver a la pantalla de registro).


**Imagen 5.4-1 Pantalla principal**

Si el usuario pulsa sobre la imagen de una de las salas, este accederá a la visión de la sala seleccionada.

## 2. Visión de sala (SalaActivity)

La visión de sala muestra el estado actual de los campos que la componen (Ver imagen 5.4-2).
En la parte superior de la pantalla se muestra la imagen de la sala y su nombre, y sobre esta los botones de volver (volver al menú principal), edición de sala (configuración de la visión de salía, visto en el paso 5 de la configuración), y asistente de voz.
En la parte inferior izquierda de la imagen se muestra una cuenta atrás, con el tiempo restante para la nueva actualización de los campos.
Bajo la imagen de la sala, se muestra una vista desplazable (a izquierda y derecha) que cuenta con dos apartados:

- **Estado de los campos (Apartado izquierda)**

Situado en la parte derecha de la pista, este apartado dispone de una vista expandible con el estado actual de los campos.
Como se puede ver en la imagen 5.4-2 se muestra el estado de los campos con las unidades y los valores que han sido introducidos en la fase de configuración. De esta manera, se permite una visión muy cómoda de los valores, con un formato más natural para el usuario
Si el usuario pulsa sobre uno de los parámetros, este se expandirá, mostrando los datos que este haya configurado, como se puede ver en la imagen 5.4-3.

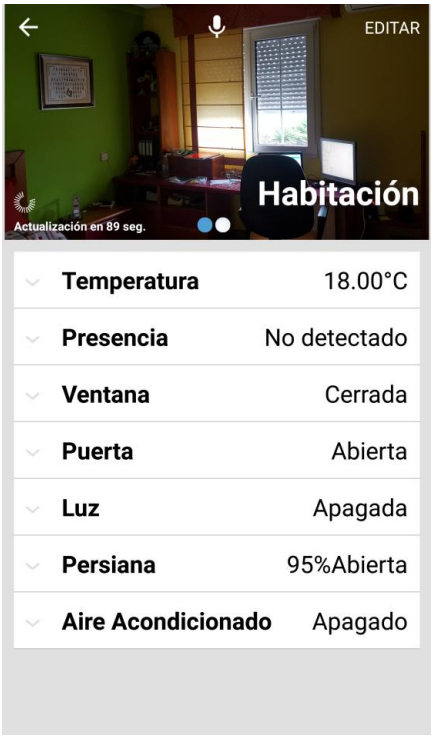



Imagen 5.4-2 Campos de la sala


Imagen 5.4-3 Campo expandido

Si se desea ampliar la gráfica para una vista más en detalle, el usuario solo tendrá que pulsar sobre ella. De esta forma, la gráfica se mostrará a pantalla completa.

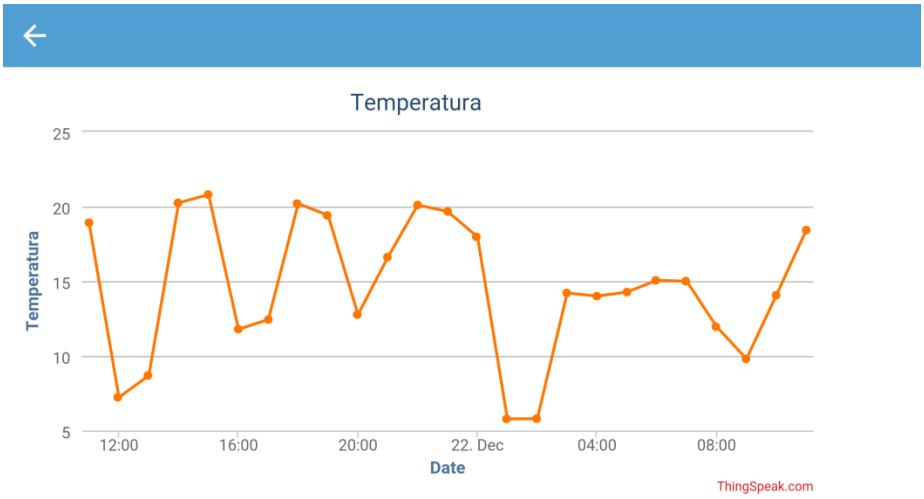



**Imagen 5.4-4 Gráfica de campo**

Para volver a la pantalla anterior, el usuario solo tiene que pulsar la flecha izquierda de la barra de navegación o el botón de atrás del dispositivo.

- **Botones de la sala (Apartado derecha)**

Si el usuario realiza un movimiento de desplazamiento hacia la derecha en la visión de sala, será mostrada la lista de botones definidos por el usuario, los cuales que han sido creados para modificar algunos de los campos de la sala (Ver imagen 5.4-5).
El usuario solo tiene que pulsar sobre estos, para proceder a la modificación de los campos.

Por otro lado, en la parte inferior de la lista se muestra el botón "+ Pulse para añadir", mediante el cual el usuario puede añadir nuevos botones a la lista de la sala. Si pulsa sobre este, será mostrada una nueva pantalla en la cual puede seleccionarse el tipo de botón a añadir, como se puede observar en la imagen 5.4-6.

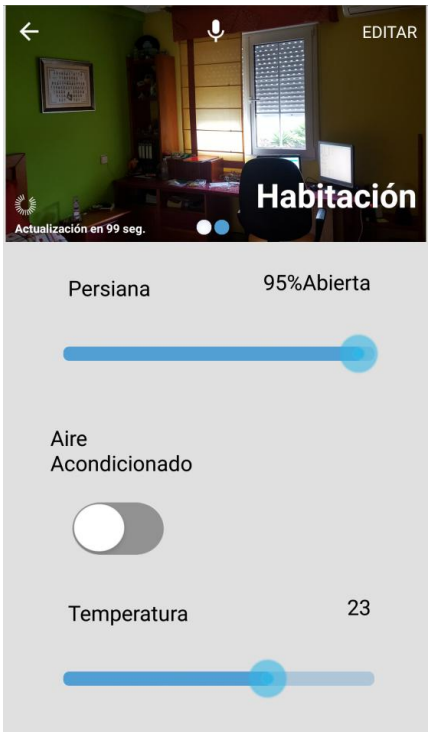



Imagen 5.4-5 Botones de sala


Imagen 5.4-6 Tipo de botón

Han sido definido tres tipos de botones:

- Botón simple
- Interruptor
- Barra de progreso

En función del tipo de botón que el usuario elija, será mostrada una pantalla de configuración diferente como puede ver en las imágenes 5.4-7, 5.4-8, 5.4-9.

Básicamente la estructura de la configuración es la misma en todos los botones. Se muestra en la parte superior los botones "aceptar" y "cancelar", e inmediatamente después, una imagen con una muestra del botón a añadir, y un campo de texto para introducir el nombre del botón.

Posteriormente la configuración varia ligeramente en función del botón elegido, pero la filosofía es la misma:
El usuario debe de introducir una URL, a la cual la aplicación realizará una petición HTTP GET cuando el botón sea pulsado. En el caso de que se trate de una barra de progreso, también se debe de introducir el nombre del parámetro que se enviará con el valor del progreso.

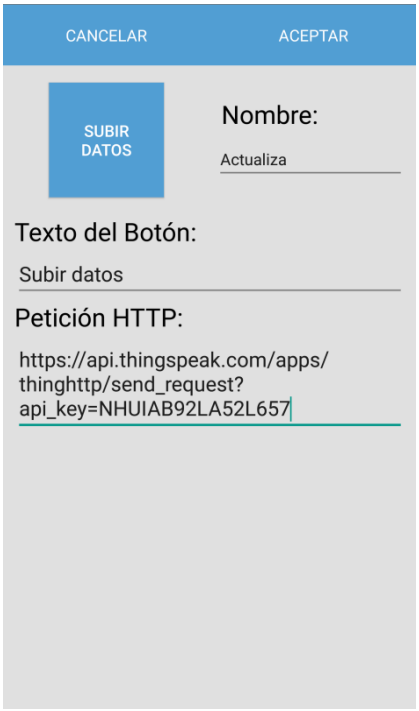



**Imagen 5.4-7 Configuración Botón simple**

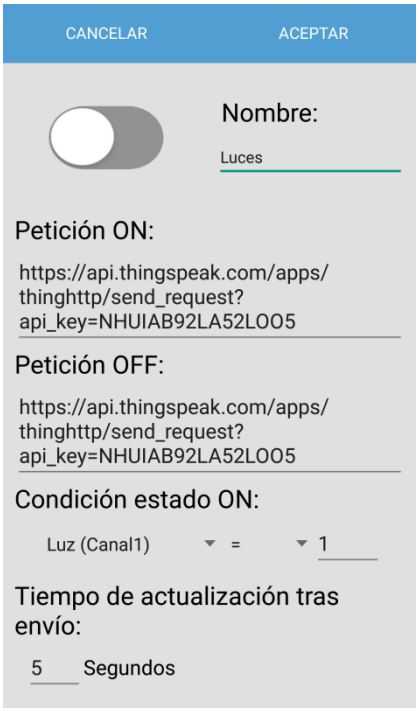



**Imagen 5.4-8 Configuración Interruptor**

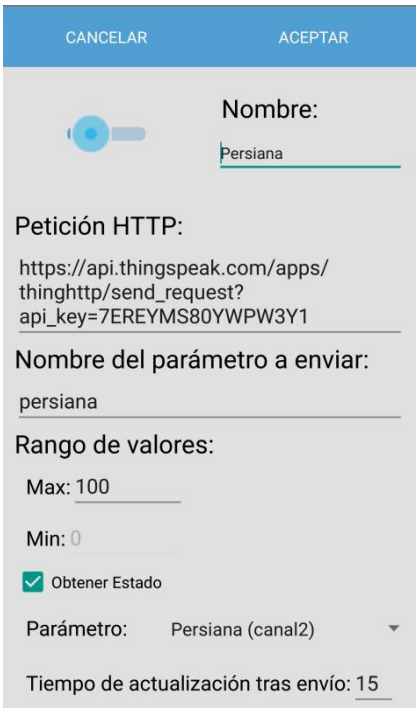



**Imagen 5.4-9 Configuración Barra de progreso**

Por otro lado, en el caso de los interruptores y barras de estado, que pueden tener varios estados, se debe de seleccionar un campo, mediante el cual la aplicación pueda obtener cual es el estado del botón a mostrar.

## 3. Notificaciones y Alertas (NotificacionesyAlertasActivity)

Esta pantalla muestra un listado con el nombre de todas las alertas creadas, así como los botones: Crear, eliminar, editar, activar y desactivar.
Si el usuario selecciona una de las alertas, puede observar todas sus propiedades mediante los campos de texto: Nombre, descripción, condición y evento, situados en la parte inferior de la pantalla, como se puede comprobar en la imagen 5.4-10.
Por último, en la parte superior de la pantalla, se sitúa la barra de navegación con la flecha para volver al menú principal y el botón del asistente de voz.

## 4. Creación de alertas (EdicionAlertasActivity)

Este activity es empleado tanto en la creación como la edición de las alertas.
Como se puede comprobar en la imagen 5.4-11, en esta pantalla se muestran los campos de texto nombre y descripción así como las secciones Condición y Evento.
La lista de condiciones, presenta todas las condiciones que deben cumplirse para que la alerta sea lanzada.
Por otro lado, la lista de eventos, presenta todos los eventos que deben ser lanzados cuando las condiciones de la alerta se cumplan.

El usuario puede añadir tantas condiciones y eventos como desee, pulsando al botón "+" de la lista correspondiente.

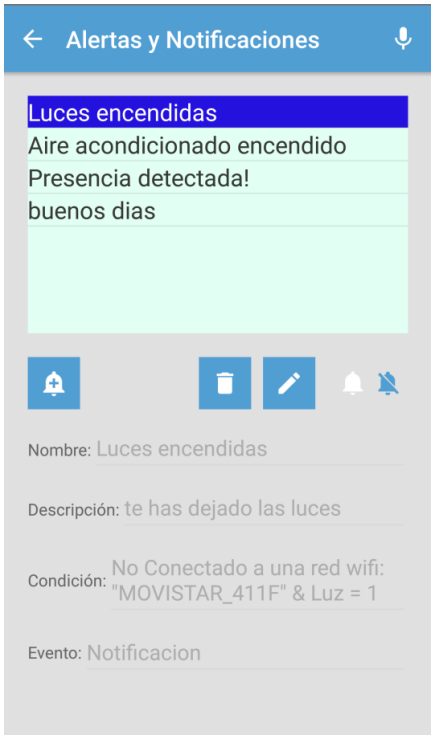



Imagen 5.4-10 Notificaciones y alertas

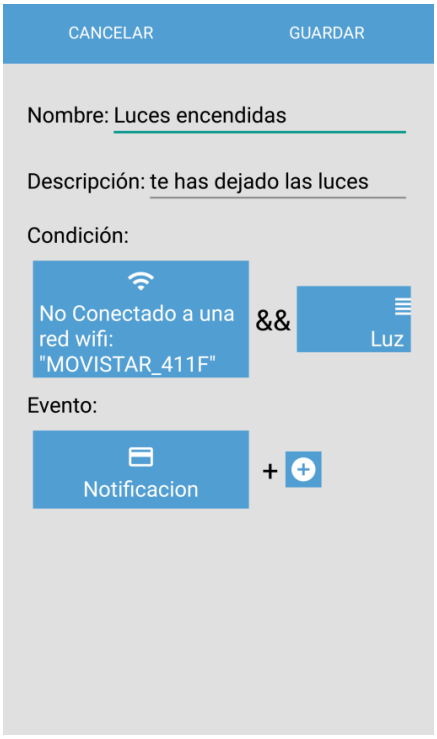



Imagen 5.4-11 Edición de Alertas

- **Creación de Condición (EdicionCondiciónActivity)**

Si el usuario pulsa sobre el botón "+" de la lista de condiciones, se muestran los diferentes tipos de condiciones que permite establecer la aplicación, estos se pueden separar en 3 grandes grupos (ver imagen 5.4-12):

- Parámetro: La condición se cumple en función del estado del campo que el usuario seleccione.
- Conectividad: Condiciones relativas a la conectividad del dispositivo.
- Temporal: Condiciones temporales, con posibilidad de repetición.

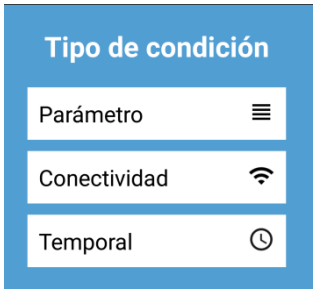



Imagen 5.4-12 Tipo de condición

Dentro de cada grupo, se han definido un conjunto de condiciones relacionadas con ese tipo, para que el usuario pueda configurar de acuerdo a sus necesidades.
En la imagen 5.4-13, se muestra de forma esquemática todas las posibles condiciones de cada grupo

| TIPO DE CONDICIÓN | CONDICIONES |
|---|---|
| Parámetro | = != < > |
| Conectividad | Conectado a wifi |
| | No conectado a wifi |
| | Conectado a una red wifi |
| | No conectado a una red wifi |
| | Veo una red wifi |
| | No veo una red wifi |
| Temporal | A una hora determinada |
| | Dentro de un periodo horario |
| | Fuera de un periodo horario |
| | Dentro de un periodo diario |
| | Fuera de un periodo diario |

**Imagen 5.4-13 Esquema Condiciones**

En función del tipo de condición escogida, se mostrará una ventana para configurar estas condiciones, como se puede ver en las imágenes 5.4-14, 5.4-15 y 5.4-16.

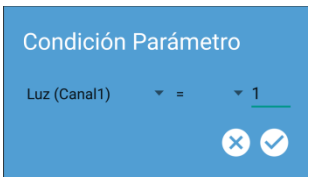



**Imagen 5.4-14** Condición Parámetro


**Imagen 5.4-15** Condición Conectividad

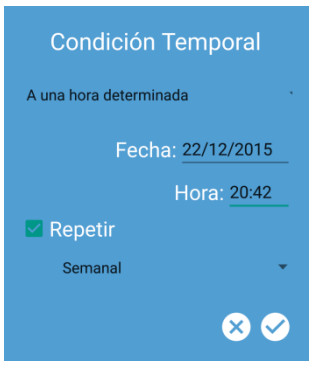



**Imagen 5.4-16** Condición Temporal

- **Creación de Evento (EdicionEventoActivity)**

La aplicación tiene definidos múltiples tipos de eventos a elegir por el usuario.
Si el usuario pulsa sobre el botón "+" de la lista de eventos, se muestran los diferentes tipos de eventos (Imagen 5.4-17):

- Notificación: Este evento, lanza una notificación en el dispositivo móvil del usuario.
- Pulsar Botón: Produce la pulsación automática de uno de los botones definidos por el usuario.
- Petición HTTP: Ejecuta una petición HTTP GET a la dirección URL especificada.


**Imagen 5.4-17 Tipo de evento**

En función de la opción escogida, se mostrará una ventana para configurar estos eventos:

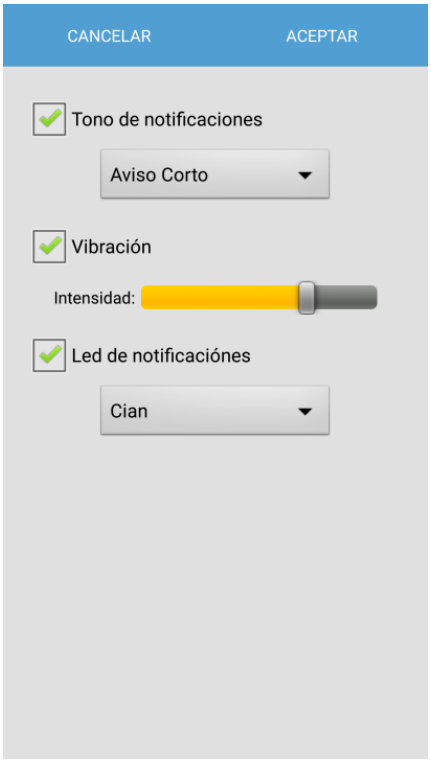



Imagen 5.2.3-18 Edición Notificación

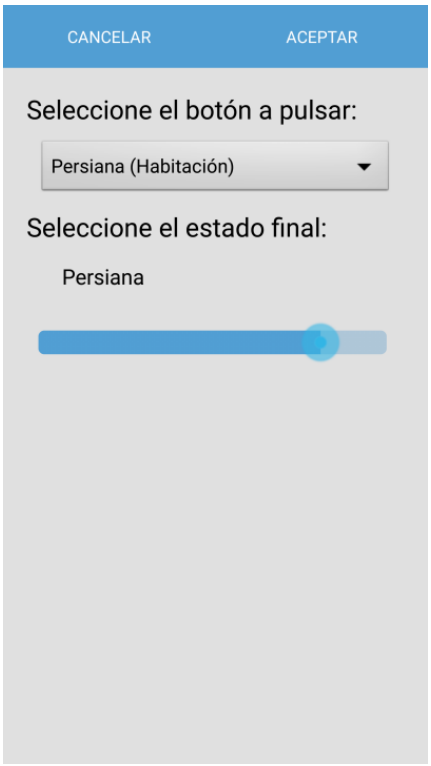



Imagen 5.2.3-19 Edición Pulsación Botón

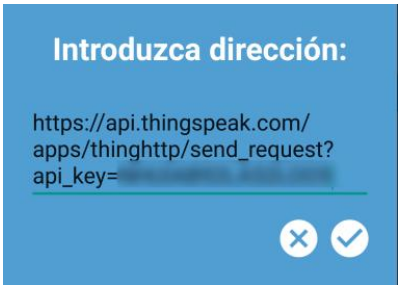



Imagen 5.2.3-20 Edición Petición HTTP

## 5. Asistente de Voz (SiriActivity)

El asistente de voz es accesible desde las principales pantallas de la aplicación (HomeActivity, SalaActivity y NotificacionesyAlertasActivity) a través del botón que se muestra en la imagen 5.4-21.

**Imagen 5.4-21** Botón Asistente de voz

El objetivo de este asistente, es que el usuario obtenga la información o realice las acciones que necesita con unas pocas palabras, de una forma más nativa y natural.
Además, de resultar un elemento diferencial de esta aplicación con respecto al resto de aplicaciones existentes, y que puede ser un importante motivo para que el usuario decida instalar Domote en su dispositivo.

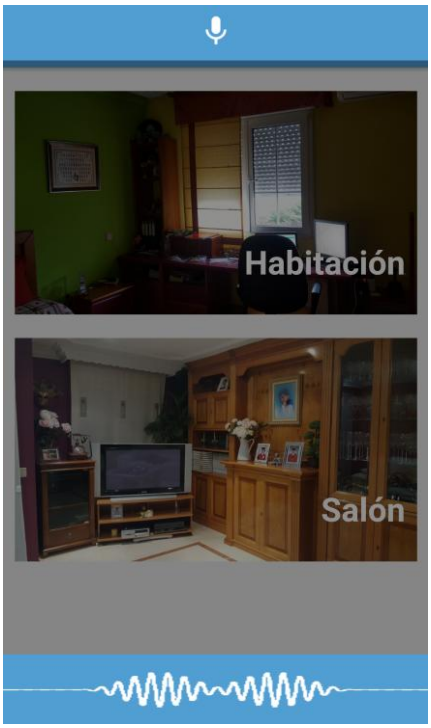



Imagen 5.4-22 Asistente de voz

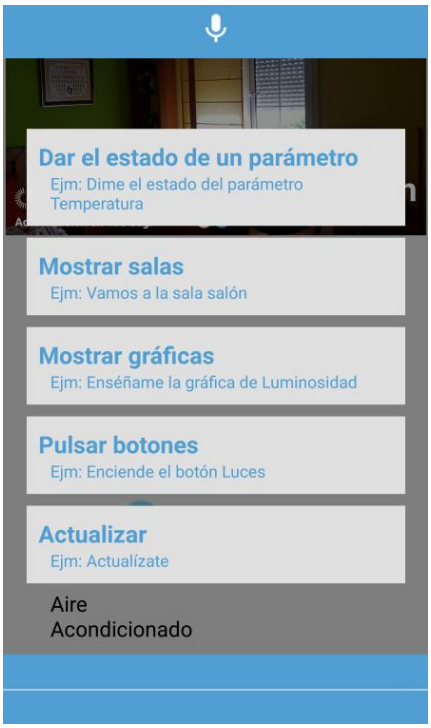



Imagen 5.4-23 Comando "¿Qué puedes hacer por mi?"

Simplemente con pulsar este botón, la aplicación comenzará a escuchar al usuario, oscureciendo ligeramente la pantalla, y mostrando en la parte inferior, una animación simulando la onda sonora recogida, como se muestra en la imagen 5.4-22.

El sistema es capaz de detectar el final de la alocución. Cuando este es detectado, se procede al procesado.
El procesado se encarga de comprobar si se trata de uno de los comandos registrados en el sistema, y si es así actuar en consecuencia. Más adelante, veremos más en profundidad su funcionamiento interno.

Mediante el comando de voz "¿Que puedes hacer por mi?", el usuario puede consultar al asistente la lista con todos los comandos procesables por el asistente, así como ejemplos de cómo ejecutarlos (Ver imagen 5.4-23).

El asistente de voz puede realizar las siguientes acciones:

- Mostrar Salas
- Mostrar Graficas
- Actualizar Campos
- Pulsar botones
- Dar estado de campos
- Dar estado de salas
- Contar Chistes

# CAPÍTULO 6:
# ESTRUCTURA INTERNA

# CAPÍTULO 6: ESTRUCTURA INTERNA

## 6.1 Introducción

Una vez ha sido expuesto el funcionamiento a nivel de usuario de la aplicación Domote, en este capítulo se procederá a explicar su funcionamiento a nivel interno.

El código de la aplicación tiene una longitud demasiado extensa como para ser expuesta punto por punto en esta memoria, por ello se hará una descripción general del funcionamiento interno, para posteriormente centrarnos en los puntos más importantes del código que serán comentados en profundidad.

## 6.2 Funcionamiento y sistema de clases

La aplicación está dividida en dos partes fundamentales, por un lado, la ejecución de forma secuencial de las diferentes **actividades** (Pantallas) que la componen, según el usuario se va desplazando entorno a ella. Por otro lado, se ejecuta un **servicio,** el cual es independiente del usuario, y se encontrará en todo momento en ejecución, independientemente de si la aplicación se encuentra abierta o no.

En el esquema 6.2-1, se muestra gráficamente el funcionamiento de nuestra aplicación.

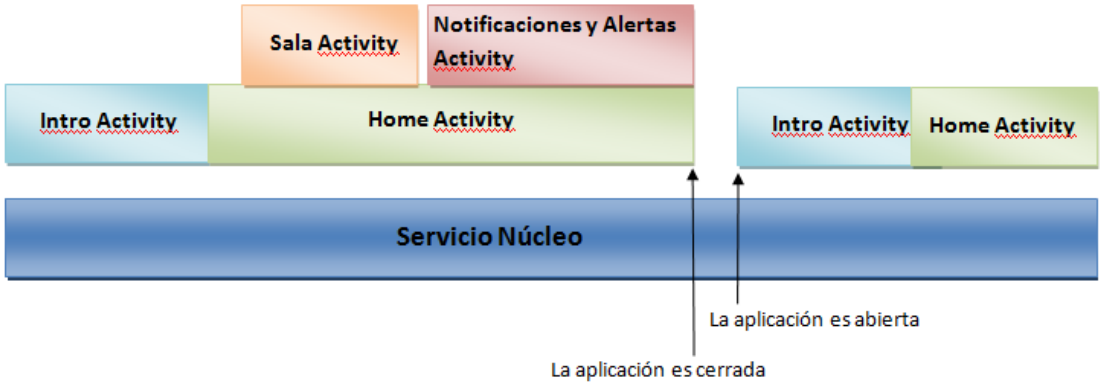



Imagen 6.2-1 Esquema Funcionamiento interno

Con este esquema de funcionamiento, podemos mantener una actualización periódica tanto de los campos y alertas, independientemente de si la aplicación se encuentra en ejecución o no. Esto, nos permitirá generar en el usuario una mayor sensación de fluidez, sin esperas de actualización cada vez que este abra la aplicación, además de un control constante de las alarmas, aun cuando el usuario haya cerrado la aplicación.

## 6.3 Servicio Núcleo

Este será el encargado de las siguientes funciones:

- **Almacenamiento y Gestión de los datos de la configuración del usuario**
- **Actualizaciones periódicas de los campos**
- **Comprobación de notificaciones y alarmas**

Como se puede observar, este servicio realiza una labor fundamental, es por ello que ha sido llamado "ServicioNucleo".
A continuación, se describirá de forma resumida como realiza estas funciones:

### 6.3.1 Almacenamiento y gestión de los datos

Para entender el almacenamiento de datos del sistema, debemos de explicar previamente el sistema de clases empleado.
Se ha definido una clase llamada Memoria, la cual es la encargada de aunar todos los datos que definen la configuración del usuario[25].

Como se puede observar en la imagen 6.3.1-1, esta clase contiene el nombre y el apiKey del usuario, así como una serie de objetos ArrayList que almacenarán los canales, salas y notificaciones definidos por el usuario.

| MEMORIA | |
|---|---|
| ArrayList <TSChannel> | canales |
| ArrayList <Sala> | salas |
| ArrayList <Notificaciones> | notificaciones |
| String | user |
| String | apiUser |
| int | updateTime |

Imagen 6.3.1-1 Esquema Clase memoria

El objeto TSChannel es el encargado de almacenar todos los datos relativos a un canal. Como se puede observar en la imagen 6.3.1-2, esta clase contiene una serie de variables, tales como su nombre, id o descripción, así como un ArrayList de objetos TSField que almacena los campos de este canal.

| TSChannel | |
|---|---|
| String | id |
| String | name |
| String | description |
| String | apikey |
| ArrayList<TSField> | campos |
| boolean | privado |
| boolean | desbloqueado |

Imagen 6.3.1-2 Esquema Clase TSChannel

[25] Ver apartado Fase de configuración.

Los objetos TSField contienen la información relativa a un campo, como puede ser nombre, canal al que corresponde, posición dentro de este canal, o información relativa a la última entrada del mismo.
Además contiene un objeto de la clase ConfiguacionParametros, esta clase ha sido creada para almacenar toda la configuración que el usuario ha definido sobre la visualización del campo[26] (tipo de dato, unidades, tipo de gráfica…).

| TSField | |
|---|---|
| String | name |
| String | channel |
| boolean | seleccionado |
| int | num |
| String | apiKey |
| boolean | ocualto |
| ConfiguracionParametros | parametrosVisualizacion |
| String | ultimaEntradaId |
| String | ultimaEntradaFecha |
| String | ultimaEntradaHora |
| String | ultimaEntradaValor |

**Imagen 6.3.1-3 Esquema Clase TSField**

Los objetos Sala contienen la información que define a cada una de las salas creadas por el usuario. Como vemos incluye un ArrayList "campos" que almacena los campos asignados a dicha sala. Y un segundo ArrayList "botones", que contiene una serie de objetos de tipo Botón que serán los encargados de definir la configuración de cada uno de los botones que contiene la sala.

| Sala | |
|---|---|
| String | nombre |
| int | color |
| String | descripcion |
| ArrayList<TSField> | campos |
| String | imagen |
| ArrayList<Boton> | botones |

**Imagen 6.3.1-4 Esquema Clase Sala**

Por último, los objetos Notificación definen cada una de las notificaciones creadas por el usuario. Como se puede observar contiene una serie de variables booleanas (atendida, lanzada y enabled) que serán empleadas como flags por el sistema.

| Notificacion | |
|---|---|
| String | nombre |
| String | descripcion |
| ArrayList<Condicion> | condiciones |
| ArrayList<Evento> | eventos |
| boolean | atendida |
| boolean | lanzada |
| boolean | enabled |

**Imagen 6.3.1-5 Esquema Clase Notificación**

Además objeto Notificación contiene dos ArrayList de objetos Condición y Evento, que definirá las condiciones creadas por el usuario, así como la información con los eventos deseados. El funcionamiento de estos objetos será explicado más adelante[27].

Todas estas son las principales clases creadas para definir el almacenamiento de los datos del sistema. Se han definido las más importantes e imprescindibles para entender su funcionamiento básico.

Una vez descrito el sistema de clases, ya se puede entender mejor la gestión de los datos. El Servicio Núcleo es el encargado del almacenamiento del objeto Memoria, que como hemos visto, es el objeto que contiene toda la información.

[26] Ver apartado FASE DE CONFIGURACIÓN.
[27] Ver apartado ALARMAS Y NOTIFICACIONES.

Los activities, conectarán con este servicio, y obtendrán los datos que necesiten para mostrar la interfaz de usuario. Estos datos pueden ser tanto la configuración de usuario ya almacenada, como el último valor de los campos, los cuales se encuentran actualizados gracias a la labor del núcleo[28].

Por último, se debe de apuntar, que aunque el servicio no se cierra cuando el usuario cierra la aplicación, este sí puede ser destruido (por ejemplo: cuando el dispositivo es apagado). Es por ello, que el objeto Memoria debe ser almacenado en la memoria No Volátil del dispositivo para evitar su pérdida. Este proceso se realiza dentro del método OnDestroy() del servicio, el cual es llamado justo antes de la destrucción del mismo.

Posteriormente, al iniciar de nuevo el servicio, lo primero que este hace, es comprobar la existencia de dicho objeto en la memoria, y si es así lo obtiene. De esta forma, se evita la perdida de información en todo momento.

### 6.3.2 Actualización periódica de los campos

Para realizar esta labor, el servicio cuenta con un objeto CountDownTimer configurado con el tiempo de actualización definido por el usuario. De modo que cuando este llega al final de su cuenta atrás, comienza el proceso de actualización.

Este proceso comienza con la activación del flag "actualización en curso", y la creación de una serie de hilos (tantos como campos a actualizar) que serán los encargados de realizar las peticiones http con los datos de los campo a actualizar. La imagen 6.3.2-1 muestra la definición de estos hilos.

```
public class HiloUpdateField extends AsyncTask<String,Float,Integer> {
    public boolean finalizado=false;
    private TSField field;

    public HiloUpdateField(TSField field){
        this.field=field;
    }
    @Override
    protected Integer doInBackground(String... urls) {
        field.updateField();
        finalizado=true;
        return 1;
    }
    public TSField getCampo(){
        return field;
    }
}
```

**Imagen 6.3.2-1 Fragmento código actualización**

[28] Ver apartado ACTUALIZACIÓN PERIODICA DE LOS CAMPOS.

Gracias a estos hilos, conseguimos el lanzamiento en paralelo de todas las peticiones, de esta forma aumentamos de manera considerable la velocidad de la actualización, sin necesidad de esperas para que acabe una petición, para comenzar la siguiente.

Además la realización de las peticiones mediante hilos, tiene otra importante ventaja, y es que durante el proceso de actualización, el usuario puede continuar desplazándose por la interfaz sin necesidad de esperas de actualización. Lo cual aumenta la sensación de fluidez y estabilidad de la aplicación. En la Imagen 6.3.2-1 se puede observar la estructura interna de estos hilos. Esta es muy sencilla, solamente cuenta con el campo y un flag para indicar que la actualización ha sido realizada. Para realizar esta actualización el hilo llama a la función updateField() del campo recibido.

En la imagen 6.3.2-2 se muestra como esta función comienza realizando la petición, determinando su estructura en función de su canal, posición en este, y si se trata de una campo publico o no. La estructura de estas peticiones viene marcada por el Api de Thingspeak[20].

```
public void updateField(){
    JSONObject datos=null;
    HttpResponse salida;
    if(getOculto()) {//Comprobamos si el campo es publico
        salida = Tools.hacerGet("http://api.thingspeak.com/channels/" + getChannel()
        + "/fields/"+Integer.toString(getNum())+
        "/last.json?timezone=Europe/Madrid&api_key="+getApiKey(), null);
    }else{
        salida= Tools.hacerGet("http://api.thingspeak.com/channels/" + getChannel()
        + "/fields/"+Integer.toString(getNum())+
        "/last.json?timezone=Europe/Madrid", null);
    }
    if(salida!=null) {//si se ha realizado correctamente
        datos = Tools.obtenerJSON(salida);//transformamos en objeto json
    }
    if (datos != null) {//si no hay error en en la transformacion
        try {//tomo los datos obtenidos en la respuesta y actualizo los valores
        String fecha=datos.getString("created_at");
        setUltimaEntradaFecha(fecha.substring(0, fecha.indexOf('T')));
        setUltimaEntradaHora(fecha.substring(fecha.indexOf('T')+1,fecha.indexOf('+')));
        setUltimaEntradaValor(datos.getString("field" + Integer.toString(getNum())));
        setUltimaEntradaId(datos.getString("entry_id"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

**Imagen 6.3.2-2 Código updateField**

Además, como se puede ver, se solicita la respuesta en forma de objeto Json, pues este formato facilita mucho su posterior manejo.
Una vez obtenida la respuesta, se interpreta esta como un objeto Json para finalmente obtener sus datos y almacenarlos en el campo.

Todo este proceso, se realiza de forma paralela, de modo que un hilo puede finalizar antes que otro, y el encargado de comprobar el estado de estos hilos, será el propio CountDownTimer que ha inicia el proceso de actualización, el cual es reiniciado al final de su cuenta atrás. Este CountDownTimer cuenta con una función llamada tick, la cual es llamada con una periodicidad (configurable) durante la cuenta atrás. En esta función, será la encargada de comprobar el estado de los hilos durante el proceso de actualización, además de desactivar el flag de "actualización en curso", cuando este haya finalizado.

### 6.3.3 Comprobación de notificaciones y alarmas

El funcionamiento de esta parte del núcleo sigue la misma filosofía que la explicada en el apartado anterior.

De nuevo se dispone de un CountDownTimer con el tiempo de comprobación de alarmas, el cual inicia el procedimiento de comprobación cuando este llega al final.

Como se puede observar en la imagen 6.3.3-1, la función *iniciarTimerEventos()*, es la encargada de realizar la comprobación periódica de las alarmas y eventos.
La parte más importante de la comprobación de eventos, está en la función *onFinish()* del CountDownTimer, que será ejecutado cuando este finalice.

Esta función comienza obteniendo los objetos Notificación de la memoria del sistema, y recorriendo cada uno de ellos comprobando si se cumplen las condiciones de aquellas notificaciones que se encuentren activas. Esta comprobación se realiza mediante la función *boolean checkCondiciones(Context);*, la cual devolverá un valor verdadero si se cumplen.
Posteriormente, se comprueba si el evento se encuentra lanzado, y si no es así se lanza, mediante la función *lanzarEvento(Context);*
Por último, una vez comprobados todas notificaciones y eventos, se vuelve a llamar a la función iniciarTimerEventos() para iniciar un nuevo timer.

```
public void iniciarTimerEventos() {
    tEventos = new CountDownTimer(tiempoUpdateEventos * 1000, 1000) {
        @Override
        public void onTick(long millisUntilFinished) {

        }
        @Override
        public void onFinish() {//comienza el proceso de comprobacion de alarmas
            ArrayList<Notificacion> notificaciones=mem.getNotificaciones();
            //tomamos notificaciones
            for(int i=0;i<notificaciones.size();i++){
                if(notificaciones.get(i).isEnabled()) {
                //comprobamos las notificaciones activas
                    if(notificaciones.get(i).checkCondiciones(c)){//si se cumple
                        if(!notificaciones.get(i).lanzada) {//no se encuentra lanzada
                            notificaciones.get(i).lanzarEventos(c);
                            //lanzamos notificacion
                        }
                    }else{
                        notificaciones.get(i).lanzada=false;
                    }
                }
            }
            iniciarTimerEventos();
        }
    };
    tEventos.start();
}
```

Imagen 6.3.3-1 Código comprobación de eventos

Debemos de tener en cuenta que al existir diferentes tipos, tanto la clase Condición como la clase Evento, son clases abstractas, las cuales declaran las funciones y las variables básicas, de forma que las posteriormente las clases extensión definen estos métodos de acuerdo a sus propias necesidades.
En la imagen 6.3.3-2 se muestran las diferentes extensiones creadas de las clases Condición y Evento. Más adelante, se comentará más en profundidad la estructura interna de estas clases[29].

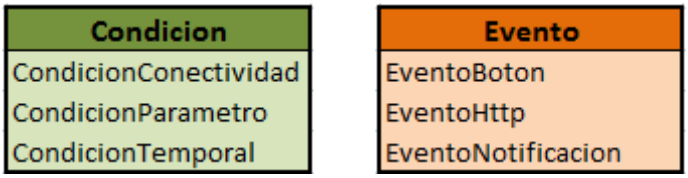



Imagen 6.3.3-2 Clases extensiones de Condición y Evento

[29] Ver apartado ALARMAS Y NOTIFICACIONES.

## 6.4 Conexión Activities - Núcleo

Como bien hemos apuntado anteriormente, los activities se conectan al núcleo para obtener los datos del usuario y el estado actual de los parámetros, es decir, para tomar el objeto memoria.
Esta conexión se establece mediante la clase MiVinculo, extensión de la clase Binder, y que es definida dentro del propio Servicio Núcleo.
Esta clase, actúa de interfaz entre las diferentes actividades y el servicio. Como se observa en la imagen 6.4-1, se define el método *ServicioNucleo getService()*, mediante esta función se devuelve el propio objeto ServicioNucleo.

```
public class MiVinculo extends Binder {
    ServicioNucleo getService() {
        return ServicioNucleo.this;
    }
}
```

Imagen 6.4-1 Código MiVinculo

Por otro lado, en la parte del activity, se define el objeto ServiceConnection, el cual es el encargado de gestionar esta conexión con el servicio a través del interfaz MiVinculo anteriormente definido en el ServicioNucleo.
Como se puede observar en la imagen 6.4-2, el método *onServiceConnected()* que será llamado tras realizar la conexión, obtiene el interfaz MiVinculo que es empleado para la comunicación entre el activity y el servicio a través de su función *getService()* que devolverá el propio objeto ServicioNucleo.

```
public ServiceConnection miServicioConexion = new ServiceConnection() {
    @Override
    public void onServiceDisconnected(ComponentName name) { servicioConectado = false; }

    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        ServicioNucleo.MiVinculo miVinculo = (ServicioNucleo.MiVinculo) service;
        nucleo = miVinculo.getService();
        servicioConectado = true;
    }
};
```

Imagen 6.4-2 Código ServiceConnection

Una vez definidas y explicadas todas las herramientas que emplearemos para establecer la conexión, procedemos a establecerla:

Como se puede observar en la imagen 6.4-3, esta conexión se establece mediante la función *bindService(Intent, Sercive, int)*, a la cual se pasa el objeto intent con el servicio objetivo, el objeto ServiceConnection que gestionará la conexión y la constante BIND_AUTO_CREATE, mediante la cual se define que si el servicio no está en ejecución, este será creado.

```
Intent intent = new Intent(this, ServicioNucleo.class);
bindService(intent, miServicioConexion, BIND_AUTO_CREATE);
```

Imagen 6.4-3 Código conexión con servicio.

De esta forma, queda establecida completamente la conexión entre el servicio, y el activity. Este proceso, será repetido en todos los activities que requieran el uso del objeto Memoria.

## 6.5 Obtención de gráficas

Las gráficas mostradas en la aplicación, son generadas por el propio servidor ThingSpeak, el cual dispone de su propio api para definir las características de esta[20].
Para obtener la gráfica, se realiza una petición http con todas las características de la grafica solicitada. Posteriormente, el servidor Thingspeak contestará con el html que contendrá la gráfica solicitada.

La filosofía seguida en Domote, es la creación de un objeto ParámetrosGrafica cuyo fragmento es mostrado en la imagen 6.5-1, este objeto se encuentra dentro de cada uno de los objetos parámetro (TSField), y en él se almacena todas las características de la gráfica definida por el usuario, tanto en la fase de configuración como en la propia visión general[30].

Los objetos ParametrosGrafica cuentan con la función *String obtenerPeticion()*, la cual genera la URL con todos los parámetros definidos.

```
public class ParametrosGrafica  {
    //-------------------------GRAFICA
    private int color;
    private int tipo;
    private boolean escalonado;

    private boolean valorMinimo;
    private int valorMinimoEjeY;
    private boolean valorMaximo;
    private int valorMaximoEjeY;

    private int periodoAnalisis;
    private int nResultados;
    private int nDias;
    private String inicio;
    private String hasta;

    private boolean timescale;
    private int ntimescale;
    private boolean sum;
    private int nsum;
    private boolean average;
    private int naverage;
    private boolean median;
    private int nmedian;

    public ParametrosGrafica(){
```

Imagen 6.5-1 Fragmento Parámetros Gráfica

[30] Ver apartado FASE DE CONFIGURACIÓN

Por último, solo se tiene que mostrar esta gráfica que ha sido obtenida mediante el objeto WebView, mediante el código de la imagen 6.5-2.

```
WebView web1=(WebView)findViewById(R.id.webView);
web1.getSettings().setJavaScriptEnabled(true);
web1.setWebViewClient(new WebViewClient());
web1.setVerticalScrollBarEnabled(false);
web1.loadUrl(parametrosGrafica.getPeticion());
```

**Imagen 6.5-2 Carga de gráfica con parámetros configurados.**

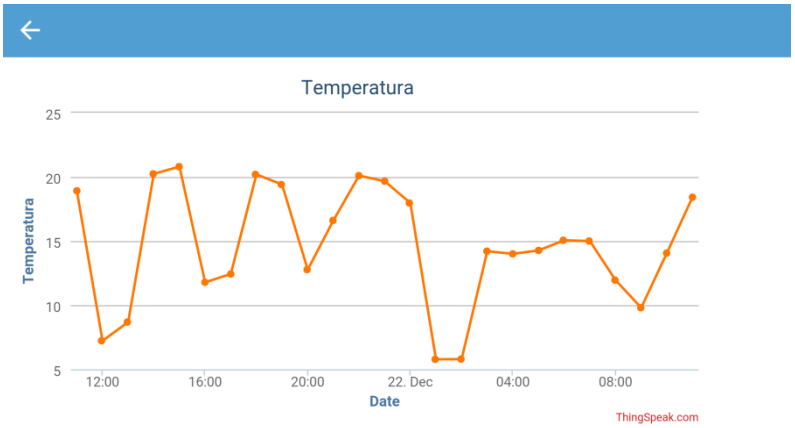



**Imagen 6.5-3 Gráfica Resultante**

## *6.6 Botones*

Los botones han sido creados, para permitir al usuario definir acciones que este puede realizar sobre su sistema, modificando parámetros o el comportamiento del mismo
La filosofía básica de los botones es bastante simple. Se basa en el envío de peticiones http a una dirección URL configurada por el usuario para cada botón.

Atendiendo a las necesidades que se han podido observar en el sistema domótico, se han definido los siguientes botones:

### 6.6.1 Botón Simple

Su funcionamiento es muy básico, enviará la petición cuando el usuario pulse sobre este, o cuando la aplicación reciba la orden de pulsarlo.
Para su configuración solamente se requiere:

- Url destino
- Titulo
- Texto del botón


Imagen 6.6.1-1 Botón Simple

### 6.6.2 Interruptor

Este botón tiene dos posibles estados (activo y desactivo). Es por ello que se hace necesario el establecimiento de un sistema de obtención del estado del mismo. Para ello, el usuario debe ingresar una condición a cumplir por un parámetro del sistema, para que el botón se encuentre en modo activo, solo si esta se cumple, el botón se mostrará como activo, en caso contrario se mostrará como desactivado.

Por otro lado, si el usuario pulsa para cambiar de estado el botón, este se mantendrá en el nuevo estado un tiempo definido por el usuario, hasta que el sistema vuelva a comprobar la condición definida.

Como vemos, este botón requiere de más elementos para su configuración[31]:

- Url destino para ON
- Url destino para OFF
- Titulo
- Parámetro y condición a cumplir para estado On
- Tiempo de espera tras ser pulsado


Imagen 6.6.2-1 Interruptor

[31] Ver apartado VISIÓN GENERAL.

### 6.6.3 Barra de progreso

Se trata del botón más complejo, pues este puede tomar múltiples estados, que deben ser controlados, tanto en el envío en una variable cuando el usuario pulsa sobre él, como en la obtención de su estado actual a través de algún parámetro del sistema.
Para ello, el usuario debe de ingresar primero el rango de valores que podrá tener dicha barra, así como el campo del sistema de donde se obtendrá su estado actual.
Por último, se vuelve a definir un tiempo de espera entre que el usuario cambia de estado el botón hasta que se vuelve a comprobar su nuevo estado.
Este botón requiere de los siguientes campos para su configuración[32]:

- Titulo
- Rango de valores de la barra
- Url destino
- Nombre del parámetro junto al que se enviará
- Parámetro del que se toma su estado
- Tiempo de espera tras ser pulsado

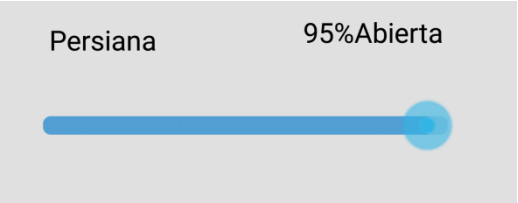



**Imagen 6.6.3-1 Barra de Progreso**

### 6.6.4 Evento de pulsado

Cuando el usuario pulsa sobre cualquiera de estos tres tipos de botones, el funcionamiento es similar al comentado en el proceso de actualización de los campos[33].
Se crea un hilo que realizará la petición contenida en la configuración del botón, y de nuevo, paralelamente, este hilo es controlado por un CountDownTimer, de forma que si la petición falla, o se demora, el timer iniciará el proceso para comunicárselo al usuario.

En caso de que todo el proceso se realice correctamente, se iniciará un nuevo timer, con el tiempo de espera configurado por el usuario, para que una vez finalizado este, solicitar al Núcleo la actualización de los campos, y con ello también el estado de los botones.

[32] Ver apartado VISIÓN GENERAL.
[33] Ver apartado ACTUALIZACIÓN PERIÓDICA DE LOS CAMPOS.

## *6.7 Estados de la Aplicación*

Como hemos visto, el servicio núcleo sigue funcionando aun cuando la aplicación se encuentra cerrada, o bien esta se encuentra en segundo plano[34].
El funcionamiento del núcleo, debe ser diferente en dichos casos, puesto que un funcionamiento constante y continuo del mismo podría provocar un consumo excesivo tanto de datos como de batería.

Por ello, se han establecido 3 modos de funcionamiento:

- Aplicación abierta y en primer plano
- Aplicación abierta en segundo plano
- Aplicación cerrada

A continuación se procederá a explicar el comportamiento de la aplicación en cada uno de estos modos:

### 6.7.1 Aplicación Abierta en primer plano

Se trata del modo de funcionamiento explicado hasta ahora, en él, **el núcleo actualiza todos los campos del sistema** según el tiempo de actualización establecido, independientemente del tipo de conexión que se esté empleando. Este comportamiento, es debido a que se pretende mostrar al usuario el estado actual de los campos actualizados en la interfaz gráfica, evitando provocar una sensación de lentitud o mal comportamiento de la aplicación.
Por otro lado, al mismo tiempo se va comprobando el estado de las alarmas y notificaciones.

### 6.7.2 Aplicación Abierta en segundo plano

En este modo, no se tiene porque mostrar al usuario el valor actual de los campos, pues no está utilizando en este momento la aplicación. Si bien es cierto que cabe la posibilidad de que este vuelva a emplearla próximamente. Es por ello, que sería preferible seguir manteniendo la actualización de los campos de forma que cuando este vuelva a entrar, pueda observar su valor actual, sin necesidad de ningún tiempo de espera.

Pensando en cuál sería el funcionamiento ideal, y la posibilidad de un consumo excesivo de datos, se ha definido un comportamiento diferente en función al tipo de conexión que se esté empleando:

- Si el dispositivo se encuentra conectado a una red wifi: el funcionamiento se mantiene del mismo modo que cuando se encuentra en primer plano, es decir, se actualizan todos los campos en cada ciclo de actualización.

[34] Ver apartado SERVICIO NÚCLEO.

- En caso de que el dispositivo esté empleando la red móvil: solamente se actualizarán los campos que necesiten ser actualizados para el control de las alarmas y notificaciones.

### 6.7.3 Aplicación Cerrada

En este caso, sigue siendo necesario comprobar periódicamente las notificaciones y alarmas de forma que estos salten si se cumplen las condiciones, aún cuando el usuario no haya abierto la aplicación

Por otro lado, no se necesita mostrar ninguna información al usuario, pues este no ha abierto la aplicación y tampoco parece que en un periodo corto de tiempo vaya a hacerlo.

Por ello, en este estado solamente se realizará la actualización de aquellos campos que sean empleados por las notificaciones y eventos.

## *6.8 Alarmas y Notificaciones*

Ya hemos visto como el Núcleo disponía de un array de objetos Notificación con todas las alarmas y notificaciones definidas por el usuario, los cuales actualizaba y lanzaba empleando los métodos checkCondiciones() y lanzarEventos()[35], como se muesta en la imagen 6.8-1.

```
public boolean checkCondiciones(Context c) {
    boolean resultado=true;
    for (int i = 0; i < condiciones.size(); i++){
        resultado=resultado&&condiciones.get(i).checkCondicion(c);
    }
    return resultado;
}
```

```
public void lanzarEventos(Context c){
    if(!lanzada){
        lanzada=true;
        for(int i=0;i<eventos.size();i++){
            eventos.get(i).lanzarEvento(c, nombre, descripcion);
        }
    }
}
```

**Imagen 6.8-1 Métodos lanzarEventos() y checkCondiciones()**

[35] Ver apartado COMPROBACIÓN DE NOTIFICACIONES Y ALARMAS.

Estos métodos no hacen más que recorrer el array de elementos Condiciones y Eventos ejecutando las funciones checkCondiciones() y lanzarEventos() de forma individual como se muestra en imagen 6.8-1.
Estos métodos, son implementados de forma diferente en función del tipo de Evento y Condiciones que se trate como veremos a continuación.

| Notificacion | |
|---|---|
| String | nombre |
| String | descripcion |
| ArrayList<Condicion> | condiciones |
| ArrayList<Evento> | eventos |
| boolean | atendida |
| boolean | lanzada |
| boolean | enabled |

Imagen 6.8-2 Esquema clase Notificación

### 6.8.1 Condiciones

La clase Condición, se trata de una clase abstracta (Como se puede comprobar en la imagen 6.8.1-1), en la cual se han declarados sus métodos y variables, para que posteriormente cada una de sus extensiones los definen en función a sus necesidades.

```
public abstract class Condicion implements Serializable {
    public boolean estado=false;
    public abstract boolean checkCondicion(Context c);
}
```

Imagen 6.8.1-1 Definición clase Condición

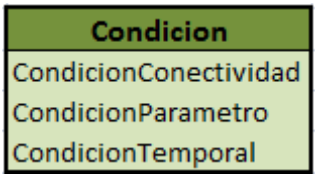



Imagen 6.8.1-2 Extensiones de condición

En este caso se han creado tres extensiones:

- **CondicionConectividad:** Estará dedicada a comprobar condiciones relativas a la conectividad del dispositivo, tales como estar conectados a una red wifi determinada o estar empleando la red móvil
- **CondicionParametro:** Toma un campo de los definidos en el sistema, el cual es actualizado por el núcleo, y comprueba si este toma un valor determinado.
- **CondicionTemporal:** Dedicado a cuestiones de temporales, este puede comprobar por ejemplo: si la hora actual está dentro de un rango temporal determinado, o si no ha pasado de una hora determinada.

Debido al amplio rango de posibilidades que ofrecen cada uno de estos tipos condiciones, la definición de sus métodos *checkCondición()* resulta algo extensa para mostrarla en esta memoria, por ello se muestran pequeños fragmentos que ilustran su comportamiento:
En el fragmente de CondicionConectividad de la imagen 6.8.1-3, se puede ver como en función del tipo de condición de conectividad definida por el usuario, el método comprueba si el dispositivo se encuentra a una red wifi (Casos 0 y 1), o si está conectado a una red wifi determinada (Caso 2). Posteriormente devuelve el resultado almacenado en la variable estado.

```
public boolean checkCondicion(Context c){
    switch (tipo){
        case 0:
            estado=networkInfo.isConnected();
            break;
        case 1:
            estado=!networkInfo.isConnected();
            break;
        case 2:
            if (connectionInfo != null && !(connectionInfo.getSSID().equals(""))) {
                estado=connectionInfo.getSSID().equals(ssid);
                break;
            }else{
                estado=false;
                break;
            }
```

**Imagen 6.8.1-3 Fragmento checkCondicion() en CondiciónConectividad**

En el fragmento de CondiciónParametro de la imagen 6.8.1-4, podemos ver como toma el último valor del parámetro, y se compara según el tipo de condición con el valor introducido por el usuario. Posteriormente se devuelve el resultado almacenado en estado.

```
public boolean checkCondicion(Context c) {
    estado = false;
    switch (tipo){
        case 0:
            estado=(parametro.getUltimoValorReal()==valor);
            break;
        case 1:
            estado=(parametro.getUltimoValorReal()!=valor);
            break;
        case 2:
            estado=(parametro.getUltimoValorReal()<valor);
            break;
        case 3:
            estado=(parametro.getUltimoValorReal()>valor);
            break;
    }
    return estado;
}
```



**Imagen 6.8.1-4 Fragmento checkCondicion() en CondiciónParámetro**

Por último, en el caso de CondiciónTemporal de la imagen 6.8.1-5, se puede comprobar cómo en el fragmento se toma la hora actual, y como en el caso 1 y 2, se comprueba si nos encontramos dentro o fuera de un rango horario determinado.

```
public boolean checkCondicion(Context c) {
    final Calendar c1 = Calendar.getInstance();
    estado=false;
    int hora = c1.get(Calendar.HOUR_OF_DAY);
    int minuto = c1.get(Calendar.MINUTE);
    Date diaActual=c1.getTime();
    switch (tipo){
        case 1: case 2:
            int inicio=horaI*60+minutoI;
            int fin=horaF*60+minutoF;
            int actual=hora*60+minuto;
            if(fin>inicio){
                if((actual>inicio)&&(actual<fin)){
                        estado=true;
                        break;
                }else{
                    estado=false;
                    break;
                }
```

Imagen 6.8.1-5 Fragmento checkCondicion() en CondiciónTemporal

## 6.8.2 Eventos

De nuevo, la clase Evento, se trata de una clase abstracta, la cual se han declarado sus métodos y variables, para que posteriormente cada una de sus extensiones los definan en función a sus necesidades.

```
public abstract class Evento implements Serializable{
    public abstract String getText();
    public abstract void lanzarEvento(Context c,String nombre,String descripcion);
}
```

Imagen 6.8.2-1 Definición clase Evento


Imagen 6.8.2-2 Extensiones de Evento

En este caso se han creado tres extensiones:

- **EventoHttp:** Este evento realiza una petición Http a la URL introducida al usuario.
- **EventoBoton:** Este evento, realizará la acción de modificar un botón de la misma forma que lo haría el usuario. Este tipo de evento podría ser creado también como EventoHttp, pero se ha definido como tal, por simplicidad de uso de cara al usuario.
- **EventoNotifición:** Este evento lanza una notificación en el dispositivo móvil, las características de esta notificación (Tales como sonido, vibración, o color del led de notificaciones) pueden ser editadas por el usuario.

A continuación se muestran los fragmentos de código con la implementación del método lanzarEvento() en cada uno de estos tipos:

En el caso de EventoHttp Y EventoBoton sigue la filosofía ya seguida en múltiples partes del código de la aplicación. Como vemos en la imagen 6.8.2-3, de nuevo se crea un Hilo que será el encargado de realizar la propia petición, mientras que un objeto CountDownTimer controla su correcto funcionamiento.

```
public void lanzarEvento(Context c,String nombre,String descripcion){
    final HiloPeticion miHilo=new HiloPeticion(peticion);
    CountDownTimer t=new CountDownTimer(10000,1000) {
        @Override
        public void onTick(long millisUntilFinished) {
            if(miHilo.finalizado!=0){
                this.cancel();
                this.onFinish();
            }
        }
        @Override
        public void onFinish() { miHilo.cancel(true); }
    };
    miHilo.execute();
    t.start();
}
```

**Imagen 6.8.2-3 Método lanzarEvento() en EventoHttp y EventoBoton.**

En el caso de EventoNotificación, se puede observar en la imagen 6.8.2-4 cómo se da a la notificación las propiedades que el usuario le ha asignado, para posteriormente solicitar al sistema el lanzamiento de esta.

```
public void lanzarEvento(Context c,String nombre,String descripcion){
    NotificationManager nm = (NotificationManager)c.getSystemService(c.NOTIFICATION_SERVICE);
    NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(c)
            .setSmallIcon(R.drawable.notificacion)
            .setContentTitle(nombre)
            .setContentText(descripcion).setVibrate(getVibracion());
    //-------------------------ponemos accion al click en la notificacion
    Intent resultIntent = new Intent(c, HomeActivity.class);
    PendingIntent resultPendingIntent = PendingIntent.getActivity(c,0,resultIntent
    ,PendingIntent.FLAG_UPDATE_CURRENT);
    mBuilder.setContentIntent(resultPendingIntent);
    //-----------------------------------------------------------
    if(tono){
        mBuilder.setSound(Uri.parse("android.resource://"
                + c.getPackageName() + "/" + Tools.SONIDOS[numTono]));
    }
    if(led){
        mBuilder.setLights(Tools.COLORES[colorLed],500,500);
    }
    mBuilder.setAutoCancel(true);
    nm.notify(ID_NOTIFICACION_CREAR, mBuilder.build());
}
```

Imagen 6.8.2-4 Método lanzarEvento() en EventoNotificación

## 6.9 Asistente de Voz

Se trata de uno de los factores diferenciales de la aplicación. Un asistente de voz que es capaz de reconocer las ordenes vocales que recibe del usuario y actuar en consecuencia.

Para su implementación se han empleado dos herramientas:

- **Plataforma Api.ai:** Encargada de obtener el audio recogido por el micrófono del dispositivo, pasar este a texto a través del servicio de reconocimiento de voz de Google, y posteriormente procesar este texto y determinar si se trata de alguno de los comandos definidos.

- **Sintetizador TTS de Android:** Servicio proporcionado por el propio sistema operativo, el cual es capaz de generar archivos de audio a partir de un texto que se le proporciona.

### 6.9.1 Api.ai

Es una plataforma gratuita, que permite a cualquier desarrollador a través de su SDK establecer su propio asistente de voz (Como puede ser Siri, Google Now o Cortana) mediante una interfaz fácil y simple.

Su funcionamiento se basa en 4 pasos:

1º Recoge el audio del micrófono del dispositivo.

2º Analiza el audio recogido a través del servicio de reconocimiento de voz de Google, pasando este a texto.

3º Envía el texto generado al servidor api.ai, que lo procesa, comparando este con las ordenes ya definidas previamente.

4º El servidor responde si ha identificado el texto con una de las ordenes definidas.
Es en este último paso, el procesado, en el que se define al asistente y será comentado en profundidad.

El registro en api.ai es muy sencillo, además, su web cuenta con una gran cantidad de documentación y tutoriales para emplear api.ai en múltiples plataformas.
Una vez el usuario se ha registrado, se muestra un menú desplegable con los agentes de voz creados por el usuario, así como la opción para crear uno nuevo. Tras crearlo, (introduciendo simplemente el nombre, idioma, y franja horaria por defecto), se muestra en el menú lateral los elementos "Intent" y "Entity" (que serán descritos en profundidad más adelante), así como herramientas como Logs y Dominos, como se muestra en la imagen 6.9.1-1.

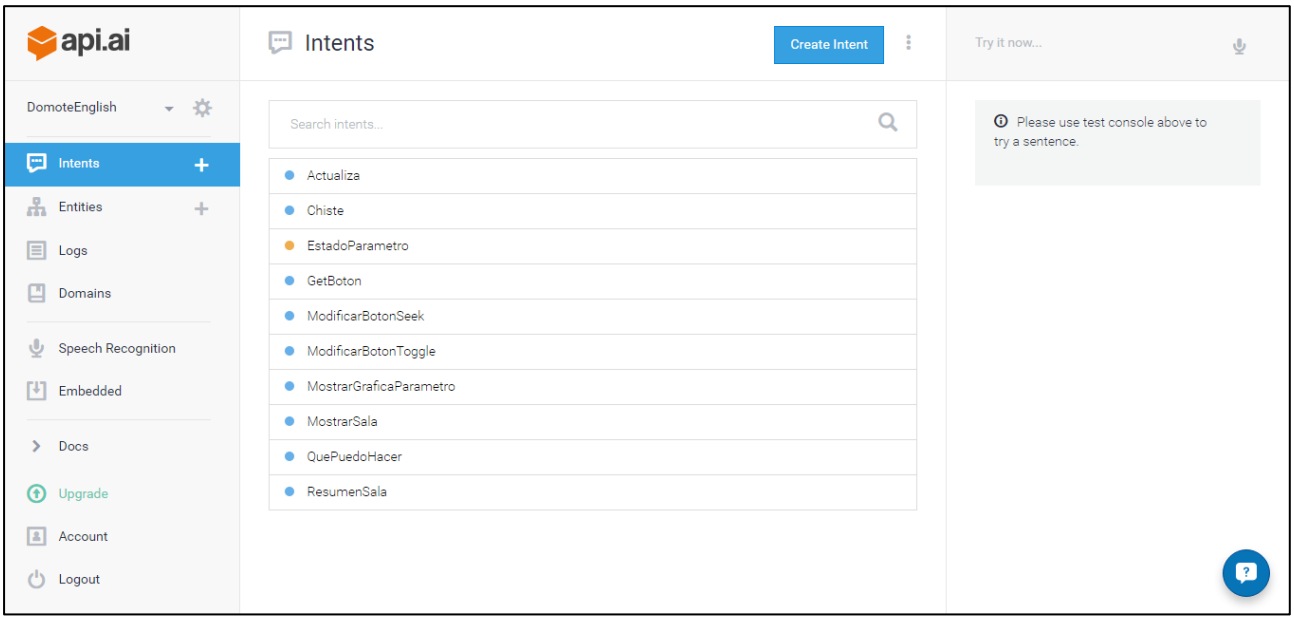



**Imagen 6.9.1-1 Pantalla de edición agente Api.ai**

A continuación se procederá a describir estos elementos que definen al agente de voz:

## Entities:

Son grupos de palabras, los cuales engloban palabras del mismo tipo o significado. Por ejemplo, hemos definido las entidades "Salas" las cuales englobará todos los nombres de las salas creadas por el usuario, o la entidad "Estado Interruptor" que incluirá las palabras "Encendido, activo, On…".

En la imagen 6.9.1-2 se muestra una tabla con todos los entities definidos para nuestra aplicación.

**Imagen 6.9.1-2 Entities definidos**

| | | |
|---|---|---|
| @Botones | Boton | |
| @De | De | de, del, de la, de los, de la |
| @En | En | en, en la, en el |
| @EstadoInterruptor | On | On, encendido, activo, activa, abre, enciende, dar |
| | Off | Off, apagado, desactivo, desactiva, desactivar, cierra, apaga, quita |
| @Grafica | Grafica | Gráfica, gráfico |
| @OrdenesModificar | OrdenesModificar | Pon, modifica, baja, sube, puedes subir, puedes bajar, subir, bajar puede subir, puede bajar, poner, modificar |
| @OrdenesMuestra | OrdenesMuestra | Muestra, muéstrame, enseña, vamos a, ve a, quiero ver, vete a |
| @Parametros | Parametro | |
| @Preguntas | Preguntas | Cuál es, Dime, Cómo está, dame |
| @Salas | Sala | |
| @Valor | Valor | Valor, estado, último valor, estado actual, actual estado último estado, valor actual |

Como se observa, los entities Botones, Salas y Parámetros, no están definidos ya en el servidor. Esto es debido a que estos entities se definirán de forma personalizada para cada usuario, con sus propios elementos. De esta forma, la plataforma podrá identificar cuando el usuario esta pronunciando uno de sus entities. Estos entities son enviados en el mismo mensaje en el que se envía el texto con las palabras dichas por el usuario.

## Intents:

Se emplean para especificar las posibles frases y ordenes que puede decir el usuario. Puesto que la forma de expresar una misma orden puede variar mucho de un usuario a otro, los intents pueden estar compuestos por los entities creados previamente. De esta forma, se obtiene una gran flexibilidad a la hora de definir todas las posibles combinaciones de las que dispone el usuario para ejecutar una misma orden. En la imagen 6.9.1-3 se muestran los diferentes elementos Intents definidos en la definición del agente de voz de Domote.

A continuación se muestran algunos ejemplos que ilustran como es el procesado de los comandos dichos por el usuario con los Entities e Intents definidos:

- "¿Cual es la temperatura actual?"
  - Cual es la -> @Preguntas
  - temperatura -> @Parametros

"¿Cual es la temperatura actual?" => @Preguntas + la + @Parametros + actual

| | |
|---|---|
| **Actualiza** | Actualiza [el sistema, los datos] |
| | Toma de nuevo los datos |
| | Actualizate |
| | Actualización |
| **Chiste** | Cuéntame un chiste |
| | Cuenta un chiste |
| | chiste |
| **EstadoParametro** | @Preguntas @Valor @Parametros |
| | @Preguntas @Valor @De @Parametros |
| | @Preguntas @Parametros @De @Sala |
| | @Preguntas [el, la] @Parametros [actual] |
| **GetBoton** | @Preguntas @Valor @De [botón] @Boton |
| | @Preguntas [botón] @Botones |
| **ModificarBarra** | @OrdenesModificar [el, el boton, la] @Boton [a] @sys.number [porciento, grados] |
| | @Ordenes [el, la] @Valor [@De] @Botones [a] @sys.number [porciento, grados] |
| | @Botones [a] @sys.Number [porciento, grados] |
| **ModificaInterruptor** | @OdenesModificar [botón] @Botones [a] @EstadoInterruptor |
| | @EstadoInterruptor [la, las, el] [botón] @Botones [@De] [sala] [@Salas] |
| **MostrarGraficaParametro** | @OrdenesMuestra @Grafica [@De] [parámetro] @Parametros |
| | @Grafica @Parametros @Salas |
| **QuePuedoHacer** | Qué puedes hacer por mi |
| | Qué puedes hacer |
| **ResumenSala** | @Preguntas @Valor @De [sala] @Sala |
| | @Preguntas [sala] @Salas |

**Imagen 6.9.1-3 Intents definidos**

Como vemos, la combinación @Preguntas+la+@Parametros+actual corresponde al intent **EstadoParametros**.

Pero esta orden puede ser dicha de otra forma:

- "Dime la temperatura"
  - Dime -> @Preguntas
  - temperatura -> @Parametros

"Dime la temperatura" => @Preguntas + la + @Parametros

De nuevo esta combinación se trataría del comando o intent **EstadoParametros.**
Finalmente, una vez analizado el comando dicho por el usuario como hemos visto, el servidor responderá con el Intent correspondiente, así como los objetos entity que lo conforman, y el texto de respuesta.
Una vez que la aplicación recibe este mensaje con el intent ya procesado, esta solo tiene que ejecutarla el comando correspondiente.

Además otra de las ventajas de la plataforma Api.ai es la existencia de un registro, en el cual, se quedaran almacenados todos los textos que no se identifican con ninguno de los intents definidos. De esta forma, el desarrollador puede mejorar con el paso del tiempo

el comportamiento de su asistente vocal, modificando la definición de sus intents con aquellas posibilidades que no haya tenido en cuenta en un principio.

### 6.9.2 Integración de Api.ai en Android

Para llevar a cabo la integración del servicio Api.ai dentro de la aplicación Android lo primero que se debe de realizar es la agregación de las librerías que Api.ai emplea para su compilación, las cuales son las siguientes:

```
'ai.api:sdk:1.9.0@aar'
'com.android.support:appcompat-v7:23.2.1'
'com.google.code.gson:gson:2.3'
'commons-io:commons-io:2.4'
```

Una vez hecho esto, ya se puede proceder a emplear las clases que el SDK de api.ai define para su fácil uso. Este SDK define el servicio **AIService**, este es el encargado de realizar todas las acciones para la resolución del comando de voz introducido por el usuario. Para iniciar este servicio, es necesario definir previamente dos objetos AIListener y AIConfiguration:

**AIListener** es el interfaz encargado de comunicar el estado del servicio con el resto de la aplicación, este cuenta con las siguientes funciones que se definirán en función de las necesidades de cada aplicación (Ver imagen 6.9.2-1).

```
public interface AIListener {
    void onResult(AIResponse result); // Respuesta de api.ai
    void onError(AIError error); // Ha ocurrido un error
    void onAudioLevel(float level); // Cambio de nivel de audio
    void onListeningStarted(); // Se procede a escuchar
    void onListeningCanceled(); // Se ha cancelado la escucha
    void onListeningFinished(); // La escucha ha sido finalizada
}
```

Imagen 6.9.2-1 Definición AIListener

La función más importante de este interfaz, es onResult(AIRespone), la cual será llamada por el servicio, cuando este reciba una respuesta del servidor Api.ai tras la introducción de un comando de voz. A continuación, en la imagen 6.9.2-2 se muestra un fragmento de la función onResult() definida en Domote. Como se puede observar, la función comprueba si el resultado de la consulta corresponde a alguno de los comandos (Intents) definidos en la plataforma, actuando en consecuencia. Por ejemplo en este caso, cuando se trata del comando "Que puedo hacer" muestra en la pantalla la vista con todos los comandos de voz que el usuario puede utilizar.

```
@Override
public void onResult(AIResponse aiResponse) {
    if(aiResponse.getResult().getAction().length()==0) {
        nucleo.hablar("I am sorry, but I have not understood you");
        finish();
    }else if(aiResponse.getResult().getAction().equals("QuePuedoHacer")) {
        //debemos de tener en cuenta que entraremos aqui si el boton existe
        nucleo.hablar(aiResponse.getResult().getFulfillment().getSpeech());
        ayuda.setVisibility(View.VISIBLE);
    }
```

Imagen 6.9.2-2 Fragmento onResult()

El segundo elemento a definir para emplear el servicio Api.ai es el objeto AIConfiguration, en el que se definen la configuración básica del asistente a utilizar, y cuyos parámetros son:

- **Client access token**, una cadena de caracteres generado por api.ai y que es empleado para verificar el acceso al asistente.
- **Idioma soportado,** idioma en el que se recibe el comando de voz
- **Sistema de reconocimiento,** software encargado del reconocimiento de voz

En la imagen 6.9.2-3 se muestra la creación de este objeto de configuración en el código de Domote cuando el idioma del sistema es el español.

```
AIConfiguration config = new AIConfiguration("[redacted]",
AIConfiguration.SupportedLanguages.Spanish, AIConfiguration.RecognitionEngine.System);
```

Imagen 6.9.2-3 Definición objeto AIConfig

Una vez ya se han definido tanto el objeto AIConfiguration como el interfaz AIListener, se puede proceder a la creación del servicio AIService (Ver imagen 6.9.2-4).

```
AIService aiService = AIService.getService(this, config);
aiService.setListener(listener);
```

Imagen 6.9.2-4 Inicio AIService

Como se puede observar, se inicia pasando el objeto de configuración, para posteriormente definir el interfaz que comunicará el servicio y aplicación.

Una vez configurado el servicio, solo se tiene que iniciar el proceso de escucha mediante el startListening(), ver imagen 6.9.2-5.

```
aiService.startListening();
```

Imagen 6.9.2-5 Inicio de escucha

# CAPÍTULO 7:

# SEGURIDAD DEL SISTEMA DOMÓTICO

# CAPÍTULO 7: SEGURIDAD DEL SISTEMA DOMÓTICO

## *7.1 Introducción*

Tal y como se ha enfocado en este proyecto, la información el sistema domótico gestiona está dentro del ámbito domestico, personal y por supuesto privado. Es por ello, que el análisis de esta apartado del sistema es vital para asegurar la privacidad e integridad del mismo, y de esa forma dotar de viabilidad al proyecto.

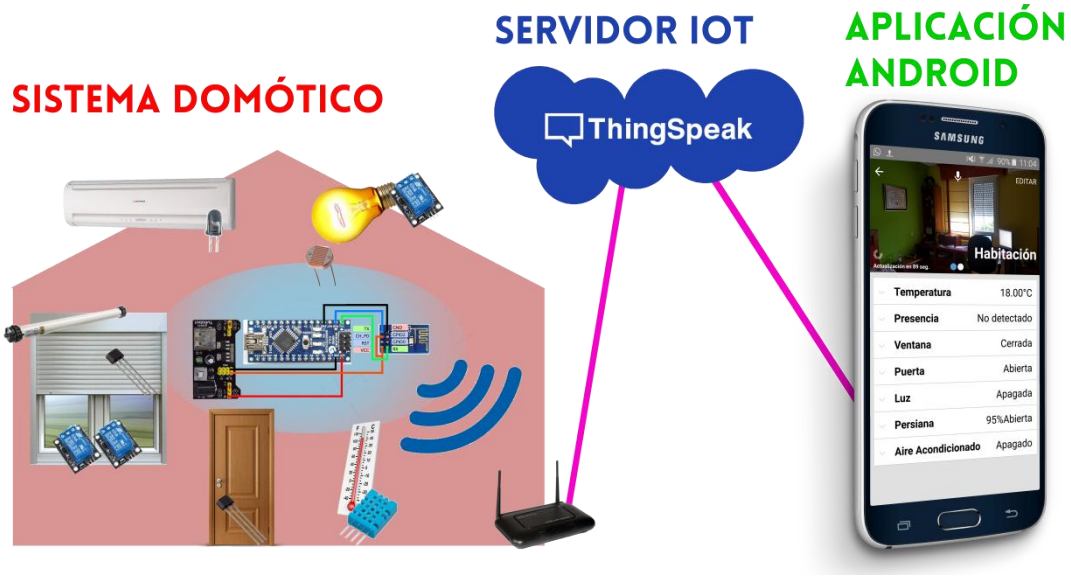



Imagen 7.1-1 Estructura del sistema domótico

Analizando la estructura del sistema (Ver imagen 7.1-1), se observan dos aspectos fundamentales los cuales deben ser especialmente cuidados ya que pueden ser fuente de ataques, estos son:

- **Comunicaciones**: Las comunicaciones entre los diferentes elementos del sistema es uno de los puntos críticos del sistema. Pues durante su transmisión, si esta no se realiza correctamente, se corre el riesgo de escucha por parte del un posible atacante, pudiendo obtener credenciales o valores actuales de campos del sistema.
- **Almacenamiento de datos**: Se debe de tener especial cuidado con el almacenamiento de los parámetros que va obteniendo el sistema domótico, así como de las claves de acceso a los mismos.

## *7.2 Servicios de seguridad ThingSpeak*

Tal y como se aprecia en la imagen 7.1-1, el centro del sistema domótico se encuentra en el servidor Thingspeak. Esta topología, hace que el servidor sea el encargado de aportar gran parte de los servicios y soluciones de seguridad requeridos en el sistema. A continuación, se definirán algunas de estas soluciones.

### 7.2.1 Apikeys

En la plataforma Thingspeak se almacenan los valores de los campos, por ello su primer servicio de seguridad que ofrece es el de **control de acceso a los datos,** mediante el establecimiento de Apikeys. Estas son cadenas de caracteres generadas aleatoriamente por la propia plataforma y asignadas individualmente a los diferentes canales. Gracias a ellas se puede acceder (Apikey de lectura) o escribir (Apikey de escritura) a los campos de un determinado canal definido en el servidor. Estas cadenas son evidentemente privadas, y son requeridas por el servidor cuando se realiza una acción sobre estos canales.
Por otro lado, Thingspeak asigna a cada usuario un apikey de lectura general, el cual es requerido para obtener los canales definidos por un usuario.
Todos estos apikeys, son visibles en la plataforma, y en caso de que estos se vean comprometidos se pueden fácilmente regenerar como se muestra en la imagen 7.2.1-1.

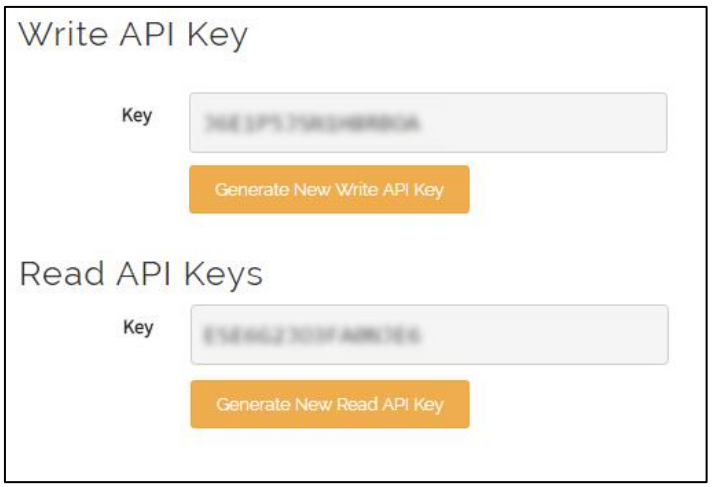



**Imagen 7.2.1-1 Generación apikeys plataforma Thingspeak**

### 7.2.2 Conexiones SSL

Thingspeak ofrece la posibilidad de realizar todas las conexiones con la plataforma de forma segura bajo la capa de seguridad SSL. Esta capa de seguridad ofrece un cifrado extremo a extremo de todo el tráfico web transmitido sobre ella, la cual proporciona confidencialidad, autenticación e integridad. De esta forma, todas las acciones que se

realicen en el servidor, ya sea para actualizar, modificar, obtener graficas, etc... estarán seguras bajo este servicio.

Lamentablemente, el microcontrolador Arduino Nano que se está empleando en este proyecto[36], no tiene la capacidad de procesado suficiente como para implementar esta capa de seguridad, es por ello, que en las comunicaciones con el núcleo del sistema domótico, esta capa de seguridad no podrá ser empleada. Más adelante se podrán observar sus consecuencias, y como este problema puede ser subsanado.

## *7.3 Ataques al sistema domótico*

Una vez analizado los puntos críticos del sistema domótico, se van a realizar una serie de ataques al mismo, tratando de auditar la seguridad del mismo, y para posteriormente poner en relieve posibles vulnerabilidades y plantear soluciones.
Para llevar a cabo estos ataques, se empleará la distribución para auditoria de redes WifiSlax[28] en su versión 4.11, y sus herramientas Wireshark[30] y Ettercap[29]

### 7.3.1 Ataque MITM (Hombre en el medio) al núcleo de sistema

Tal y como hemos visto, una de las debilidades del sistema es la ausencia de cifrado extremo a extremo de las comunicaciones con el núcleo del sistema. En este ataque, nos aprovecharemos de esa debilidad, registrándonos en la red en la que se encuentra el núcleo (Es decir el atacante debe conocer previamente la clave de acceso a la red), y haciéndonos pasar por el router mediante un envenenamiento de ARP, como se muestra en la imagen 7.3.1-1.

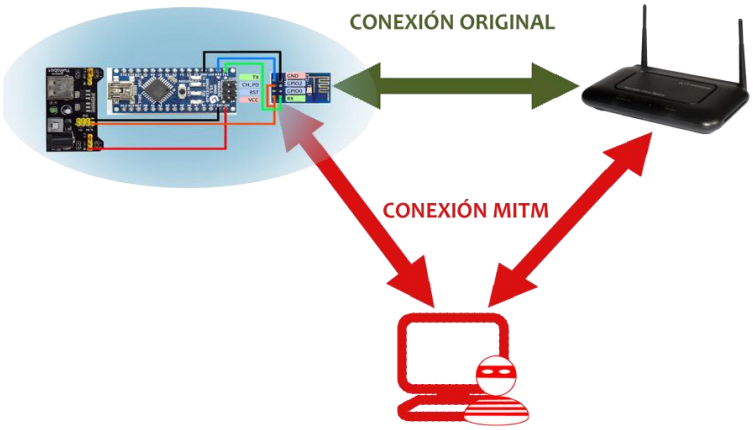



Imagen 7.3.1-1 Ataque MITM

Para lograr situarnos en el medio de la comunicación empleamos Ettercap, una suite con múltiples herramientas para la auditoria de seguridad de red, con una interfaz gráfica

[36] Ver apartado NÚCLEO DEL SISTEMA DOMÓTICO

amigable que permite al usuario su utilización de una forma simple (Ver imagen 7.3.1-2).


Imagen 7.3.1-2 Ettercap

Una vez arrancada, solo se tiene que seleccionar el interfaz de red a utilizar para realizar la monitorización de red, dentro de su menú "Sniff". Posteriormente, una vez seleccionada la interfaz, se muestran en la barra de menú todas las herramientas de las que Ettercap dispone como se muestra en la imagen 7.3.1-3.


Imagen 7.3.1-3 Ettercap, interfaz de red seleccionada

En este punto se procede a realizar un análisis de los dispositivos conectados a la red, para posteriormente, seleccionar cuáles de ellos serán víctimas del ataque que se realizará. En este caso la victima a seleccionar será el núcleo del sistema domótico el cual tiene asignada la dirección ip 192.168.1.40. Una vez seleccionada la víctima, Ettercap se muestra como en la imagen 7.3.1-4.

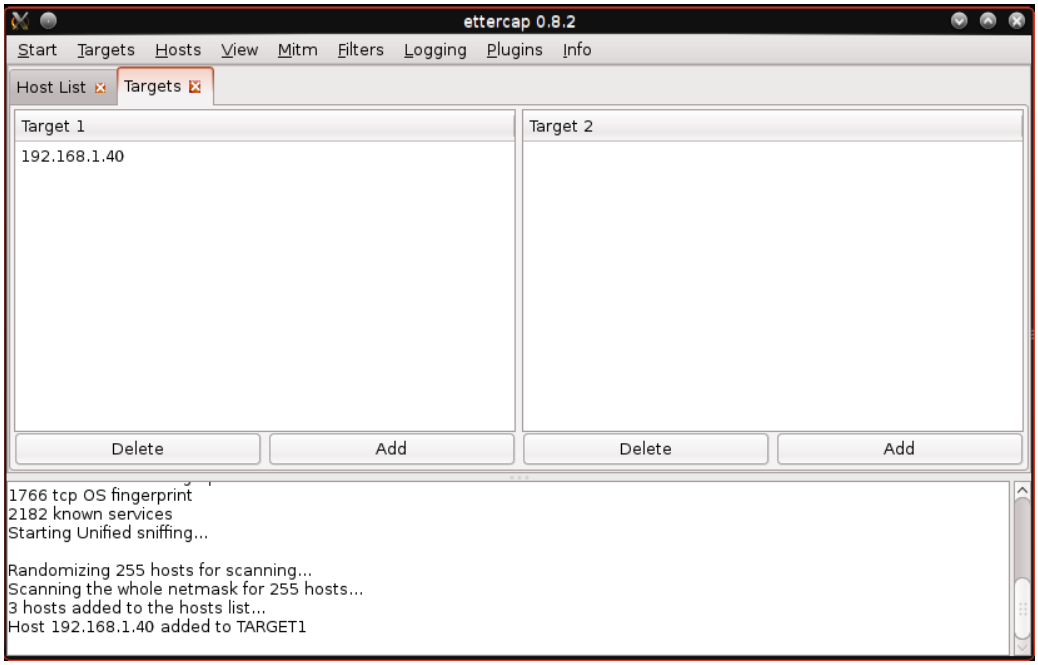



**Imagen 7.3.1-4 Ettercap, victima núcleo seleccionada**

Llegados a este punto, solo queda iniciar el ataque Mitm por envenenamiento de arp seleccionando dicho ataque dentro del menú Mitm, de esta forma comenzará el ataque.

Situados el medio de la comunicación entre el núcleo del sistema y el router, solo queda analizar el tráfico que se está generando mediante el analizador de protocolos Wireshark. Con este programa, si se selecciona la interfaz de red a monitorizar, y se realiza un filtrado de solo los paquetes http, se puede observar los mensajes que el núcleo del sistema está enviando al servidor Thingspeak, los cuales contienen la autenticación con el Apikey de escritura de los canales que actualiza, así como los valores de los parámetros que se están actualizando (Ver imagen 7.3.1-5).
Como se observa, mediante este ataque, el atacante podría obtener el apikey de Escritura de los canales pará su posterior manipulación. Por otro lado, también obtiene el valor actual de los campos que gestiona el sistema, tales como la detección de movimiento en las salas o la apertura de puertas y ventanas, que puede poner en riesgo la seguridad de la estancia.

**Soluciones:**
Como vemos, la solución ideal sería el poder implementar la capa de seguridad SSL en las comunicaciones que lleva a cabo el núcleo de red, ya que de esta forma, aun que el atacante pudiese obtener el tráfico que se está intercambiando con el router, este cifrado, por lo que su contenido seria ilegible.

Lamentablemente como hemos dicho, el núcleo de red no tienen capacidad de procesado suficiente como para implementar esta capa de seguridad, por lo que una solución sería la cambio de microprocesador del núcleo por uno más potente, como puede ser por ejemplo una Raspberry Pi, aun que esta solución aumentaría ostensiblemente los costes del sistema.

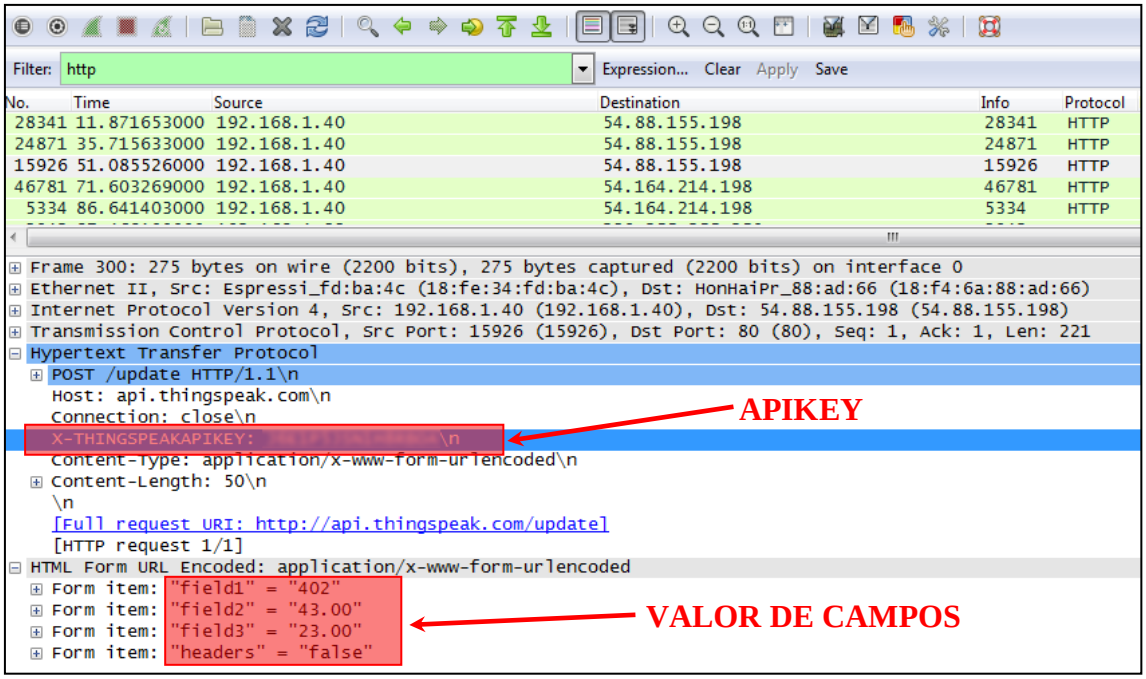



Imagen 7.3.1-5 Wireshark, Análisis MITM a núcleo.

Una solución más realista, es el cuidar la seguridad de la red a la que el núcleo de red se encuentra conectado, ya que como hemos visto al comienzo, el atacante debe de conocer la clave de dicha red y estar conectado a ella. Si la seguridad de esta es suficientemente buena, el atacante no podrá llevar a cabo su ataque. Por ello, como medidas para asegurar la seguridad de estas redes, se recomienda usar seguridad WPA o WPA2 en la red inalámbrica en la que el núcleo del sistema se encuentra conectado (quedando altamente desaconsejado WEP). Otras medidas de seguridad adicionales para mejorar la seguridad de acceso a la red podrían ser el filtrado de acceso por direcciones MAC, o el cambio periódico de las claves de acceso. De todos modos, si atendemos al contexto en el que se sitúa el núcleo del sistema domótico, este está conectado a una red doméstica, en la que los usuarios que conocen la clave de acceso no entrañan ningún peligro.

### 7.3.2 Ataque MITM (Hombre en el medio) a la aplicación Domote

La aplicación Domote se encuentra instalada en el dispositivo móvil del usuario, y este puede conectarse a redes wifi no seguras en las cuales el administrador puede monitorizar el tráfico que está pasando a través de ella, o puede ser víctima del mismo ataque de hombre en el medio con la misma estructura que en el apartado anterior. En cualquier caso, el objetivo de ambos ataques es el mismo, obtener el tráfico intercambiado entre el dispositivo y la red.

Para analizar cuales serian las consecuencias de este ataque, por simplicidad en la explicación se volverá a realizar la misma estructura de ataque que en el apartado anterior[37], en este caso para observar que clase de información el atacante puede obtener a partir del tráfico generado por Domote (Ver imagen 7.3.2-1).

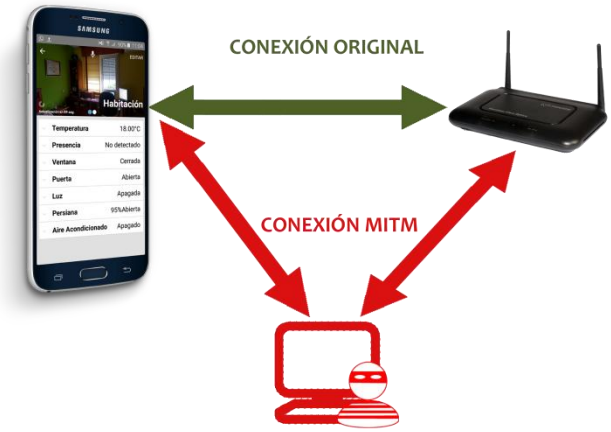



Imagen 7.3.2-1 Ataque MITM Domote

De nuevo, se vuelve a realizar el mismo proceso de configuración del ataque con la herramienta Ettercap, esta vez seleccionando como victima la dirección ip del dispositivo móvil conectado a la red, en este caso 192.168.1.33, quedando la lista de víctimas de Ettercap como se muestra en la imagen 7.3.2-2.

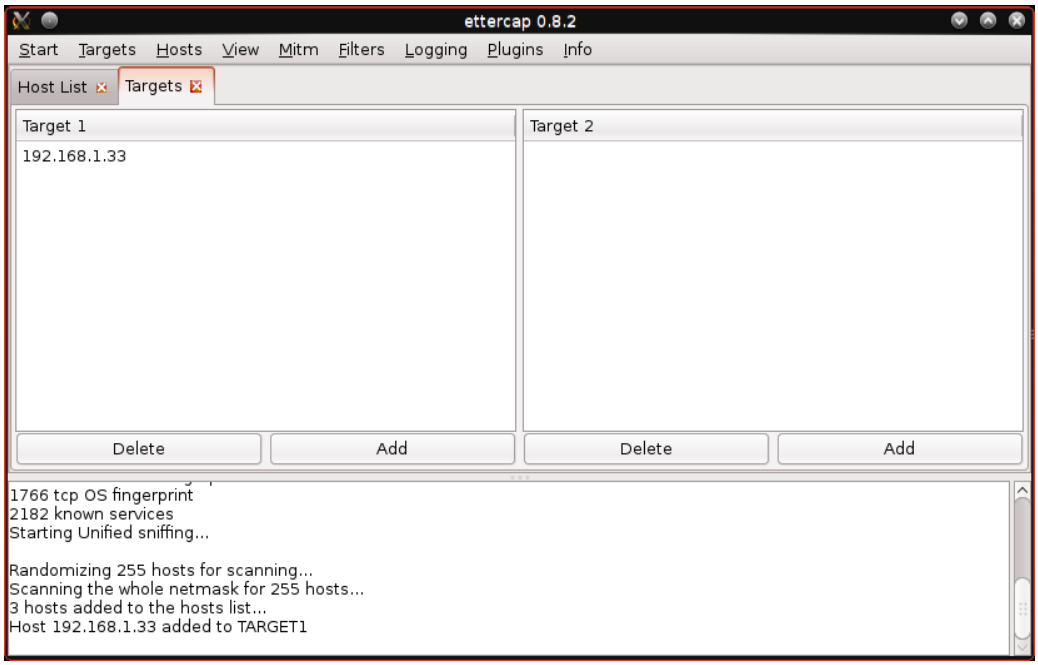



Imagen 7.3.2-2 Ettercap, victima Domote seleccionada

[37] Ver apartado ATAQUE MITM AL NÚCLEO DEL SISTEMA

Una vez lanzado el ataque, el atacante se encuentra en medio de la comunicación entre el dispositivo móvil y el router, solo queda analizar el tráfico que se está capturando mediante el analizador de protocolos Wireshark. De nuevo, se selecciona la interfaz de red que se está utilizando para realizar el ataque, y se realiza un filtrado de solo los paquetes http, obteniéndose unos resultados como los que se muestran en la imagen 7.3.2-3, en los cuales se observa todos los intercambios de mensajes entre Domote y el servidor Thingspeak.

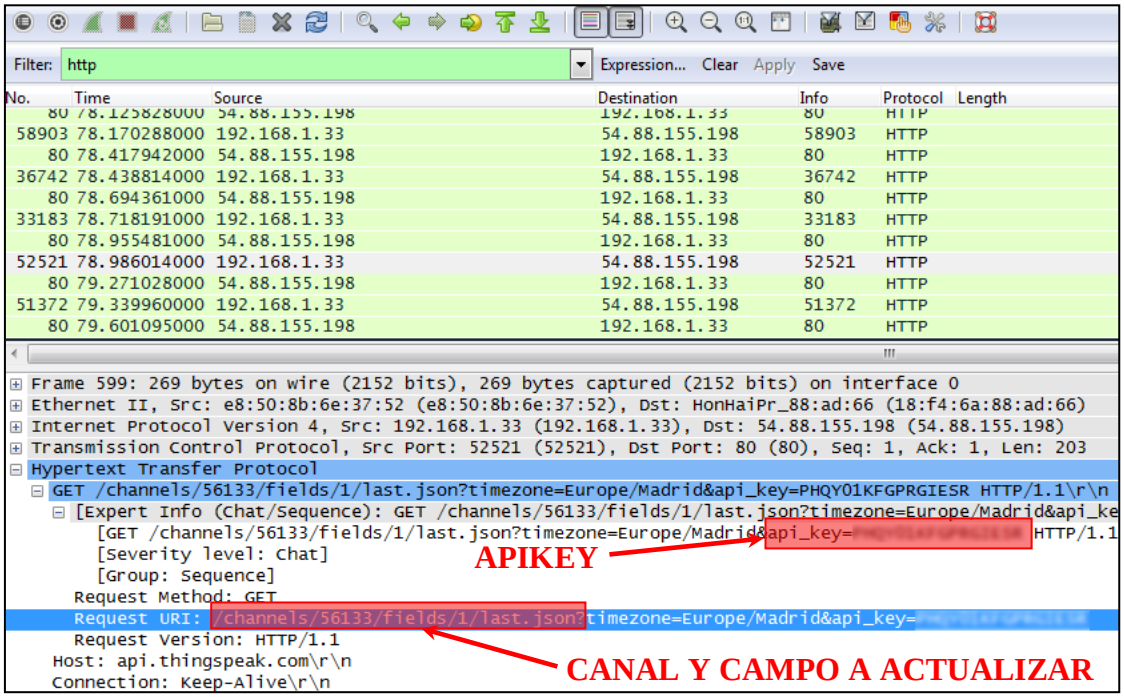



**Imagen 7.3.2-3 Wireshark, Análisis MITM a núcleo.**

Se puede comprobar el intercambio de mensajes para la actualización del valor de los campos dentro de la aplicación, los cuales comienzan con una petición desde la propia aplicación, esta petición contiene el apikey de lectura del canal al que corresponde dicho campo, y que como vemos, se envía en claro. Por otro lado, se observa las posteriores respuestas del servidor, con el valor actual de los campos solicitados.

Por otro lado, no simplemente se puede observar el intercambio de mensajes para la actualización de campos, sino que también quedan capturadas las peticiones que se realizan cuando se ejecutan los botones definidos por el usuario en la aplicación[38].

Esta vulnerabilidad es importante de solucionar, pues el dispositivo móvil puede conectarse a redes inseguras, y el atacante no solo puede conocer el valor actual de los campos en el momento de la captura, sino que dispone del apikey para obtener el valor de dichos campos en cualquier momento. Además con la captura de las peticiones de los botones, este puede replicarlos cuando lo desee. Acciones como apertura de persianas o encendido y apagado de luces puede ser realizadas por el atacante una vez realiza la captura de dichos mensajes.

[38] Ver apartado "BOTONES"

**Soluciones:**

La solución a los problemas de seguridad en la comunicación entre la aplicación y el servidor es sencilla, ya que como se ha mencionado en apartados anteriores[39], Thingspeak permite el establecimiento de conexiones bajo la capa de seguridad SSL. Para establecer este tipo de conexiones, solo se debe de definir en el código de Domote explícitamente que las comunicaciones http deben añadir esta capa de seguridad, esto se hace definiendo tal conexión como https, como se muestra en la imagen 7.3.2-4

```
public static JSONObject obtenerUltimoFeed(TSField field,Context c){
    HttpResponse salida;
    if(field.getOculto()) {
        salida = Tools.hacerGet("https://api.thingspeak.com/channels/" +
        field.getChannel() + "/fields/"+Integer.toString(field.getNum())+
        "/last.json?timezone=Europe/Madrid&api_key="+Tools.decrypt(field.getApiKey(), c), null);
    }else{
        salida= Tools.hacerGet("https://api.thingspeak.com/channels/"
        + field.getChannel() + "/fields/"+Integer.toString(field.getNum())+
        "/last.json?timezone=Europe/Madrid", null);
    }
```

Imagen 7.3.2-4 Fragmento código actualización seguro

De esta forma tan sencilla, se solucionan todos los problemas de seguridad en las comunicaciones entre Domote y el servidor Thingspeak.
Llegados a este punto queda por analizar el problema de los botones. Los botones, son peticiones http que se realizan directamente al núcleo del sistema domótico, el cual las recibe y procesa para comprobar si contienen alguna acción a realizar en el sistema. Como hemos dicho anteriormente[40], el núcleo del sistema no puede implementar la capa de seguridad SSL, pero al mismo tiempo surge la necesidad de asegurar las conexiones realizadas por Domote, ya que estas pueden establecerse en redes no seguras.
Para solucionar este problema, Thingspeak provee la herramienta ThingHTTP, que será explicada en el siguiente apartado.

## *7.4 ThingHTTP*

Se trata de una herramienta que ofrece Thingspeak dentro de su apartado Apps. ThingHTTP permite la programación de peticiones HTTP que serán lanzadas cuando Thingspeak reciba a su vez una petición con un apikey determinado. La programación de estas peticiones es muy flexible, y permite la definición de múltiples campos tales como tipo de petición (GET, POST…), dirección a la que se realiza, cabeceras o cuerpo de la petición, así como la definición del mensaje de respuesta que devolverá a la petición de origen (Ver imagen 7.4-1).

Con esta solución, se establece un ente intermedio en la comunicación entre la aplicación Domote y el núcleo del sistema domótico, asegurando la comunicación en la

[39] Ver apartado SERVICIOS DE SEGURIDAD THINGSPEAK
[40] Ver apartado "CONEXIONES SSL"

parte más expuesta a ataques, y adaptándonos a las limitaciones de procesamiento del núcleo del sistema.

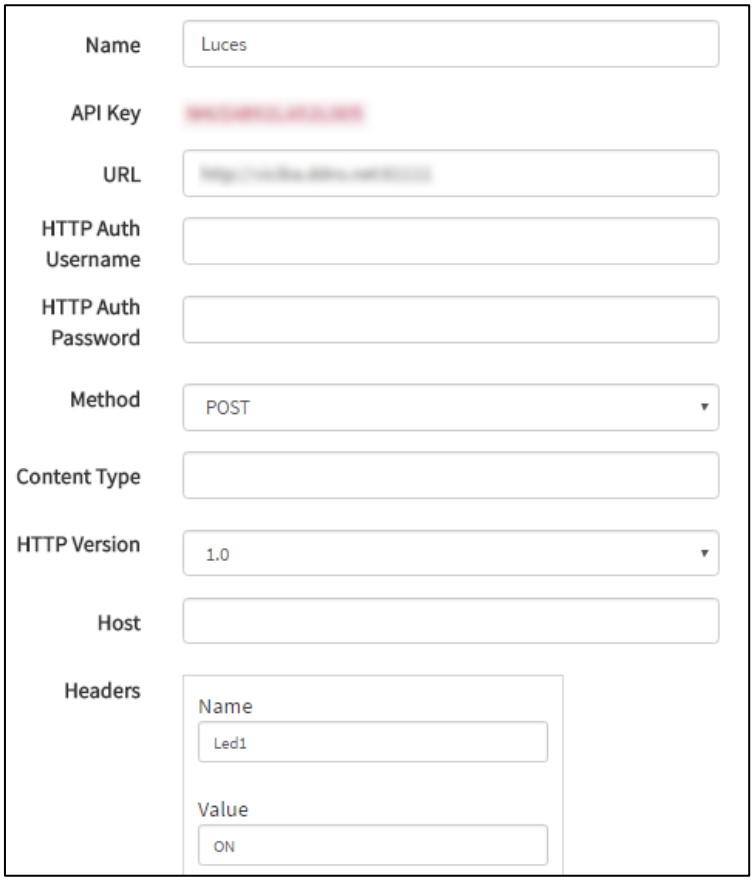



Imagen 7.4-1 Edición petición ThingHTTP

En la imagen 7.4-2, se puede observar la estructura y tipo de comunicación que finalmente se establece en el sistema domótico.

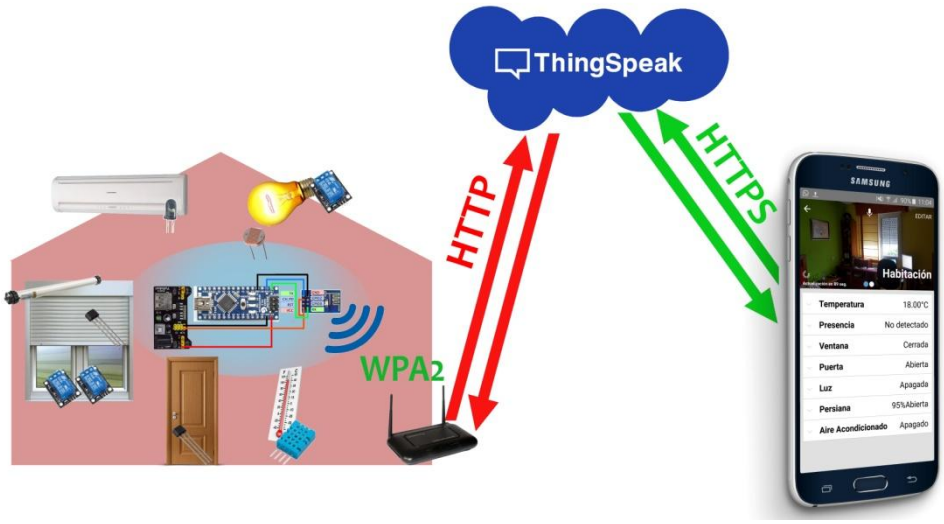



Imagen 7.4-2 Estructura y comunicaciones

# CONCLUSIONES Y FUTUROS TRABAJOS

En primer lugar, comenzaremos repasando los objetivos expuestos al comienzo de este proyecto, para verificar que estos han sido cubiertos.
Por un lado, ha diseñado el sistema domótico, tanto a nivel hardware con el núcleo del sistema y los diferentes sensores y actuadores que son capaces de tomar y modificar todos los parámetros que se pretendían controlar. Como a nivel de software, programando todo el sistema para que este para que este fuese capaz de mantener actualizada la plataforma Thinkspeak con el valor actual de los parámetros a controlar.
De esta forma, se ha conseguido que estos datos actualizados, puedan ser consultados por los dispositivos que requieran dicha información.
Es en este punto, donde reside el segundo gran objetivo cumplido de este proyecto. Se ha podido desarrollar una aplicación genérica para esta plataforma (Thinkspeak), la cual podrá ser empleada por todos sus usuarios gracias a la gran flexibilidad de la que se le ha dotado.
Esta aplicación, que ha sido denominada Domote, cumple todos los requisitos marcados al comienzo del proyecto, siendo capaz de obtener los valores de la plataforma, mostrarlos de una forma clara, gracias al establecimiento de unidades y gráficas.
También se ha ido un paso más allá de la simple consulta de datos, y se ha dotado a Domote de la posibilidad del establecimiento de Alarmas y eventos, de forma que pueda avisar al usuario, si ciertos valores cumplen o no una condición previa.
Domote también cuenta con la posibilidad de crear diferentes tipos de botones, los cuales podrán modificar mediante peticiones HTTP los actuadores de un sistema domótico. Por último, se le ha dotado de un asistente de voz, que consigue diferenciar la aplicación en el mercado, y facilita su uso al usuario.

Con lo visto anteriormente, podemos concluir que este proyecto ha cumplido con todos los objetivos que se marcaba al comienzo del mismo.

Con el trabajo desarrollado en este proyecto, hemos demostrado que el uso de esta tecnología resulta muy interesante para múltiples aplicaciones como puedan ser control y ayuda de personas dependientes, ahorro energético o a nivel industrial como puede ser el control constante de la temperatura en cámaras frigoríficas o seguridad.

Como futuros trabajos, se observa que el mayor punto de complejidad que se encuentra en este proyecto está en el establecimiento del sistema electrónico, con la programación del microcontrolador y montaje de todos los sensores. Sería interesante como futuro trabajo, la creación de una gama de dispositivos de fácil montaje (Como llaves de luz, sensores térmicos y de presencia, o enchufes) que se comuniquen de una forma inalámbrica y fácilmente emparejables con un dispositivo central que sea el encargado de subir sus parámetros al servidor. Posteriormente con la aplicación Domote ya desarrollada, el usuario solo tendría que definir a que sala asigna estos nuevos dispositivos. De este modo, se eliminarían las barreas existentes actualmente entre el sistema completo aquí expuesto y el publico general.

Otro futuro trabajo que se apunta, es la investigación de las múltiples posibilidades en el análisis de datos que ofrece Thingspeak, gracias a su herramienta de análisis matemático Matlab. A través de esta, se puede desarrollar métodos para manejar la información obtenida por el sistema, aportando de cierta inteligencia artificial al sistema.

Por último, otra vía de investigación posible para futuros trabajos, sería el desarrollo de un sistema domótico similar al aquí desarrollado, pero con un microprocesador más potente, como pueda ser una Raspberry Pi. De esta forma se puede dotar de mucha más inteligencia al sistema, pudiendo actuar el mismo como servidor propio. Y siendo a posible a la vez que necesario, el establecimiento de mecanismos de seguridad estrictos.

# REFERENCIAS


[1] Thinkspeak. (2016). Thinkspeak. 22/01/2016, de Thinkspeak Sitio web: https://thingspeak.com

[2] Temboo. (2016). Temboo. 22/01/2016, de Temboo Sitio web: https://www.temboo.com

[3] TempoIQ. (2016). TempoIQ. 22/01/2016, de TempoIQ Sitio web: https://www.tempoiq.com

[4] Xively. (2016). Xively. 22/01/2016, de Xively Sitio web: https://xively.com

[5] Mnubo. (2016). Mnubo. 22/01/2016, de Mnubo Sitio web: http://mnubo.com

[6] Beebotte. (2014). Beebotte. 16/02/2016, de Beebotte Sitio web: https://beebotte.com/

[7] Arduino. (2016). Arduino Nano. 22/01/2016, de Arduino Sitio web: https://www.arduino.cc/en/Main/ArduinoBoardNano

[8] Espressif Systems IOT Team. (2015). ESP8266EX Datasheet. 22/01/2016, de Adafruit Sitio web: https://www.adafruit.com/images/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf

[9] Itead Studio. (2015). ESP8266 Serial WIFI Module. 22/01/2016, de Itead Studio Sitio web: http://wiki.iteadstudio.com/ESP8266_Serial_WIFI_Module

[10] D-Robotics UK. (2010). DHT11 Humidity & Temperature Sensor. 22/01/2016, de Micropik Sitio web: http://www.micropik.com/PDF/dht11.pdf

[11] Arduino. (2014). DHT11 Class for Arduino. 22/01/2016, de Arduino Sitio web: http://playground.arduino.cc/Main/DHT11Lib

[12] Parallaz Inc. (2007). PIR Sensor. 22/01/2016, de Ladyada Sitio web: http://www.ladyada.net/media/sensors/PIRSensor-V1.2.pdf

[13] Melexis. (2006). US1881 Hall Latch – High Sensitivity. 22/01/2016, de Sparkfun Sitio web: https://www.sparkfun.com/datasheets/Components/General/Hall-US1881EUA.pdf

[14] RS Components. (1997). Light dependent resistors Datasheet. 22/01/2016, de bilim ve teknik Sitio web: http://www.biltek.tubitak.gov.tr/gelisim/elektronik/dosyalar/40/LDR_NSL19_M51.pdf

[15] Songle Relay. (2008). SRD. 22/01/2016, de GHI Electronics Sitio web: https://www.ghielectronics.com/downloads/man/20084141716341001RelayX1.pdf

[16] Everlight. (2005). Technical Data Sheet 5mm Infrared LED. 22/01/2016, de Adafruit Sitio web: https://www.adafruit.com/datasheets/IR333_A_datasheet.pdf

[17] Itead Studio. (2015). WeeESP8266. 22/01/2016, de Github Sitio web: https://github.com/itead/ITEADLIB_Arduino_WeeESP8266

[18] r45635. (2016). HVAC IR Control. 22/01/2016, de Github Sitio web: https://github.com/r45635/HVAC-IR-Control

[19] Jarolift. (2014). Elektromechanischer Rohrmotor Baureihe SL 35 / SL 45 / SL 59. 22/01/2016, de Jarolift Sitio web: http://anleitungen.jalousiescout.de/Jarolift_Rohrmotor_SL_NHK.pdf

[20] Thinkspeak. (2016). Documentation. 22/01/2016, de Thinkspeak Sitio web: https://thingspeak.com/docs

[21] Google Inc. (2016). Android Debelopers Reference. 22/01/16, de Google Inc. Sitio web: http://developer.android.com/intl/es/reference

[22] Google Inc. (2016). Google Play. 22/01/2016, de Google Inc. Sitio web: https://play.google.com/store/apps

[23] Google Inc. (2016). Material Design. 22/01/2016, de Google Inc. Sitio web: https://www.google.com/design/spec/material-design

[24] Online Generator. (2016). Preloaders. 22/01/2016, de OnlineGenerator.net Sitio web: http://preloaders.net/

[25] Stackoverflow. (2016). Stackoverflow. 22/01/2016, de Stackoverflow Sitio web: http://stackoverflow.com/

[26] Api.ai. (2016). Api.ai. 22/01/2016, de Apia.ai Sitio web: https://api.ai/

[27] eBay Inc. (2015). Ebay. 23/10/2015, de eBay Inc Sitio web: http://www.ebay.es/

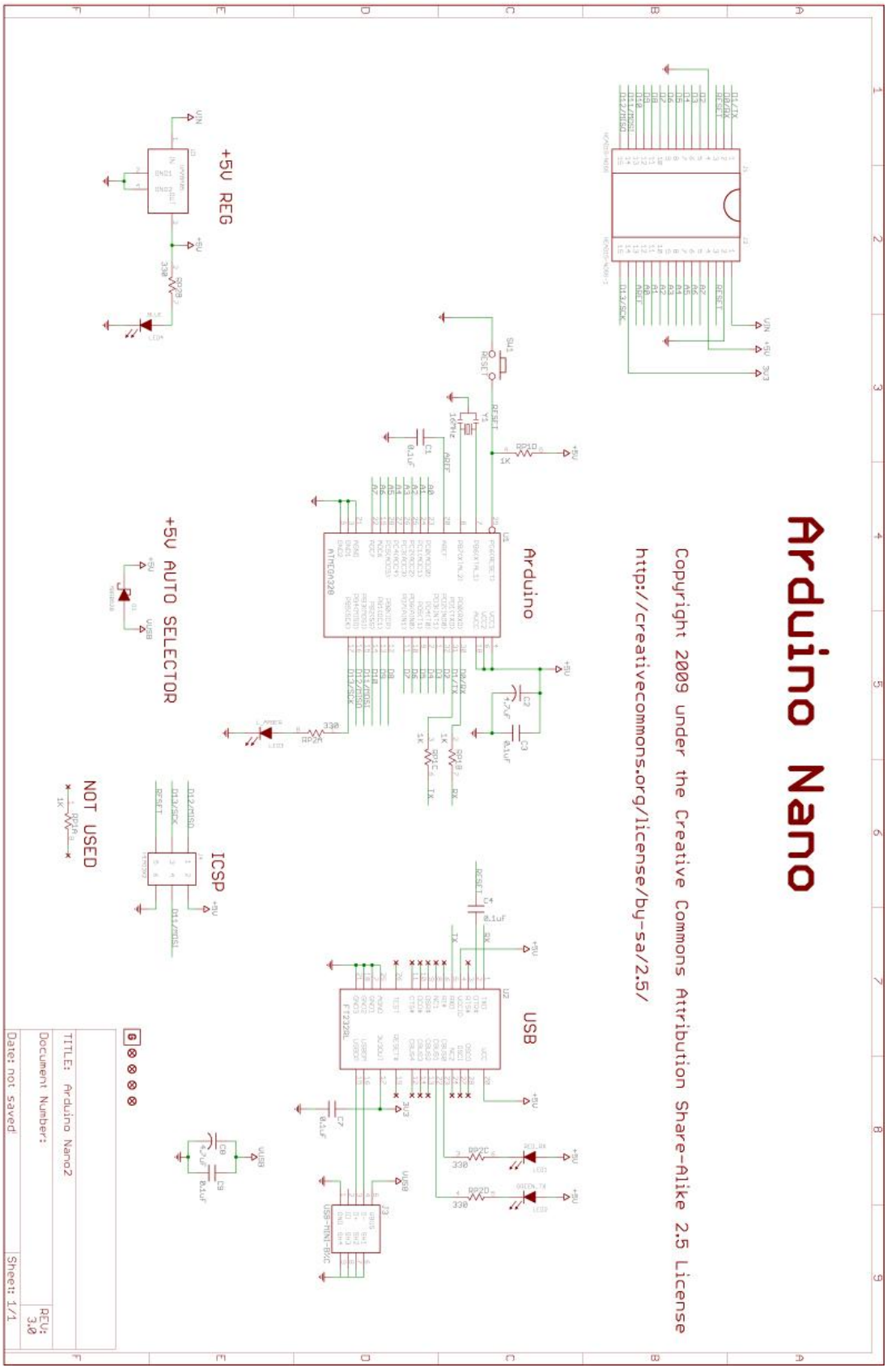
[28] SeguridadWireless.net. (2015). WifiSlax. 10/05/2016, de SeguridadWireless.net Sitio web: www.wifislax.

[29] Ettercap Project. (2015). Ettercap. 10/05/2016, de Ettercap Project Sitio web: http://ettercap.github.io/ettercap

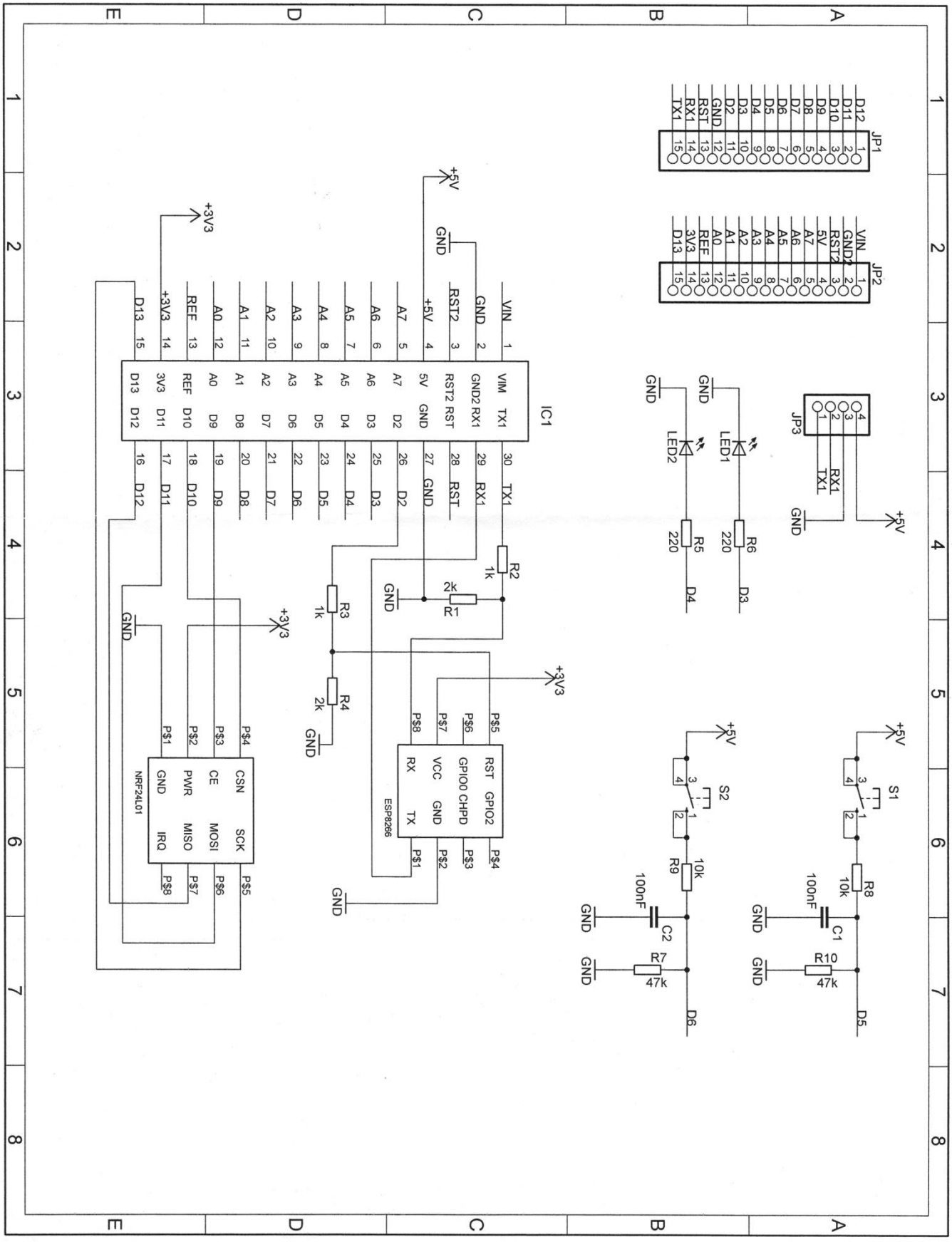
[30] Wireshark Foundation. (2016). Wireshark. 10/05/2016, de Wireshark Foundation Sitio web: www.wireshark.org

# ANEXO

## 1. *Esquemático Arduino Nano*



## 2. Esquemático de Sistema de Prototipado Departamento



# 3. Código Sistema Domótico

```
#include <ESP8266.h>
#include <MemoryFree.h>
#include "IRremote2.h"
#include "SoftwareSerial.h"
#include "DHT.h"

//------------------CONSTANTES
#define SSID        "MOVISTAR_4444F"
#define PASSWORD    "miClave"
#define LEDCONFIGURACION 3
#define LEDTCP 4
#define CH_PD 2
#define TX_MON 7
#define RX_MON 8
#define BUFFER 400
#define LIGHTPIN A0
#define DHTTYPE DHT11
#define DHTPIN 9
#define PRESENCIA 10
#define RELE 11
#define VENTANA 12
#define PUERTA A1
#define INTERRUPTORLUZ A2
#define MOTORCONTROL A4
#define MOTORDIRECCION A3
#define BOTONSUBIR 6
#define BOTONBAJAR 5
#define tiempoEntrarAuto 1500
#define tiempoTotalPersianaSubida 25000
#define tiempoAperturaOffset 6500
#define  HVAC_MITSUBISHI_SKETCH
#define intervaloAnalogicos 120000
#define timeoutEntreIntentosConexionWifi 60000
#define tiempoMinimoEntreEnvios 15000
#define tiempoEsperaConexiones 100
#define tiempoEsperaRecepcionMensaje 1000
#define tiempoPresenciaLuz 15000
#define tiempoFiltroConexiones 2000
#define tiempoEsperaRespuesta 2000
#define thingSpeakAddress "api.thingspeak.com"
#define writeAPIKeyCH1 "J6E1P5********"
#define writeAPIKeyCH2 "13IHJF********"
#define tweetThinkApiKey "05EY**********"

//Defino objetos
SoftwareSerial mon(RX_MON,TX_MON);// Establecemos conexion serie software para el monitor
ESP8266 wifi(Serial);//definimos el objeto wifi con comunicacion por el puerto serie
DHT dht(DHTPIN, DHTTYPE);//inicializamos dht
IRsend irsend;//objeto para envio de señales IR

//Defino variables
uint8_t buffeer[BUFFER] = {0};
String cero="0";
String uno="1";
int numErroresContinuos = 0;//variable que nos cuenta el numero de errores de conexion continuos
long ultimoEnvioAnalogico = 0; //timestamp de envio analogico
long lastConnectionReceived = 0;//timestamp empleada para limitar por tiempo las conexiones a
nuestro servidor TCP
long ultimoEnvioTS=0;//timestamp con la ultima actualizacion de un campo
long lastPresencia=0;//timestamp empleada en sistema luz-presencia
//PARAMETROS PERSIANA
boolean moviendoPersiana=false;//muestra si la persiana se encuentra en movimiento
long tpresion;
long tautomatico;
```

```
boolean modoAuto=false;
long tpersiana=0;//tiempo relativo de movimiento de persiana, con el cual se calculara su
posicion actual
long tmovimiento;//timestamp empleado para calcular movimientos
boolean bajada=false;//estado del rele de direccion
//PARAMETROS DE ESTADO
boolean estadoLuz;
boolean estadoPuerta;
boolean estadoVentana;
boolean luces;
boolean presencia;
boolean estadoRele=false;
boolean acEncendido=false;
boolean sistemaLuzPresencia=false;
int acTemperatura=23;


void setup() {
  //Configuracion de hardware y comunicacion
  establecerPins();
  delay(1000);
  mon.begin(9600);//Iniciamos monitor
  Serial.begin(9600);//Iniciamos serie con modulo wifi
  dht.begin();//Iniciamos temp/humedad
  //Comenzamos la configuracion inicial
  mon.print(F("----------COMIENZO DE CONFIGURACION----------\r\n"));
  digitalWrite(LEDCONFIGURACION, HIGH);//led1 encendido
  wifi.setEcho(0);//quitamos echo
  wifi.enableMUX();//activamos conexiones multiples
  wifi.startTCPServer(80);//iniciamos servidor TCP
  mon.println(wifi.getVersion().c_str());//Mostramos version de firmware
  conectarARedWifi();//conectamos a red
  //Envio de tweet como log
  String tweetInicio=F("Inicio del sistema, posible fallo de corriente");
  enviarTweet(&tweetInicio);
  //bajar persiana para inicializacion
  mon.print(F("Bajando persiana....."));
  delay(16500);
  mon.println(F("OK"));
  digitalWrite(MOTORCONTROL, LOW);//detenemos motor
  digitalWrite(MOTORDIRECCION, LOW);//direccion vuelve a estado normal

  //Preparamos el sistema que actualice los valores del servidor
  inicializarCH1();//inicializa las variables del ch1
  inicializarCH2();//inicializa las variables del ch2
  //Fin de configuracion
  mon.println(F("-------------FIN DE CONFIGURACION-------------\n\n"));
  digitalWrite(LEDCONFIGURACION, LOW);//Apagamos el led de configuracion
}


void loop() {
  if(!moviendoPersiana){//si la persiana no se esta moviendo
    if(millis()-ultimoEnvioAnalogico>intervaloAnalogicos){//Si ha pasado el intervalo de envio
        enviarValoresAnalogicos();
    }
    comprobarPeticionesServidor();//compruebo si ha llegado una nueva peticion al servidor TCP
    //-------------NUEVAS FUNCIONES
    comprobarPuerta();//chequeo si se ha producido un evento en puerta
    comprobarVentana();//chequeo si se ha producido un evento en ventana
    comprobarLuces();//chequeo si se ha producido un evento en luces
    comprobarPresencia();//chequeo si se ha producido un evento en presencia
  }
  conprobarLuzPresencia();//compruebo el sistema Luz-presencia
  comprobarPersiana();//compruebo si la persiana registra algun evento
}

//-----------------FUNCIONES DE ENVIO DE PARAMETROS
//Esta funcion recive los balores de los parametros, asi como su numero y el canal, y prepara el
mensaje HTTP
//para su posterior envio
int updateParametrosTS(int numerosCampos[], String* valores[],int canal,int cantidadParametros){
```

```
    String cuerpo="";//string con el cuerpo del mensaje
    for(int i=0;i<cantidadParametros;i++){
      if(i!=0){
        cuerpo=cuerpo+"&";//añadimos caracter "&" entre parametros
      }
      cuerpo=cuerpo+F("field");
      cuerpo=cuerpo+String(numerosCampos[i]);
      cuerpo=cuerpo+F("=");
      cuerpo=cuerpo+*valores[i];
    }
    cuerpo=cuerpo+F("&headers=false");//se solicita no recibir cabeceras en la respuesta
    //estas son las cabeceras del mensaje
    String mensaje=F("POST /update HTTP/1.1\nHost: api.thingspeak.com\nConnection: close\nX-
THINGSPEAKAPIKEY: ");
    String mensaje2=F("\nContent-Type: application/x-www-form-urlencoded\nContent-Length: ");
    String mensaje3=F("\n\n");
    String mensaje4=F("\n");
    String numero1=String(cuerpo.length());//en la cabecera debe de constar la longitud del
cuerpo
    int longitudMensaje=0;//esta variable sera la longitud total del mensaje
    //a continuación se porcede a ir añadiendo cada una de las partes del mensaje al buffer
    //y a calcular su longitud total
    anadirABuffer(longitudMensaje,&mensaje);
    longitudMensaje=longitudMensaje+mensaje.length();
    if(canal==1){//añadimos apikey
      anadirABuffer(longitudMensaje,writeAPIKeyCH1,16);//el tamano de apikey es 16
      longitudMensaje=longitudMensaje+16;
    }else{
      anadirABuffer(longitudMensaje,writeAPIKeyCH2,16);
      longitudMensaje=longitudMensaje+16;
    }
    anadirABuffer(longitudMensaje,&mensaje2);
    longitudMensaje=longitudMensaje+mensaje2.length();
    anadirABuffer(longitudMensaje,&numero1);
    longitudMensaje=longitudMensaje+numero1.length();
    anadirABuffer(longitudMensaje,&mensaje3);
    longitudMensaje=longitudMensaje+mensaje3.length();
    anadirABuffer(longitudMensaje,&cuerpo);
    longitudMensaje=longitudMensaje+cuerpo.length();
    anadirABuffer(longitudMensaje,&mensaje4);
    longitudMensaje=longitudMensaje+mensaje4.length();

    mon.print(F("Enviando datos...."));//mostramos informacion en monitor
    return enviarHTTP(longitudMensaje);//procedemos al envio del mensaje ya formado
 }

//Esta funcion se encarga del envio del mensaje HTTP a thingspeak
int enviarHTTP(int longitud)
{
  uint32_t len=-1;
  if (wifi.createTCP(1,thingSpeakAddress, 80))//establecemos conexion TCP con TS
  {
    digitalWrite(LEDTCP, HIGH);//encendemos el led TCP
    if(wifi.send(1,(const uint8_t*)buffeer,longitud)){//enviamos contenidon del buffer
      numErroresContinuos=0;//reinicimos contador de errores
      mon.print(F("ENVIOK...."));//mostramos avance en monitor
      len = wifi.recv(1,buffeer, sizeof(buffeer), tiempoEsperaRespuesta);//procedemos a recibir
la respuesta
      mon.print(F("RECIVOK...."));//mostramos avance en monitor
      wifi.releaseTCP(1);//liberamos conexion TCP
      mon.println(F("OK"));
    }else{//Si se produce un error en el envio del mensaje
      mon.println(F("Error al enviar el mensaje"));
    }
    digitalWrite(LEDTCP, LOW);//apago el led tcp
    return len;//devolvemos la longitud del mensaje recibido
  }else{//si se produce un error en el establecimiento de la conxion TCP
    numErroresContinuos++;//añadimos un error al contador
    mon.print(F("Parece que no hay conexion, fallo:"));
    mon.println(numErroresContinuos);
  }
  if(numErroresContinuos==10){//en caso de se hayan realizado 10 errores continuos
```

```
      conectarARedWifi();//se vuelve a establecer conexion wifi
      numErroresContinuos=0;//y se reinicia el contador
      String tweetInicio=F("Reestablecimiento de conexión");
      enviarTweet(&tweetInicio);//se envia un tweet como log del error
  }
  return -1;
}


//-----------------------FUNCIONES DE INICIALIZACION
void establecerPins(){
  //Modo
  pinMode(LEDCONFIGURACION, OUTPUT);//led1
  pinMode(LEDTCP, OUTPUT);//led1
  pinMode(CH_PD, OUTPUT);//chipselect
  pinMode(PRESENCIA, INPUT);//PIR SENSOR
  pinMode(RELE, OUTPUT);//RELE
  pinMode(VENTANA, INPUT);//VENTANA
  pinMode(PUERTA, INPUT);//PUERTA
  pinMode(INTERRUPTORLUZ,INPUT);//INTERRUPOR
  pinMode(MOTORCONTROL, OUTPUT);//CONTROL MOTOR PERSIANA
  pinMode(MOTORDIRECCION, OUTPUT);//DIRECCION MOTOR PERSIANA
  pinMode(BOTONSUBIR, INPUT);//BOTON PARA SUBIR PERSIANA
  pinMode(BOTONBAJAR,INPUT);//BOTON PARA BAJAR PERSIANA
  //Estado Inicial
  digitalWrite(LEDCONFIGURACION, LOW); //led2 apagado
  digitalWrite(LEDTCP, LOW);//led2 apagado
  digitalWrite(CH_PD, HIGH);//CHIP-SELECT activo
  digitalWrite(RELE, LOW);//RELE apagado
  digitalWrite(MOTORDIRECCION, HIGH);
  digitalWrite(MOTORCONTROL, HIGH);
}
//Esta función es creada para inicialiar el valor de las variables cuando el sistema es
iniciado.
//su principal misión es forzar al sistema para que envie una muestra con el estado actual
void inicializarCH1(){
  estadoPuerta=!digitalRead(PUERTA);
  estadoVentana=!digitalRead(VENTANA);
  presencia=!digitalRead(PRESENCIA);

luces=!(((digitalRead(INTERRUPTORLUZ)==HIGH)&&(!estadoRele))||((digitalRead(INTERRUPTORLUZ)==LOW
)&&(estadoRele)));//tomo valor actual
  if(luces){//si detecta que las luces estan encendidas las apaga
    cambiarLuces();
    luces=!luces;//cambio estado de luces
  }
  luces=!luces;//preparo variable para que envie al comienzo
  enviarSistemaLuzPresencia();//envio el estado del sistema luz-presencia
}
//Esta funcion inicializa el valor de las variables del canal 2 en el servidor, enviando su
estado actual
void inicializarCH2(){
  String progresoPersiana=String(getProgresoPersiana());//tomamos valor de persiana
  String temperaturaAire=String(acTemperatura);//tomamos temperatura
  int parametro[]={1,2,3};//indicamos de que campos se tratan
  String* textos[]={&progresoPersiana,&temperaturaAire,&cero};
  mon.println(F("Inicianlizando canal2"));
  updateParametrosTS(parametro,textos,2,3);//enviamos mesaje
}

//Esta funcion es empleada para crear una conxion entre el módulo y un punto de acceso
void conectarARedWifi(){
  bool conectado=false;
  long ultimoIntento = -timeoutEntreIntentosConexionWifi-1;
  while(!conectado){//entramos en un bucle que continuo hasta que se obtenga conexion
    if(millis() - ultimoIntento > timeoutEntreIntentosConexionWifi){//realizamos intentos de
conexion cada cierto tiempo
      ultimoIntento=millis();
      if (wifi.joinAP(SSID, PASSWORD)) {//llamamos a la funcion para conectar a un punto punto
de acceso
        mon.print(F("Conectado a punto de acceso!\r\n"));//si se realiza con exito monstramos
informacion en monitor
```

```
        mon.print(F("IP:"));
        mon.println( wifi.getLocalIP().c_str());
        conectado=true;
      }else{// en caso contrario mostramos error
        mon.print(F("Error al conectar con el punto de acceso :(\r\n"));
      }
    }
  }
}
//Esta funcion prepara el mensaje HTTP para el envio de un tweet con el texto pasado por
parametro
int enviarTweet(String* contenido){
    //Cabeceras del mensaje
    String       mensaje=F("POST       /apps/thingtweet/1/statuses/update       HTTP/1.1\nHost:
api.thingspeak.com\nConnection: close\nContent-Type: application/x-www-form-urlencoded\nContent-
Length: ");
    String api=F("api_key=");
    String stat=F("&status=");
    String tsData=api+tweetThinkApiKey+stat;
    String numero=String(tsData.length()+contenido->length());
    String fin=F("\n\n");
    String mensaje4=F("\n");
    int longitudMensaje=0;//Esta será la longitud total del mensaje
    //A continuación se va añadiendo al buffer el mensaje completo y se va calculando su
longitud
    anadirABuffer(longitudMensaje,&mensaje);
    longitudMensaje=longitudMensaje+mensaje.length();
    anadirABuffer(longitudMensaje,&numero);
    longitudMensaje=longitudMensaje+numero.length();
    anadirABuffer(longitudMensaje,&fin);
    longitudMensaje=longitudMensaje+fin.length();
    anadirABuffer(longitudMensaje,&tsData);
    longitudMensaje=longitudMensaje+tsData.length();
    anadirABuffer(longitudMensaje,contenido);
    longitudMensaje=longitudMensaje+contenido->length();
    anadirABuffer(longitudMensaje,&mensaje4);
    longitudMensaje=longitudMensaje+mensaje4.length();

    mon.print(F("Enviando tweet...."));//se muestra estado en monitor
    return enviarHTTP(longitudMensaje);//se envia el mensaje HTTP ya formado
}


//----------------------FUNCIONES DE SERVIDOR
//Esta funcion comprueba si se ha producido la entrada de algun cliente en nuestro servidor TCP
del modulo
//si esto es así, se atiende dicha entrada
void comprobarPeticionesServidor(){
  uint32_t len;//longitud de mensaje recibido
  uint8_t mux_id;//canal por el que se recibe
  int8_t resultado=-1;//resultado final de recepcion
  //Nos ponemos a la escucha durante un corto periodo de tiempo para ver si ha llegado algun
cliente
  len = wifi.recv(&mux_id, buffeer, sizeof(buffeer), tiempoEsperaConexiones);
  if (len > 0) {//si el cliente ha llegado, recibiremos la cabecera de su mensaje
    len   =   wifi.recv(&mux_id,   buffeer,   sizeof(buffeer),   tiempoEsperaRecepcionMensaje);
//procedemos a recibir el cuerpo del mensaje
    if(len>0){//si la del cuerpo del mensaje recibido es mayor a 0
      mon.print(F("LLega peticion canal:"));//mostramos estado en monitor
      mon.print(mux_id);
      mon.print(F("-------"));
      if(millis()  -  lastConnectionReceived  >  (long)tiempoFiltroConexiones){//filtramos  por
limitacion de tiempo entre conexiones
        mon.print(F("---Tiempo-OK,"));
        lastConnectionReceived = millis();//nuevo timestamp del filtro
        String comando=getString(len);//llamamos a la funcion getString que nos devolverá el
cuerpo del mensaje en un string
        resultado=ejecutarComandos(&comando);//posteriormente  enviamos  dicho  cuerpo  al  la
funcion ejecutarComandos
                                             //si se ha recibido un comando correcto, nos será
devuleto un numero mayor que 0
        if(resultado>0){//mostramos evolucion en monitor
```

```
          mon.println(F("-EXITO"));
        }else{
          mon.println(F("-ERROR"));
        }
      }else{//en caso de que no pase la limitacion temporal, es bloqueado y mostrado en monitor
        mon.println(F("Peticion Bloqueada por Tiempo"));
      }
      wifi.releaseTCP(mux_id);//la conexion tcp es cerrada, liberando nuestro servidor
      //tras liberar el servidor, en algunos casos es necesario enviar a thingspeak un cambio de
estado
      if(resultado==2){
        enviarSistemaLuzPresencia();//se ha modificado el estado del sistema Luz-Presencia
      }else if(resultado==3){//se ha encendido o apagado el aire acondicionado
        enviarIR();//se envia señal IR
        enviaracEncendido();//se envia cambio a TS
      }else if(resultado==4){//se ha modificado la temperatura del aire
        enviarIR();//se envia señal
        enviarACTemp();//sen envia cambio a TS
      }
    }
  }
}

//Esta funcion recibe el cuerpo de los mensajes recibidos por el servidor TCP de nuestro módulo
//y determina si se trata de alguno de los comandos definidos
int ejecutarComandos(String* comando){
  mon.print(F("Comando:"));
  //Vamos comparando el comando recibido con los definidos
  //si se trata de alguno de ellos, entrará en su condicion
  if(*comando==F("Luces on-off")){//Comando para encender/apagar luces
    mon.print(F("LUCES"));
    cambiarLuces();
    return 1;
  }else if(*comando==F("Luzpresencia: ON")){//Comando para activar el sistema de Luz-presencia
    sistemaLuzPresencia=true;
    mon.print(F("LUZ PRESENCIA ON"));
    return 2;
  }else  if(*comando==F("Luzpresencia:  OFF")){//Comando  para  desactivar  el  sistema  de  Luz-
presencia
    sistemaLuzPresencia=false;
    mon.print(F("LUZ PRESENCIA OFF"));
    return 2;
  }else  if(comando->substring(0,9)==F("Persiana:")){//Comando  para  modificar  la  apertura  de
persiana ejm:"Persiana:24"
    int posicion=comando->substring(9).toInt();//obtenemos posicion solicitada
    long                tiempo=tiempoAperturaOffset+((long)posicion*((tiempoTotalPersianaSubida-
tiempoAperturaOffset)/100));//calculamos el tiempo relativo solicitado
    if(tiempo>tpersiana+100){//si el tiempor relativo es mayor  al actual, debemos de subir la
persiana
      modoAuto=true;//ponemos en modo auto
      bajada=false;//seleccionamos sentido de movimiento
      tautomatico=millis()+tiempo-tpersiana;//fijamos hasta cuando tiene que estar en movimiento
      tmovimiento=millis();//fijamos el tiempo inicial
      digitalWrite(MOTORDIRECCION, LOW);//estableciemos rele de subida
      digitalWrite(MOTORCONTROL, HIGH);//establecemos rele de alimentacion gral
    }else if(tpersiana>tiempo+100){//si el tiempor relativo es menor al actual, debemos de bajar
la persiana
      modoAuto=true;//ponemos modo auro
      bajada=true;//seleccionamos sentido de movimiento
      tautomatico=millis()+tpersiana-tiempo;//fijamos hasta cuando tiene que estar en movimiento
      tmovimiento=millis();//fijamos el tiempo inicial
      digitalWrite(MOTORDIRECCION, HIGH);//establecemos rede de bajada
      digitalWrite(MOTORCONTROL, HIGH);//establecemos rele de alimentacion gral
    }
    moviendoPersiana=true;//se marca que la persiana se encuentra en movimiento.
    mon.print(F("MOVIENDO PERSIANA"));//Se muestra el estado en el monitor
    return 1;
  }else if(*comando==F("AC on-off")){//comando para encender y apagar el aire acondicionado
    acEncendido=!acEncendido;//cambiamos estado de variable
    mon.print(F("Aire on-off"));
    return 3;
```

```
  }else  if(comando->substring(0,7)==F("ACTemp:")){//comando  para  modificar  la  temperatura  de
aire acondicionado  ejm:"ACTem:23"
    acTemperatura=comando->substring(7).toInt();//obtenemos temperatura solicitada
    mon.print(F("Aire temp"));
    return 4;
  }
  return -1;
}




//************************TOOLS************************************************
//Funcion que añade un string al buffer, a partir de una posicion determinada por el parametro
pos
void anadirABuffer(int pos,String* cadena){
  int i=0;
  for(i=0;i<cadena->length();i++){
    buffeer[pos+i]=cadena->charAt(i);
  }
}
//Funcion  que  añade  una  cadena  de  caracteres  de  una  longitud  len  al  buffer,  a  partir  de  una
posicion determinada por el parametro pos
void anadirABuffer(int pos,char* cadena, int len){
  int i=0;
  for(i=0;i<len;i++){
    buffeer[pos+i]=cadena[i];
  }
}
//esta funcion muestra la memoria ram disponible del sistema
void checkMemory(){
    mon.print(F("**********Memoria disponible="));
    mon.print(freeMemory());
    mon.println(F("**********"));
}
//Esta función pasa el contenido del buffer hasta una longitud len a un string
String getString(int len){
  int i;
  String parametro="";
  char* puntero=(char*)buffeer;
  for(i=0;i<len;i++){
      parametro=parametro+puntero[i];
  }
  return parametro;
}



//Esta funcion enciende o apaga las luces cambiando el estado del rele
void cambiarLuces(){
  if(estadoRele){
      digitalWrite(RELE, LOW);//ENCIENDO LUCES
      estadoRele=false;
    }else{
      digitalWrite(RELE, HIGH);//APAGO LUCES
      estadoRele=true;
    }
}


//----------FUNCIONES DE PERSIANA------------
//Esta es la funcion que gestiona el estado de la persiana, controlando el estado de los botones
//y el funcionamiento correcto del modo auto
void comprobarPersiana(){
  if(digitalRead(BOTONBAJAR)==HIGH){//comprobamos si boton bajar esta siendo pulsado
    if(modoAuto){//si viene en modo auto
      if(moviendoPersiana){//si  se  esta  moviendo,  quiere  decir  que  el  usuario  quiere  parar  la
persiana
        digitalWrite(MOTORCONTROL, LOW);//paramos control
        digitalWrite(MOTORDIRECCION, LOW);//paramos direccion
```

```
        actualizaPosicionPersiana();//actualizamos la posicion de la persiana
        moviendoPersiana=false;//dejamos de mover la persiana
      }
    }else{//si no viene en modo auto
      if(!moviendoPersiana){//y no se esta moviendo, el usuario lo ha pulsado para que baje
        moviendoPersiana=true;//indicamos el movimiento de la  persiana
        tpresion=millis();//creamos timestamp para el tiempo de presion del boton
        tmovimiento=millis();//creamos timestamp para el posterior calculo de la posicion de la
persiana
        bajada=true;//indicamos que se encuentra bajando
      }
      digitalWrite(MOTORDIRECCION, HIGH);//rele de direccion en modo bajar
      digitalWrite(MOTORCONTROL, HIGH);//se activa el rele de control
    }

  }else if(digitalRead(BOTONSUBIR)==HIGH){//si se esta pulsando el boton subir
    if(modoAuto){//si viene en modo auto
      if(moviendoPersiana){//y la ventana esta en movimiento, quiere decir que el usuario quiera
pararla
        digitalWrite(MOTORCONTROL, LOW);//se desactiva el rele de control
        digitalWrite(MOTORDIRECCION, LOW);//se desactiva el rele de direccion
        actualizaPosicionPersiana();//actualizamos la posicion de la persiana
        moviendoPersiana=false;//indicamos el estado parado de la persiana
      }
    }else{//si no viene en modo auto
      if(!moviendoPersiana){//y no se esta moviendo la persiana, quiere decir que el usuario
quiere que la persiana suba
        moviendoPersiana=true;//indicamos moviento de persiana
        tpresion=millis();//creamos timestamp para el tiempo de presion del boton
        tmovimiento=millis();//creamos timestamp para el posterior calculo de la posicion de la
persiana
        bajada=false;//indicamos que se encuentra subiendo
      }
      digitalWrite(MOTORDIRECCION, LOW);//indicamos direccion de subida con el rele
      digitalWrite(MOTORCONTROL, HIGH);//se activa el rele de control para alimentar la persiana
    }
  }else{//si no se esta pulsando ningun boton
    if(moviendoPersiana){//y se esta moviendo la persiana
      if(modoAuto){//si esta en modoAuto
        if(millis()>tautomatico){//se comprueba si ha pasado del tiempo automatico que tiene que
estar
          modoAuto=false;//quitamos modo auto
          moviendoPersiana=false;//paramos movimiento
          digitalWrite(MOTORCONTROL, LOW);//se quita la alimentacion
          digitalWrite(MOTORDIRECCION, LOW);//vuelve el rele de direccion a su estado normal
          actualizaPosicionPersiana();//actualizamos la posicion de la persiana
        }
      }else{//si no viene en modo automatico, pero la parsiana se esta moviendo, significa que
el usuario
            //ha dejado de pulsar uno de los botones
        if(millis()-tpresion>tiempoEntrarAuto){//se  comprueba  si  el  tiempo  de  presion  en
suficiente para entrar en en modo auto
          if(bajada){//si la persiana esta bajando
            tautomatico=millis()+tpersiana;//calculamos  cuando  la  persiana  estará  bajada
completamente
          }else{//si esta subiendo
            tautomatico=millis()+(tiempoTotalPersianaSubida-tpersiana);//calculamos  cuando  la
persiana ha subido completamente
          }
          modoAuto=true;//establecemos el modo auto como activo
        }else{//si el tiempo de pulsacion no ha sido suficiente para entrar en modo auto
          digitalWrite(MOTORCONTROL, LOW);//detenemos persiana
          digitalWrite(MOTORDIRECCION, LOW);//vuelve rele de direccion a su estado normal
          moviendoPersiana=false;//indicamos el estado parado de la persiana
          actualizaPosicionPersiana();//calculamos la nueva posicion de la persiana
        }
      }
    }else{//si la persiana no se encuentra en movimiento
      modoAuto=false;//aseguramos que el modo auto se encuentra desactivado
    }
  }
}
```

```
//Esta funcion calcula cual es la nueva posicion de la persiana en funcion del tiempo que
//ha estado en movimiento, además actualiza al servidor
void actualizaPosicionPersiana(){
  if(bajada){//si se encuentra bajando
    tpersiana=tpersiana-(millis()-tmovimiento);//se resta el tiempo de moviento a la de la
persiana
    if(tpersiana<0){//si el resultado es menor que 0
      tpersiana=0;//normalizamos el valor
    }
  }else{//si se encuentra subiendo
    tpersiana=tpersiana+(millis()-tmovimiento);//sumamos el tiempo que ha estado en movimiento
    if(tpersiana>tiempoTotalPersianaSubida){//si supera el tiempo maximo
      tpersiana=tiempoTotalPersianaSubida;//normalizmaos al tiempo maximo
    }
  }
  //preparamos variables para actualizacion del servidor
  String texto=String(getProgresoPersiana());
  int parametro[]={1};
  String* textos[]={&texto};
  //actualizamos servidor
  updateParametrosTS(parametro,textos,2,1);
}

//Esta funcion devuelve un valor de 0 a 100 con la posicion actual de la persiana
//donde 0 es completemente cerrada y 100 completamente abierta
int getProgresoPersiana(){
  int            progreso=(int)((tpersiana-tiempoAperturaOffset)*100/(tiempoTotalPersianaSubida-
tiempoAperturaOffset));//obtengo la posicion de la persiana
    if(progreso<0){//si obtenemos un progreso negativo normalizamos
      progreso=0;
    }else if(progreso>100){//si obtenemos un progreso mayor a 100 normalizamos
      progreso=100;
   }
   return progreso;
}


//----------------------FUNCIONES AIRE ACONDICIONADO
//Esta funcion se encarga de enviar la señal IR al aire acondicionado
void enviarIR(){
  if(acEncendido){//si el dispositivo esta encendido
    float t = dht.readTemperature();//tomo temperatura de la sala
    if(t<acTemperatura){//si la temperatura de la sala es menor a la objetivo
                        //enviamos señal con modo bomba de calor
      irsend.sendHvacMitsubishi(HVAC_HOT, acTemperatura, FAN_SPEED_AUTO, VANNE_AUTO, false);
    }else{//si la temperatura de la sala es mayor a la de objetivo
          //enviamos señal con modo aire acondicionado
      irsend.sendHvacMitsubishi(HVAC_COLD, acTemperatura, FAN_SPEED_AUTO, VANNE_AUTO, false);
    }
  }else{//si la variable encendido se encuentra desactivada quiere decir que se desea apagar el
aire
        //enviamos señal de apagado
    irsend.sendHvacMitsubishi(HVAC_HOT, acTemperatura, FAN_SPEED_AUTO, VANNE_AUTO, true);
  }
}


//------------------FUNCION PARAMETROS ANALOGICOS
//Esta funcion actualiza el estado de los parametros analógicos en el servidor
int enviarValoresAnalogicos(){
  if(millis()-ultimoEnvioTS>tiempoMinimoEntreEnvios){//si  se  cumple  el  tiempo  minimo  entre
envios
        ultimoEnvioAnalogico=millis();//actualizamos timestamp de envio analogico
        ultimoEnvioTS=millis();//actualizamos timestamp de envio a servidor
        mon.println(F("Valores analogicos"));
        float h = dht.readHumidity();//tomo humedad
        float t = dht.readTemperature();//tomo temperatura
        if (isnan(t) || isnan(h)) {
          mon.println(F("Error en la lectura"));
        } else {
          int l  = analogRead(LIGHTPIN);//tomo luminosidad
          //Preparo variables para actualizar el servidor
```

```
          String texto=String(l);
          String texto2=String(h);
          String texto3=String(t);
          int parametro[]={1,2,3};
          String* textos[]={&texto,&texto2,&texto3};
          return updateParametrosTS(parametro,textos,1,3);//actualizo el servidor
        }
        return -1;
   }
}

//---------------------------------------FUNCIONES CHECK
//Esta funcion comrueba si la variable puerta ha cambiado y si es asi notifico al servidor
void comprobarPuerta(){
  if(millis()-ultimoEnvioTS>tiempoMinimoEntreEnvios){//si  se  cumple  el  tiempo  minimo  entre
envios
    if(estadoPuerta!=digitalRead(PUERTA)){//compruebo si el estado de puerta ha cambiado
      estadoPuerta=digitalRead(PUERTA);//si es asi tomo nuevo parametro y procedo a enviar al
servidor
      ultimoEnvioTS=millis();//actualizamos timestamp de envio a servidor
      //Preparo variables para actualizar el sevidor
      int parametro[]={6};
      String* textos[1];
      if(estadoPuerta==HIGH){
        textos[1]=&cero;
        mon.println(F("Puerta cerrada"));
      }else{
        textos[1]=&uno;
        mon.println(F("Puerta abierta"));
      }
      updateParametrosTS(parametro,textos,1,1);//actualizo el servidor
    }
  }
}
//Esta funcion comprueba si la variable ventana ha cambiado y si es asi notifica al servidor
void comprobarVentana(){
  if(millis()-ultimoEnvioTS>tiempoMinimoEntreEnvios){//si  se  cumple  el  tiempo  minimo  entre
envios
    if(estadoVentana!=digitalRead(VENTANA)){//compruebo si el estado de ventana ha cambiado
      estadoVentana=digitalRead(VENTANA);//si es asi tomo nuevo parametro y procedo a enviar al
servidor
      ultimoEnvioTS=millis();//actualizamos timestamp de envio a servidor
      //Preparo variables para actualizar el servidor
      int parametro[]={5};
      String* textos[1];
      if(estadoVentana==HIGH){
        textos[1]={&uno};
        mon.println(F("Ventana abierta"));
      }else{
        textos[1]={&cero};
        mon.println(F("Ventana cerrada"));
      }
      updateParametrosTS(parametro,textos,1,1);//actualizo el servidor
    }
  }
}
//Esta funcion comprueba si la variable luces ha cambiado y si es asi notifica al servidor
void comprobarLuces(){
  if(millis()-ultimoEnvioTS>tiempoMinimoEntreEnvios){//si  se  cumple  el  tiempo  minimo  entre
envios

if(luces!=(!(((digitalRead(INTERRUPTORLUZ)==HIGH)&&(!estadoRele))||((digitalRead(INTERRUPTORLUZ)
==LOW)&&(estadoRele))))){//si ha cambiado de estado de la luz

luces=!(((digitalRead(INTERRUPTORLUZ)==HIGH)&&(!estadoRele))||((digitalRead(INTERRUPTORLUZ)==LOW
)&&(estadoRele)));//actualizo variable
      ultimoEnvioTS=millis();//actualizamos timestamp de envio a servidor
      //Preparo variables para actualizar el servidor
      int parametro[]={7};
      String* textos[1];
      if(luces){
        textos[1]={&uno};
```

```
        mon.println(F("Luz encendida"));
      }else{
        textos[1]={&cero};
        mon.println(F("Luz apagada"));
      }
      updateParametrosTS(parametro,textos,1,1);//actualizao el servidor
    }
  }
}
//Esta funcion comprueba si la variable presencia ha cambiado y si es asi notifica al servidor,
//además activa si es preciso el sistema de luz presencia
void comprobarPresencia(){
  if(millis()-ultimoEnvioTS>tiempoMinimoEntreEnvios){//si  se  cumple  el  tiempo  minimo  entre
envios
    if(presencia!=digitalRead(PRESENCIA)){//si ha cambiado el estado de presencia
      presencia=digitalRead(PRESENCIA);//actualizo variable
      ultimoEnvioTS=millis();//actualizamos timestamp de envio a servidor
      //Preparao variables para envio
      int parametro[]={4};
      String* textos[1];
      if(presencia==HIGH){//si ha detectado presencia
        if(sistemaLuzPresencia){//si el sistema Luz-Presencia se encuentra activo
          if(millis()-lastPresencia>tiempoPresenciaLuz){//compruebo  si  se  cumple  el  tiempo
minimo entre cambios del sistema Luz-Presencia
            if(!luces){//si las luces estan apagadas
                lastPresencia=millis();//actualizamos  timestamp  de  cambio  por  sistema  luz
presencia
                cambiarLuces();//encendemos las luces
            }
          }
        }
        textos[1]=&uno;
        mon.println(F("DETECTADO"));
      }else{
        textos[1]=&cero;
        mon.println(F("NO DETECTADO"));
      }
      updateParametrosTS(parametro,textos,1,1);//actualizamos el servidor
    }
  }
}
//Esta función actualiza el servidor notificando si el sistema Luz-Presencia se encuentra activo
o desactivo
void enviarSistemaLuzPresencia(){
  while(millis()-ultimoEnvioTS<tiempoMinimoEntreEnvios);//espero  hasta  que  se  cumple  el  tiempo
minimo entre mensajes al servidor
  ultimoEnvioTS=millis();//actualizamos timestamp de envio a servidor
  //Preparamos las variables para la actualizacion del servidor
  String* textos[1];
  if(sistemaLuzPresencia){
        mon.println(F("Activo sistema Luz-Presencia"));
        textos[0]=&uno;
  }else{
        mon.println(F("Desactivo sistema Luz-Presencia"));
        textos[0]=&cero;
  }
  int parametro[]={8};
  updateParametrosTS(parametro,textos,1,1);//enviamos al servidor
}
//funcion que actualiza el estado del servidor de la variable acEncendido
void enviaracEncendido(){
  while(millis()-ultimoEnvioTS<tiempoMinimoEntreEnvios);//esperamos  hasta  que  se  cumple  el
tiempo minimo entre mensajes al servidor
  ultimoEnvioTS=millis();//actualizamos timestamp de envio a servidor
  //Preparamos las variables para la actualizacion del servidor
  String* textos[1];
  if(acEncendido){
        mon.println(F("AC Encendido"));
        textos[0]=&uno;
  }else{
        mon.println(F("AC Apagado"));
        textos[0]=&cero;
```

```
  }
  int parametro[]={3};
  updateParametrosTS(parametro,textos,2,1);//enviamos al servidor
}
//funcuion que actualiza el estado del servidor de la variabble acTemperatura
void enviarACTemp(){
  while(millis()-ultimoEnvioTS<tiempoMinimoEntreEnvios);//esperamos  hasta  que  se  cumple  el
tiempo minimo entre mensajes al servidor
  ultimoEnvioTS=millis();//actualizamos timestamp de envio a servidor
  //Preparamos las variables para la actualizacion del servidor
  String* textos[1];
  String numero=String(acTemperatura);
  textos[0]=&numero;
  int parametro[]={2};
  mon.println(F("Temperatura cambiada"));
  updateParametrosTS(parametro,textos,2,1);//enviamos al servidor
}



void conprobarLuzPresencia(){//**************************REPASAR****************//
  if(sistemaLuzPresencia){
    if(millis()-lastPresencia>tiempoPresenciaLuz){//este timeout se ha fijado por problemas en
la emision de mensajes para cambiar parametro luces
      if((!luces)&&(presencia)){//las luces deben ser encendidas
          lastPresencia=millis();
          cambiarLuces();
      }else if((luces)&&(!presencia)){//las luces deben ser apagadas
        lastPresencia=millis();
        cambiarLuces();
      }
    }
  }
}
```

Universidad de Alcalá