|  |
| --- |
| A description...  **UNIVERSIDAD EUROPEA DE MADRID**  A description...  ESCUELA ARQUITECTURA INGENIERÍA Y DISEÑO  CICLO FORMATIVO DE GRADO SUPERIOR  DESARROLLO DE APLICACIONES MULTIPLATAFORMA |
| PROYECTO FIN DE CICLO |
| **FUTH (Futurizing Homes)** |
|  |
|  |
| **CURSO 2017-18** |

**TÍTULO**: Futh

**AUTORES**: Iván Gabriel Pajón Rodríguez

Jozet Stiven Quipuscoa Ponte

**TUTOR DEL PROYECTO**: Ernesto Ramiro Córdoba

**FECHA DE LECTURA**: ... de Junio de 2018

**CALIFICACIÓN**:

Fdo: NOMBRE APELLIDO1 APELLIDO2

Ernesto Ramiro Córdoba

Tutor/a del Proyecto

# RESUMEN

En este trabajo se va a desarrollar una aplicación para dispositivos Android que sea capaz de monitorizar una serie de sensores (tales como de temperatura, humedad, gas…) y controlar objetos cotidianos como una bombilla o un enchufe. Cada usuario que se descargue la aplicación deberá iniciar sesión con Google+ para poder empezar a usarla, lo que asegura un mayor control de uso sobre los usuarios de nuestra app. Uno de los fundamentos de este trabajo es una aplicación útil, real y escalable, refiriéndonos a escalable en el sentido de que para la realización del mismo solo se desarrollará un dispositivo, pero la app estará diseñada para permitir añadir múltiples dispositivos (como si fuera un producto que podemos comprar). De igual manera, nos centraremos en desarrollar una interfaz simple, organizada y atractiva, permitiendo a usuarios sin mucho conocimiento de tecnología y/o domótica entender cómo funciona nuestro producto y permitiéndoles usar las herramientas que ponemos a su disposición sin dificultad alguna.

A su vez, se va a desarrollar desde cero un dispositivo basado en código abierto que sea capaz de monitorizar una serie de sensores y objetos cotidianos como los antes mencionados, incluyendo el hardware necesario para todo ello. Este dispositivo se conectará Firebase, al igual que la app, y de esa manera serán capaces de comunicarse entre ellos, dando la posibilidad de controlarlo tanto desde una red local como de internet, sin tener que depender de direcciones IP y/u otros factores que suelen restringir el perfil medio del usuario que consume el producto.

También se va a desarrollar un servicio de notificaciones “inteligente” que interprete los datos que recibe del dispositivo y en base a ellos genere el aviso más oportuno para el usuario. Dicho servicio estará implementado no sólo con dispositivos móviles Android, sino que también tendrá soporte para coches con Android Auto y relojes con Android Wear.

# ABSTRACT

In this work we will develop an application for Android devices capable of monitoring some sensors (such as temperature, humidity, gas ...) and control everyday objects such as a light bulb or a plug. Each user that downloads the application must log in with Google+ in order to start using it, which ensures greater usage control over the users of our app. One of the fundamentals of this work is a useful, real and scalable application, referring us to scalable in the sense that for the realization of the same one device will be developed, but the app will be designed to allow adding multiple devices (as if it were a product that we can buy). In the same way, we will focus on developing a simple, organized and attractive interface, allowing users without much knowledge of technology and/or domotics to understand how our product works and allowing them to use the tools that we put at their disposal without any difficulty.

At the same time, we will develop a device from scratch based on open source that is capable of monitoring some sensors and everyday objects such as the ones mentioned above, including the necessary hardware for all this. This device will connect Firebase, like the app, and in that way they will be able to communicate with each other, giving the possibility to control it both from a local network and the internet, without having to depend on IP addresses and/or other factors that they usually restrict the average profile of the user who consumes the product.

It will also develop an "intelligent" notification service that interprets the data it receives from the device and, based on them, generates the most opportune notification for the user. This service will be implemented not only with Android mobile devices, but will also have support for cars with Android Auto and watches with Android Wear.

# AGRADECIMIENTOS

Aquí nos ponemos sentimentales y expresamos el agradecimiento a quien pueda haber sido significativo en el desarrollo del proyecto (parejas, familia, profesores…)



Esta obra se distribuye bajo una licencia Creative Commons.

Se permite la copia, distribución, uso y comunicación de la obra si se respetan las siguientes condiciones:

* Se debe reconocer explícitamente la autoría de la obra incluyendo esta nota y su enlace.
* La copia será literal y completa.
* No se podrá hacer uso de los derechos permitidos con fines comerciales, salvo permiso expreso de los autores.

El texto precedente no es la licencia completa sino una nota orientativa de la licencia original completa (jurídicamente válida) que puede encontrarse en: http://creativecommons.org/licenses/by-nc-nd/3.0/deed.es

# Índice

RESUMEN 3

ABSTRACT 4

AGRADECIMIENTOS 5

Índice 7

Introducción 8

Objetivos 9

Motivación 10

Antecedentes 11

Desarrollo del proyecto 12

Herramientas tecnológicas 13

Java 13

Android Studio 14

Firebase 15

Arduino 16

C++ 17

Planificación 18

Descripción del trabajo realizado 19

Documentación técnica 19

Documentación funcional 26

Proceso 27

Bibliografía y Webgrafía 30

Anexos 31

1. Apps domótica 31

A. Imperihome 31

B. Houseinhand KXN 34

C. TaHoma by Somfy 37

2. Protocolos 41

D. MQTT 41

3. Componentes 43

E. Relé 43

F. Sensor DHT22 44

G. Sensor MQ9 45

H. Microcontrolador ESP32 46

# Introducción

El propósito de este proyecto es desarrollar un dispositivo y una aplicación para móviles conjuntamente, con el ánimo de estudiar de qué manera se podría mejorar la calidad de vida de las personas mediante el uso de la tecnología. A su vez, a medida que se vaya desarrollando el proyecto se irán estudiando aplicaciones futuras y sectores distintos al personal (por ejemplo laboral, y cómo mejorar la productividad y el confort de los empleados).

Para el desarrollo del trabajo se han analizado apps ya existentes tanto en busca de puntos fuertes y posibles mejoras como para buscar un patrón de diseño y uso en común sobre el que luego desarrollar la nuestra, son las siguientes:

* ImperiHome
* Houseinhand KXN
* TaHoma by Somfy

## Objetivos

Objetivos principales:

* Realizar una Aplicación para dispositivos con sistema operativo Android
* Implementar diferentes sensores conectados a Arduino
* Gestión del control absoluto de sensores desde la Aplicación Android
* Capacidad de planificar y realizar las diferentes tareas cada miembro del grupo

Nuestro principal objetivo consiste en la realización de una aplicación Android capaz de poder controlar los diferentes sensores añadidos previamente por el usuario.

Para ello debemos poner en práctica todos los conocimientos adquiridos durante el curso de Desarrollo de Aplicaciones Multiplataforma y si hiciese falta aprender nuevas tecnologías y/o lenguajes.

Otro de nuestros objetivos a tener en cuenta, es el poder llevar a cabo una implementación de las notificaciones de nuestra aplicación para que sean compatibles con Android Auto y Android Wear, y que los usuarios que la usen, puedan tener al alcance y de múltiples formas la información de los diferentes sensores instalados en sus casas. Esto es un gran avance tecnológico y significativo para la vida de nuestros futuros usuarios, ya que nos libera de tener que llevar nuestro dispositivo móvil a todos lados para poder recibir cualquier alerta desde nuestra aplicación.

## Motivación

La elección final de este proyecto fue debido a la fuerte creencia que tenemos de que la domótica, a pesar de estar creciendo día a día, continuará creciendo en el futuro.

El desafío de poder juntar dos de las grandes tecnologías actualmente, como es una aplicación Android, muy utilizadas en el día a día de muchas personas; y el mundo de la domótica, una forma de simplificar y mejorar nuestras vidas; nos hace ponernos de acuerdo en realizar un proyecto ambicioso enfocado al mundo real, poniendo en práctica la visión de escalabilidad y globalidad que hemos ido adquiriendo durante las prácticas FCT.

Nuestro carácter autodidacta no nos ha permitido conformarnos con las herramientas y conocimientos que hemos adquirido durante el CFGS de Desarrollo de Aplicaciones Multiplataforma, queriéndolos llevar un paso más allá.

Aun sabiendo que en el mercado existen productos similares, la idea de realizar desde cero el nuestro propio nos inspira a pasar por los procesos de idealización, concepto, diseño y desarrollo.

## Antecedentes

Para el desarrollo de nuestra app hemos tenido en cuenta 3 que ya están en el Play Store, sirviendo de referentes a la hora de elegir paleta de colores, estilo de iconos, navegabilidad y organización de la nuestra.

Para el desarrollo de la parte de domótica hemos usado como referente desarrollos caseros que encontramos por internet, y trabajos realizados anteriormente por nosotros mismos. Cabe destacar que al diseñar nosotros mismos el hardware y los protocolos de comunicación de este proyecto hemos tenido que usar en gran parte nuestra imaginación y/o las herramientas que en el momento de la realización están a nuestro alcance.

A continuación exponemos el análisis de las 3 apps analizadas como antecedentes de nuestro proyecto:

**Imperihome** es una app con gran potencial, pero que lamentablemente su interfaz deja mucho que desear: mala organización, “look” de aplicación antigua y algún icono poco descriptivo. El mayor impedimento que hemos encontrado ha sido que para poder probarla nos hemos tenido que registrar, y una vez registrados, ya pudimos probar el modo demo, cosa que con las otras 2 apps no ocurría (las probamos sin registrarnos). Estamos seguros que esto penalizará bastante sobre el número de usuarios que se deciden por usar la app. ([Ver Anexo 1.A](#_Imperihome))

**Houseinhand KNX** tiene una interfaz muy limpia y organizada, con iconos descriptivos. La interfaz está construida en “modo noche”, con fondo negro y texto e iconos en blanco, lo cual cansa menos a la vista, pero quizá deberían dar la opción al usuario de elegir si desean usar la aplicación con dicho modo o no. ([Ver Anexo 1.B](#_Houseinhand_KXN))

En el caso de **Tahoma by Somfy** tenemos una app con funciones muy avanzadas y animaciones muy bien diseñadas para permitir al usuario entender qué está haciendo. ([Ver Anexo 1.C](#_TaHoma_by_Somfy))

# Desarrollo del proyecto

En el desarrollo de la aplicación para dispositivos *Android*, hemos empezado implementando la pantalla de *Login*, en la cual se podrá iniciar sesión mediante cuenta de *Google+*. Hemos pensado en esa única forma de poder iniciar sesión con nuestra aplicación debido a que creemos innecesarios otros métodos de inicio como *Facebook*, *Twiter* o *GitHub*. Al buscar e instalar nuestra aplicación mediante *Play Store*, damos por hecho que el usuario tiene cuenta de *Google+*, ya que sin ella no sería capaz de hacer descargas desde *Play Store* y por lo tanto no sería ningún inconveniente el inicio de sesión para nuestros usuarios.

Para la autenticación hemos utilizado el servicio ***Authentication*** que nos proporciona *Firebase*, ya que es fácil de implementar y más seguro.

Al hacer clic en el botón de Iniciar sesión, si se tiene una cuenta de asociada en nuestro dispositivo, nos abrirá un pop-up visualizándola y podrá seleccionarse directamente sin necesidad de escribir correo y contraseña de la cuenta.

Al iniciar sesión se mantendrá siempre la sesión iniciada aunque se cierre la aplicación, ya que es mucho más cómodo para el usuario, en lugar de tener que iniciar sesión cada vez que se abra nuestra aplicación.

Una vez iniciada la sesión hemos implementado una pantalla principal, en la cual se mostrarán **todos** los dispositivos que tienes sincronizados, un botón para añadir dispositivos, un panel lateral, que tendrá como cabecera la foto de perfil; y el correo y el nombre asociada a la cuenta. En el cuerpo del panel figurarán todos los dispositivos añadidos, para que en cualquier momento se pueda seleccionar uno, sin necesidad de tener que pasar por la pantalla principal.

Hemos decidido que cada dispositivo tenga un **ID único** con el cual el usuario pueda sincronizarlo posteriormente en la app. Para conocer la descripción de los componentes usados en el desarrollo de Arduino [Ver Anexo 3](#_Componentes).

## Herramientas tecnológicas

### Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado para tener las mínimas dependencias de implementación posibles. El propósito de este lenguaje es el de ***WORA*** (*write once, run anywhere*).

Originalmente fue desarrollado por James Gosling, de *Sun Microsystems* (que más tarde compraría *Oracle*), y publicado en 1995 como un componente fundamental de la plataforma *Java* de *Sun Microsystems*. Su sintaxis deriva en gran medida de **C** y **C++**, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de *Java* son compiladas a ***bytecode*** (clase *Java*), que puede ejecutarse en cualquier máquina virtual *Java* (***JVM***) sin importar la arquitectura de la computadora que lo ejecute.



### Android Studio

*Android Studio* es un entorno de desarrollo de **software libre** destinado a programar aplicaciones para la plataforma *Android*.

Es una herramienta bastante útil, nos permite hacer el diseño de la aplicación y a su vez tener una vista previa de los cambios que vamos realizando en el diseño. Nos proporciona una lista de *widgets* y diseños que podemos arrastrar directamente en el editor. Otra de las facilidades es dar la posibilidad de poder mostrar la visualización de la aplicación de forma horizontal o vertical (*landscape* o *portrait*).

Es de agradecer que el editor de código inteligente que integra, a diferencia de otros entornos como *Eclipse*, nos permite ir escribiendo código mientras recibimos sugerencias relacionadas al código que escribimos. Soporta lenguajes de programación como *Java*, *Kotlin* (añadido recientemente) y *C* / *C++*.

Tiene un emulador de dispositivos *Android* el cual nos brinda la posibilidad de ir ejecutando nuestra aplicación cada vez que hayamos realizado cambios y queramos comprobar su correcto funcionamiento. Es bastante cómodo, ya que no nos haría falta tener un dispositivo *Android* real conectado a nuestro portátil cada vez que queramos ejecutar nuestra Aplicación en la fase de desarrollo.

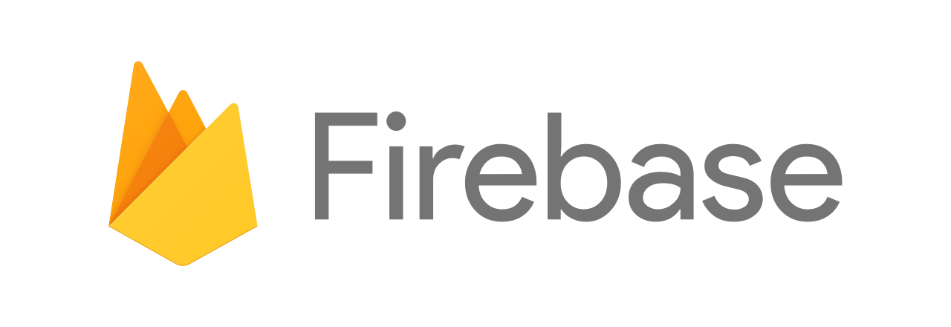


### Firebase

*Firebase* es una plataforma de desarrollo móvil en la nube. El gran atractivo de esta tecnología es que dispone de una gran variedad de productos de manera gratuita, y ofrece soporte para los más comunes *IDEs*, plataformas y sistemas operativos. En nuestro proyecto usaremos concretamente dos: ***Realtime Database*** y ***Authentication***.

Uno de ellos para almacenar los datos de los sensores y el otro para gestionar los usuarios que acceden a la aplicación.

Otra ventaja es que es **realmente escalable**, por lo que no tendríamos problemas a la hora de llevar nuestro producto a más personas, o adaptarlo en caso de que exista una gran demanda. Y en cualquier momento podríamos implementar más servicios de *Firebase* en el proyecto sin necesitar rehacerlo por completo.



### Arduino

Arduino es una compañía ***Open Source*** y ***Open Hardware***, así como un proyecto y **comunidad internacional** que diseña y manufactura multitud de dispositivos capaces de controlar objetos del mundo real mediante la programación.

Hemos elegido esta plataforma para desarrollar el dispositivo físico debido a la gran comunidad que tiene, y al gran número de placas con microcontroladores distintos que nos ofrece. Por ello, hemos elegido el ***ESP32*** ([Ver Anexo 3.H](#_Microcontrolador_ESP32)), una placa que incluye un microprocesador con **dos núcleos** y capaz de realizar tareas un tanto pesadas como puede ser la conexión a *Firebase* mientras monitoriza los sensores.

Otra de las razones por las que elegimos esta plataforma es porque pretendemos que nuestro producto sea lo más personalizable por el usuario posible, consiguiéndolo **por completo** gracias a este tipo de microcontroladores.



### C++

Cabe destacar que el entorno de desarrollo que utilizaremos con el ***ESP32*** será el de *Arduino*, que está basado en el lenguaje de programación *C++*.

A pesar de ser un lenguaje diseñado en el año *1979*, permite una programación **orientada a objetos**, lo cual es realmente útil y cómodo a la hora de trabajar con *Arduino*. Tenemos que remarcar que incluimos este lenguaje como tecnología usada, pero en realidad *Arduino* usa una versión modificada del mismo, haciéndolo más simple y comprensible por los usuarios que lo utilizan.



## Planificación

El **principal** canal de comunicación que usaremos será ***Whatsapp***, por el cual indicaremos que tareas tenemos pendientes por realizar y cuales están ya realizadas.

Otro canal de comunicaciones que usaremos con frecuencia será ***Skype***, en el cual haremos pequeñas sesiones de llamadas para detallar lo que hemos hecho hasta el momento, errores encontrados y posibles mejoras.

La comunicación con nuestro tutor de proyecto será mediante ***Slack***, en donde tendremos un canal privado que estará compuesto por los integrantes del proyecto y el tutor. Será por este canal por el que nos mantendremos informados de las entregas y reuniones por videoconferencia para informar de los avances del proyecto.



## Descripción del trabajo realizado

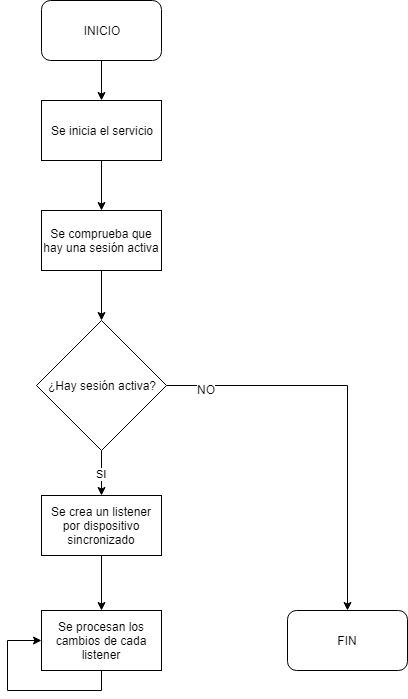
### Documentación técnica

En este apartado expondremos el trabajo que hemos realizado sobre el **modelado de datos** que aplicaremos sobre nuestro proyecto.

#### Diagramas de flujo

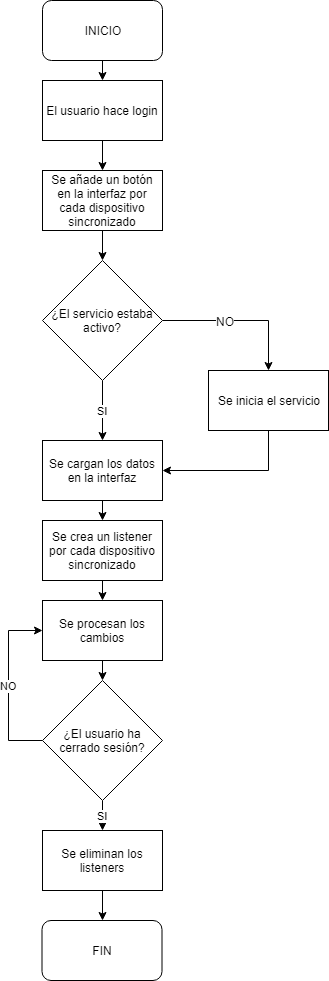
##### Flujo servicio Android

A continuación se expone el flujo con el que debería funcionar el **servicio en segundo plano** que tendrá la app para poder mostrarle notificaciones al usuario.



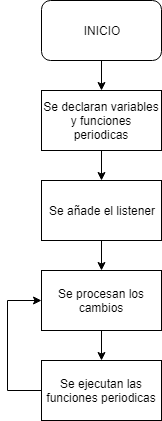
##### Flujo aplicación Android

Establecimos con el siguiente diagrama cuál debía ser el **flujo de funcionamiento** de la app en general.



##### Flujo Arduino

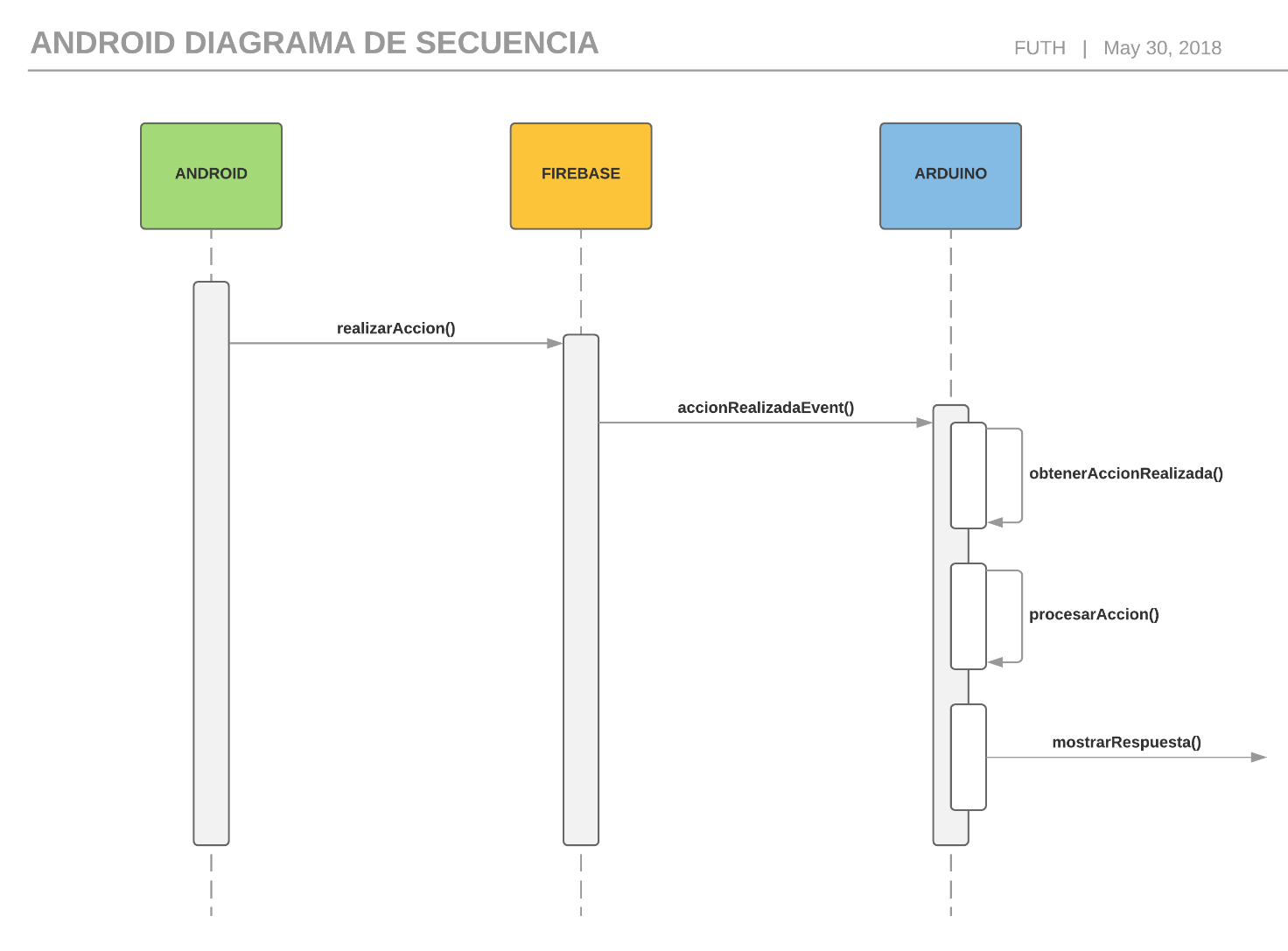
Igualmente, de la siguiente manera deberá funcionar **nuestro dispositivo**. Cabe destacar que no se indica finalización debido a que el trabajo del dispositivo **ha de ser constante e ininterrumpido**.



#### Diagramas de secuencia

##### Secuencia aplicación Android

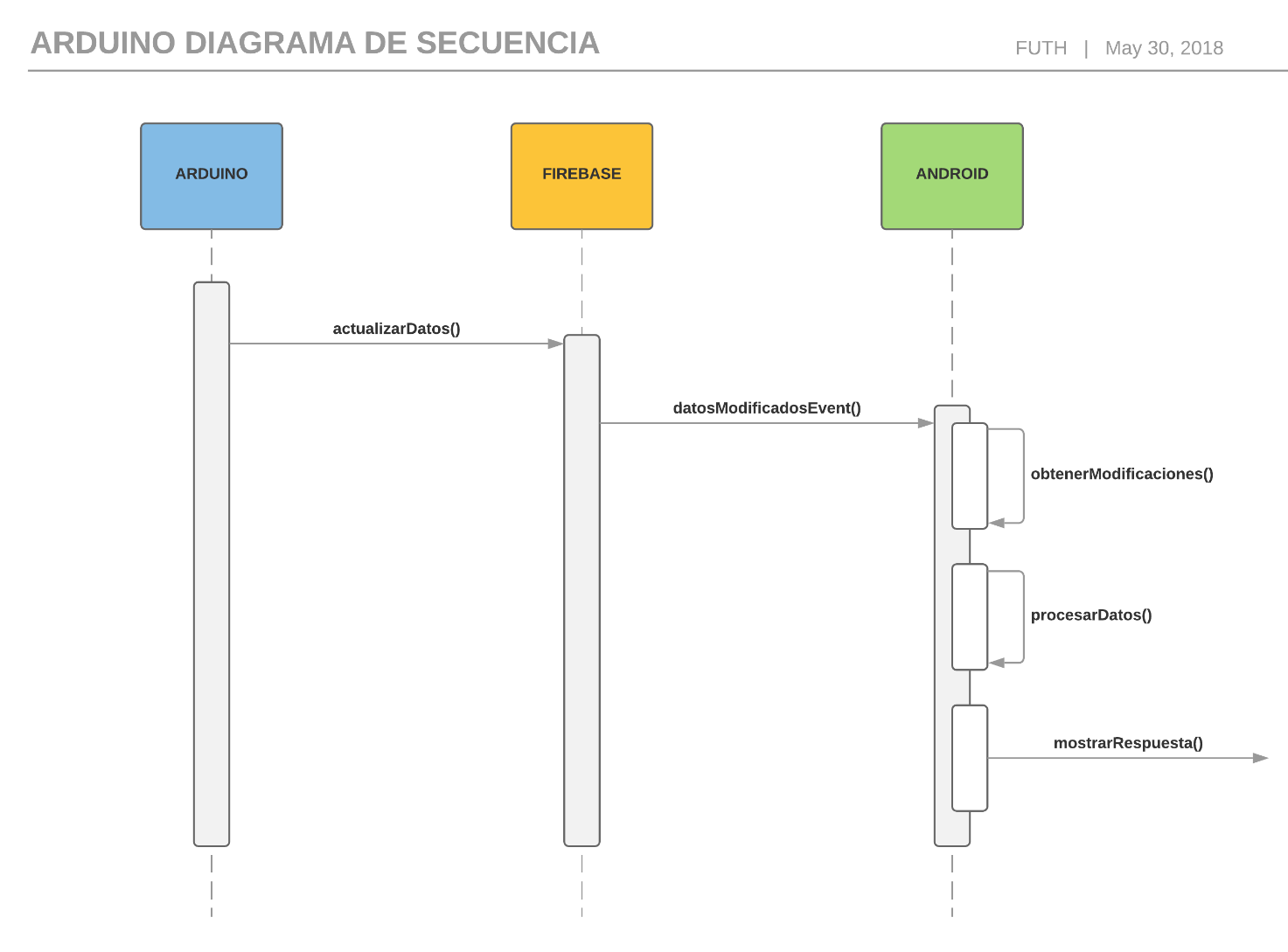
A continuación adjuntamos el **diagrama de secuencia** que hemos elaborado para la aplicación *Android* en general, incluyendo el servicio en segundo plano.



##### Secuencia Arduino

De igual manera, hemos realizado un **diagrama de secuencia** para el funcionamiento de nuestro dispositivo *Arduino*.

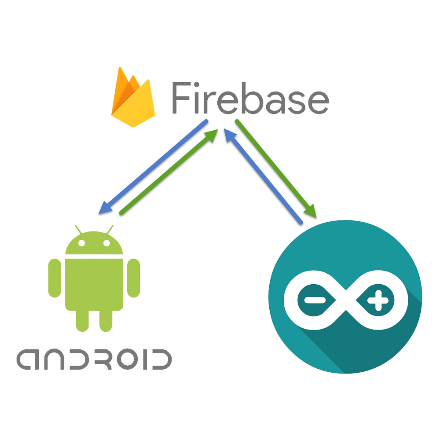
Cabe destacar que este diagrama hay que interpretarlo como un *loop*, debido a que **constantemente** está actualizando los datos de los sensores.



#### Esquemas

##### Esquema de comunicación

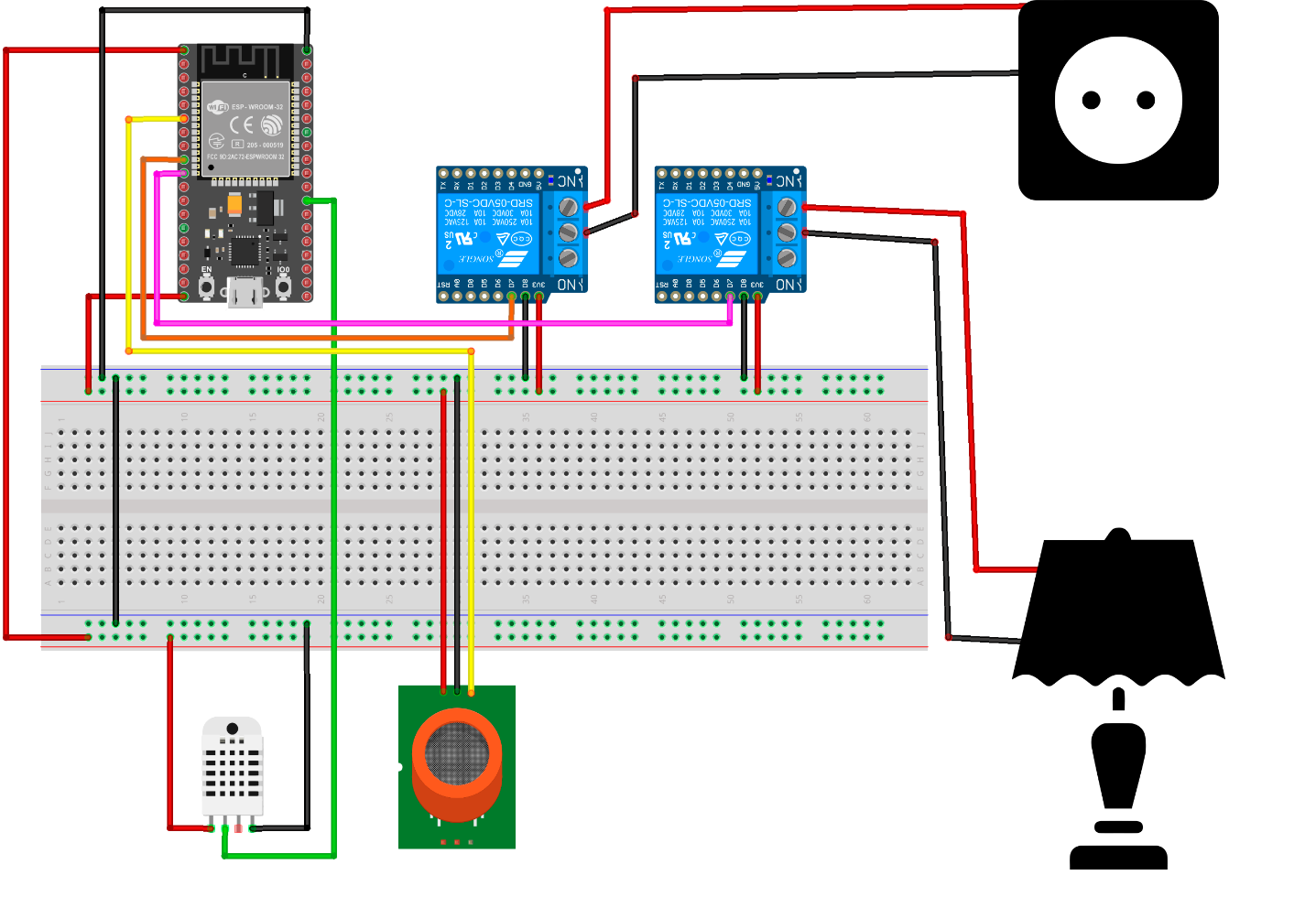
En el siguiente gráfico hemos detallado la manera en la que nuestro dispositivo y la app se deberán comunicar **entre sí**, dando como resultado una comunicación ***p2p*** o ***m2m*** (*machine-to-machine*). Hay que destacar que *Firebase Realtime Database* tiene una forma peculiar de trabajar, muy similar (por no decir idéntica) al protocolo ***MQTT*** ([Ver Anexo 2.D](#_MQTT)), por lo que nosotros seguiremos las directrices de **buenas prácticas** de dicho protocolo.



##### Esquema de conexiones

Hemos realizado un **esquema de conexiones** para todos los componentes físicos que integra nuestro proyecto. Esto ayudará a la hora de **mantener** el hardware en el tiempo y **modificar** el código fuente en base a las propias conexiones.

Los iconos del enchufe y la lámpara representan los objetos cotidianos de los que hemos hablado al principio. Realmente no controlaremos cada uno de ellos, sino que usaremos un relé para controlar si les llega electricidad o no, simulando el control real sobre el objeto. ([Ver Anexo 3.E](#_Relé))



### Documentación funcional

#### Requisitos

La aplicación **debe** permitir registrarse a los usuarios, personalizando el contenido que se muestra según su perfil.

La aplicación **debe** permitir registrar uno o varios dispositivos por el usuario, y mantenerse sincronizada con el/ellos.

La aplicación **debe** disponer de un servicio en segundo plano que se inicie con el sistema para proveer al usuario de notificaciones sobre los sensores que tiene sincronizados.

Las notificaciones que se muestren **deben** ser amigables y desenfadadas para que el usuario no se las tome como un estorbo.

El sensor de gas **podrá** guardar en Firebase **únicamente** un valor comprendido entre 0 y 2, correspondiendo el mismo al nivel de peligro que se ha detectado.

El sensor de humedad **podrá** guardar en Firebase **únicamente** un valor comprendido entre 0 y 100, correspondiendo el mismo al porcentaje de humedad que se ha detectado.

El sensor de temperatura **podrá** guardar en *Firebase* un número con **coma flotante** comprendido entre -40 y 125, correspondiendo el mismo a la temperatura que se ha detectado, en una escala de grados **Celsius**.

Ambos relés **podrán** guardar en *Firebase* **únicamente** una **cadena de caracteres**, ya sea ***on*** u ***off***, representando la misma el estado en el que se encuentra o el que se pretende conseguir.

La denominación **MAC** de los dispositivos comenzará con ***0x***, siendo ésta la **parte fija y común** para todos ellos, seguido de 8 dígitos, que se corresponderán con el número del dispositivo al que hace referencia (**parte variable**). Esto nos asegurará esto un total de **108 (100.000.000) combinaciones posibles**, cumpliendo con el diseño de un producto escalable.

El servicio de notificaciones **debe** ser compatible con *Android Auto* y *Android Wear*, asegurando una mayor comodidad para los usuarios a la hora de recibirlas.

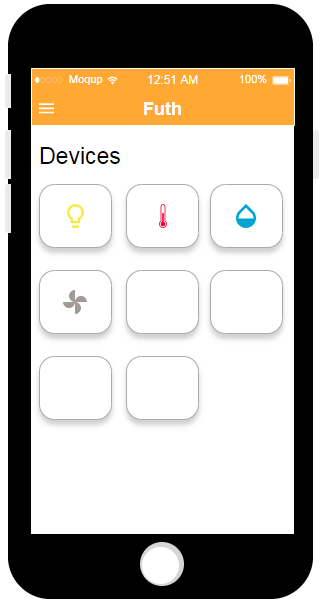
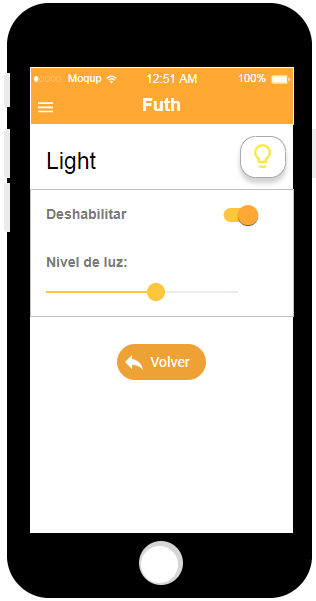
El servicio de notificaciones **solo debe** activarse si hay un usuario con la sesión iniciada, o en el momento de iniciar sesión, en caso contrario, **deberá** estar desactivado.

Las notificaciones **deben** soportar **respuesta por voz** para los dispositivos *Android Auto* y *Android Wear*.

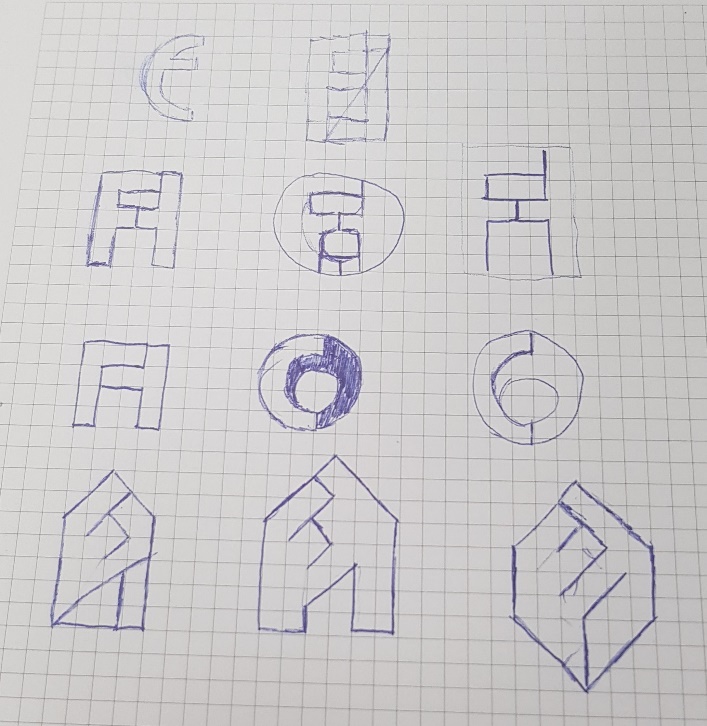
La aplicación **debe** tener una interfaz **clara y sencilla**, con **iconos representativos** de la acción que realizan para la fácil comprensión del usuario.

### Proceso

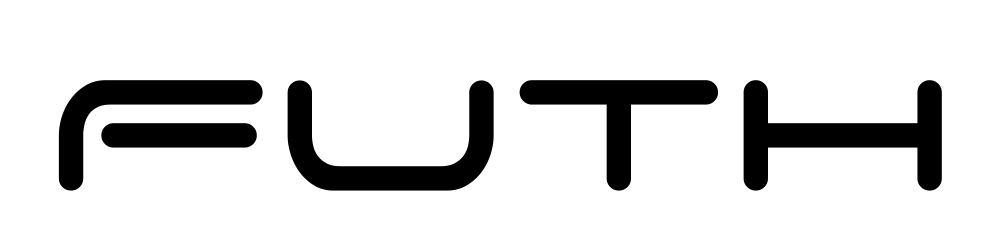
El primer paso de nuestro proyecto fue desarrollar un *mockup*, para tener una visión global de lo que pretendíamos conseguir y una base sobre la que empezar a trabajar.

Una vez realizamos el boceto de la aplicación, nos pusimos a intentar diseñar el logo de la misma. A continuación dejamos unos bocetos iniciales.

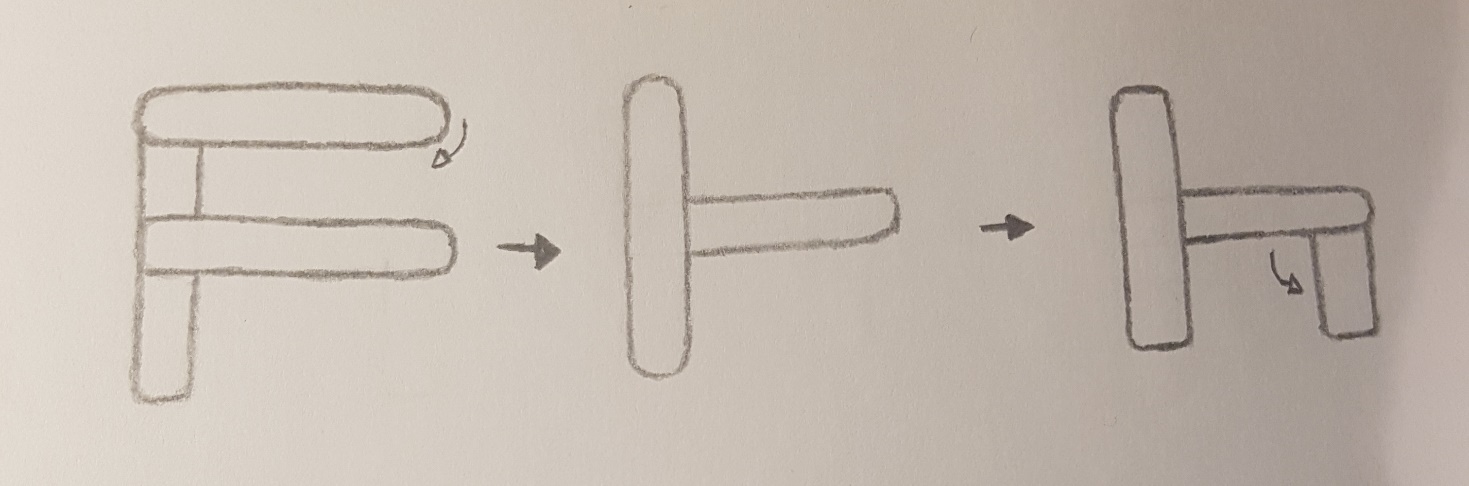


Finalmente tomamos la decisión en elegir un logo menos complejo, acorde con el estilo *material* de la app. Obtuvimos un logo estilo icono y otro de texto para aplicar en títulos y demás.

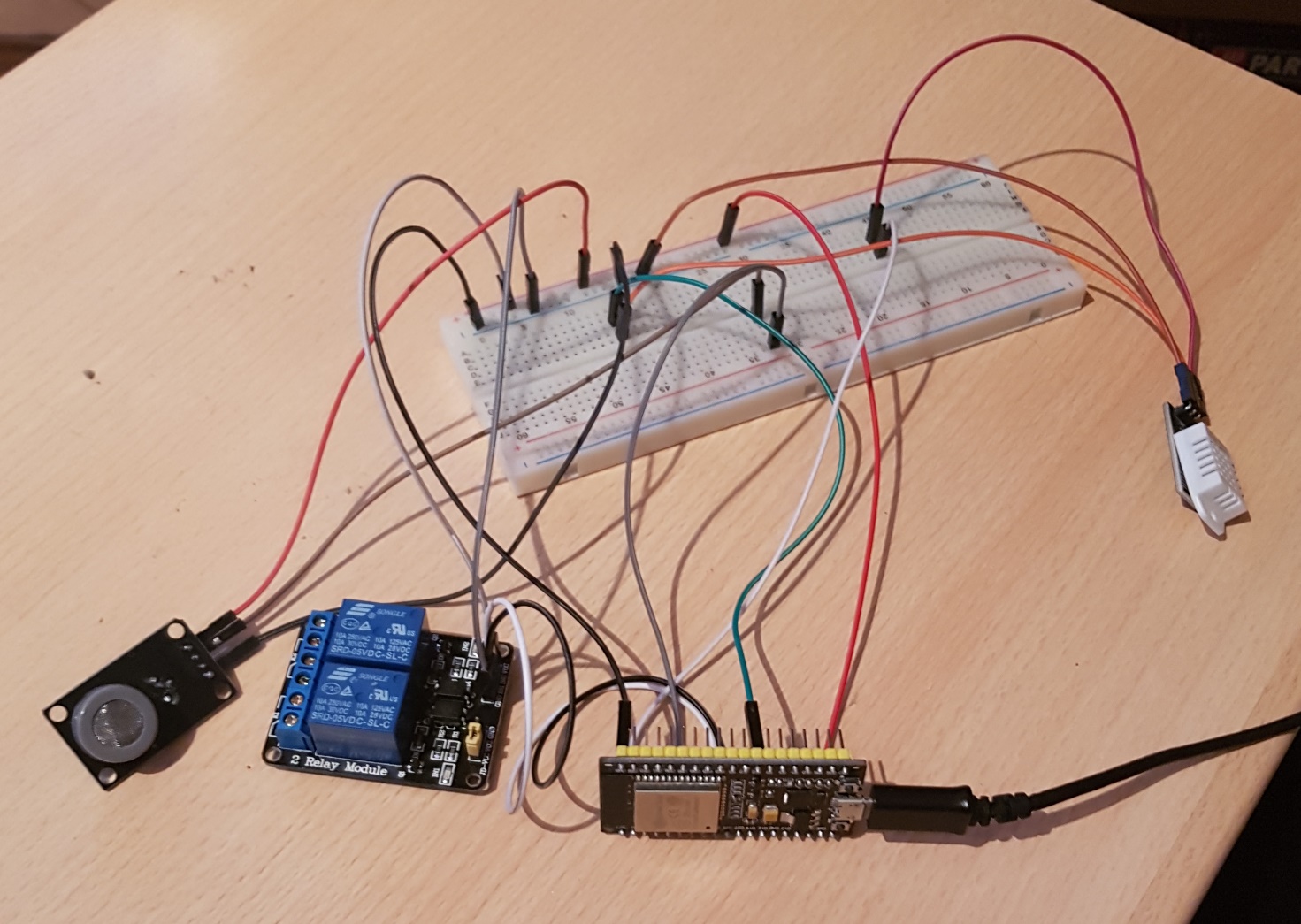




De igual manera, desarrollamos un **boceto** de cómo debía transcurrir la animación que posteriormente haríamos con el logo para implementar en el *splash screen*.



Terminados el logo y la animación, nos pusimos a desarrollar el dispositivo que sería con el que la app se conectara como se puede ver a continuación.



Finalizado el proyecto y comprobado que todo funciona correctamente, montaremos una maqueta donde colocar de forma más ordenada y visual todos los componentes.

Una vez tuvimos el dispositivo físico desarrollado (aún con fallos), nos pusimos a desarrollar la aplicación *Android*. Al mismo tiempo que íbamos creando la aplicación, mejoramos y/o añadimos fragmentos de código al programa de *Arduino*, para ir corrigiendo errores y conseguir el mejor rendimiento posible.

### Desarrollo Arduino

Para el desarrollo de *Arduino* hemos necesitado tres **librerías externas**, para poder completar todos los puntos de requisitos que detallamos.

#### IOXhop\_FirebaseESP32

Mediante esta librería implementaremos la conexión Wifi a nuestro módulo ***ESP32***, además de la conexión con *Firebase*.

Será la encargada de todas las peticiones que hagamos, de recuperar los datos necesarios y de comprobar los **eventos de modificación** que se producen en *Firebase*.

#### SimpleTimer

Con esta librería podremos **modularizar** el código del dispositivo al máximo, permitiéndonos crear ***Threads***. Cada *thread* ejecutará una función en concreto, permitiéndonos elegir si ha de ser periódicamente, una única vez o al cabo de un tiempo, e incluso nos permite gestionar los propios *threads*, deshabilitándolos, volviéndolos a habilitar o eliminándolos según necesitemos.

#### DHTesp

Esta librería nos permitirá realizar las mediciones del sensor de **temperatura y humedad**, y obtener la que necesitemos en cada momento. Además, cuenta con funciones que nos devuelven los datos ya **formateados** (por ejemplo, en grados *Celsius* o *Fahrenheit*), y funciones que en caso de producirse un error, devuelven el mismo acompañado de una breve descripción.

#### Código

Al inicio del programa declaramos las **variables globales** de las que hará uso nuestro *Arduino*, y siguiendo con buenas prácticas orientadas al rendimiento, solo las inicializaremos en el momento en el que se vayan a usar. ([Ver Anexo 4.I.i](#_Inicialización_de_variables))

En la función ***setup()*** del programa tendremos la inicialización de todos los componentes que se van a usar durante la ejecución. Lo primero que hacemos es iniciar el *Serial*, para poder poner trazas en el código y monitorizar qué está haciendo en cada momento, y a continuación, todos los sensores que vamos a usar. Seguidamente inicializamos el *Wifi* y nos conectamos a la red que se haya configurado (sin este paso no continua la ejecución del programa). Una vez hecho esto inicializamos el *listener* de *Firebase* y todos los *threads*. ([Ver Anexo 4.I.ii](#_Función_setup))

La función ***loop()*** normalmente es la que ejecuta el código funcional del programa, siendo ésta un bucle infinito en el que se repite una y otra vez el mismo código, pero debido a la naturaleza de nuestro desarrollo y a la necesidad de utilizar *threads*, en esta ocasión contendrá una única línea de código. Dicha línea se encarga de comprobar constantemente en qué momento debe ejecutar cada *thread* que se ha definido ([Ver Anexo 4.I.iii](#_Función_loop)).

Se han definido una serie de funciones que serán las encargadas de monitorizar y comprobar los sensores, pudiendo establecer un *refresh time* personalizado para cada una. También se encargarán de sincronizar los datos obtenidos en *Firebase* ([Ver Anexo 4.I.iv](#_Funciones_update)).

En el código se han dejado varias funciones que no se usan, pero que en un futuro se pueden llegar a necesitar: ***startTimers()*** y ***stopTimers()***. Éstas se encargan de habilitar o deshabilitar los *threads* definidos en el programa ([Ver Anexo 4.I.v](#_Funciones_timers)).

En el apartado *Wifi* hemos definido dos funciones, una es la que se ejecuta al inicio del programa, impidiendo seguir con la ejecución hasta que se conecte a una red y la otra es la que se ejecutará periódicamente para comprobar el estado de la conexión, y en caso de que se haya desconectado, intentar la reconexión ([Ver Anexo 4.I.vi](#_Funciones_Wifi)).

Las funciones *Firebase* son las que se encargan de inicializar las variables globales del programa según se conecta el dispositivo a la red y las que monitorizan los cambios que se producen, actuando correspondientemente en base a ellos ([Ver Anexo 4.I.vii](#_Funciones_Firebase)).

### Desarrollo Android

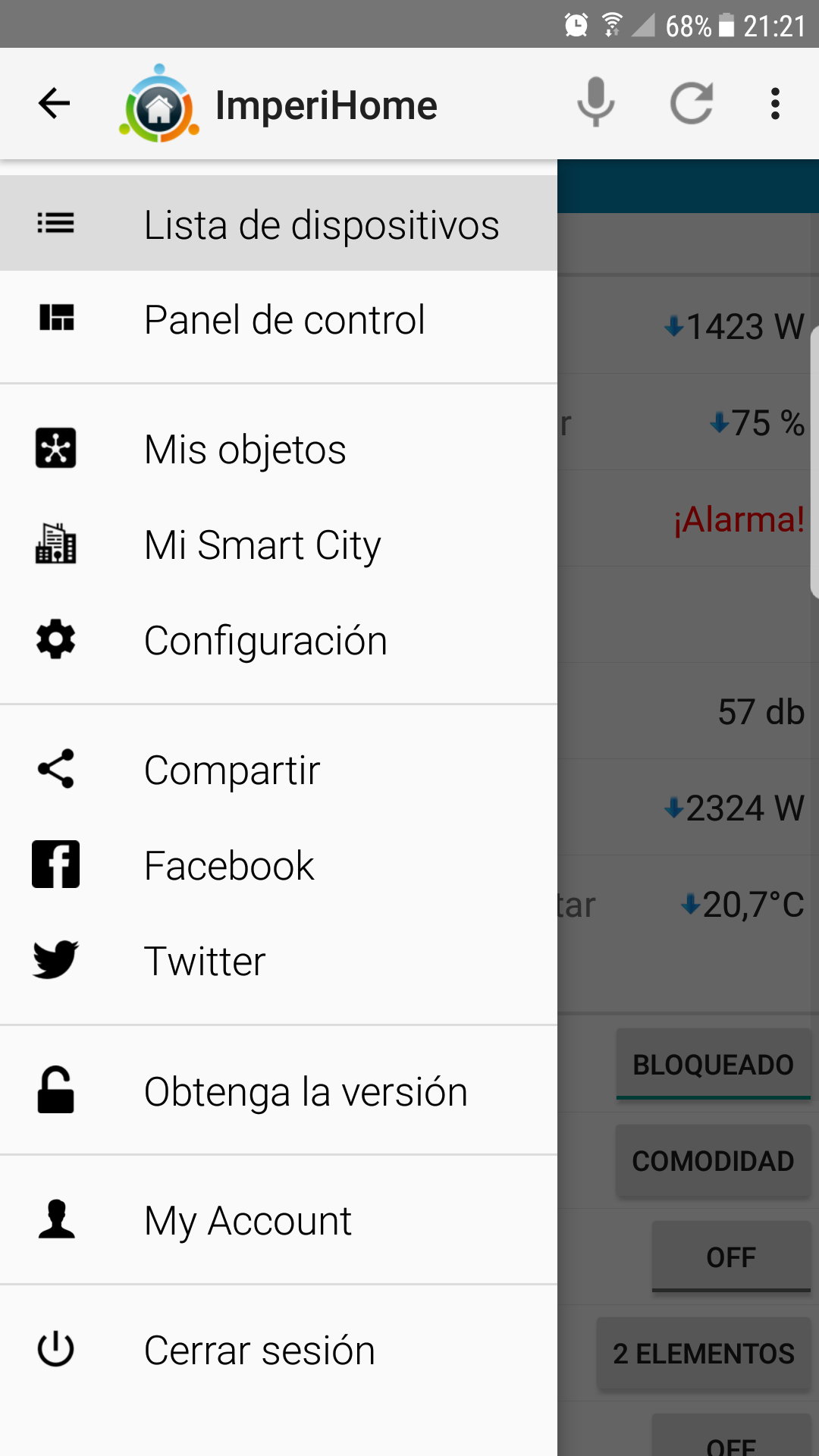
# Bibliografía y Webgrafía

* [Arduino](https://www.arduino.cc/)
* [¿Qué es Firebase?](https://elandroidelibre.elespanol.com/2016/05/firebase-plataforma-desarrollo-android-ios-web.html)
* [Productos Firebase](https://firebase.google.com/products/?hl=es-419&gclid=Cj0KCQjw0a7YBRDnARIsAJgsF3OLHnSP6gsxEHp0JKOdHHQUjMK_uVP-APz4Nzvux9Xr1eBoHig1VIAaAi10EALw_wcB)
* [Firebase](https://firebase.google.com/)
* [¿Qué es MQTT?](https://geekytheory.com/que-es-mqtt)
* [MQTT](http://mqtt.org/)
* [C++ Wikipedia](https://es.wikipedia.org/wiki/C%2B%2B)
* [Arduino Wikipedia](https://es.wikipedia.org/wiki/Arduino)
* [¿Cómo funciona un relé?](http://www.areatecnologia.com/electricidad/rele.html)
* [Sensores MQ](https://naylampmechatronics.com/blog/42_Tutorial-sensores-de-gas-MQ2-MQ3-MQ7-y-MQ13.html)
* [Sensor MQ9](http://teslabem.com/sensor-de-co2-y-gases-mq9.html)
* [Sensor DHT22](https://www.luisllamas.es/arduino-dht11-dht22/)
* [¿Qué es el ESP32?](https://www.luisllamas.es/esp32/)

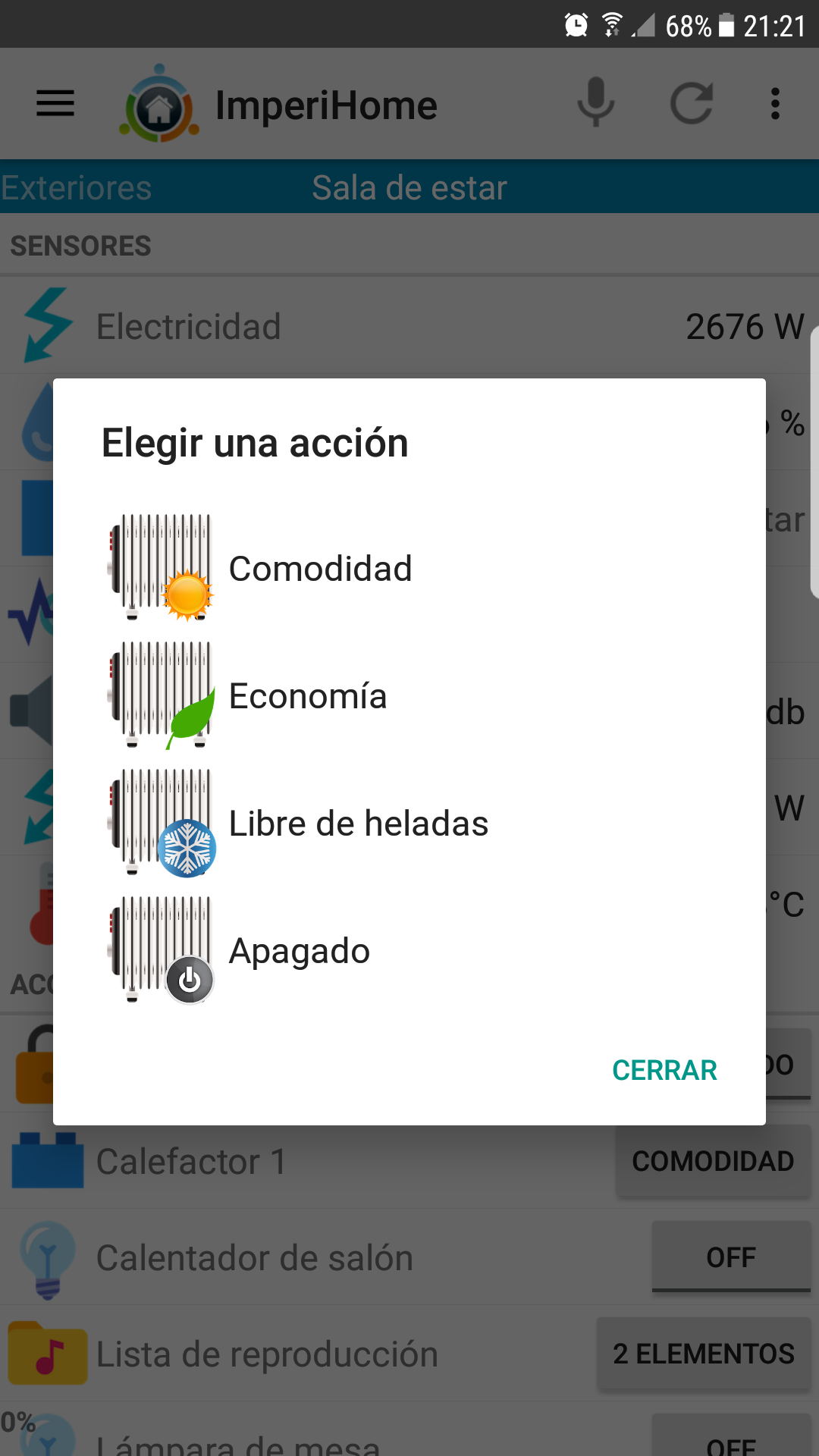
# Anexos

## Apps domótica

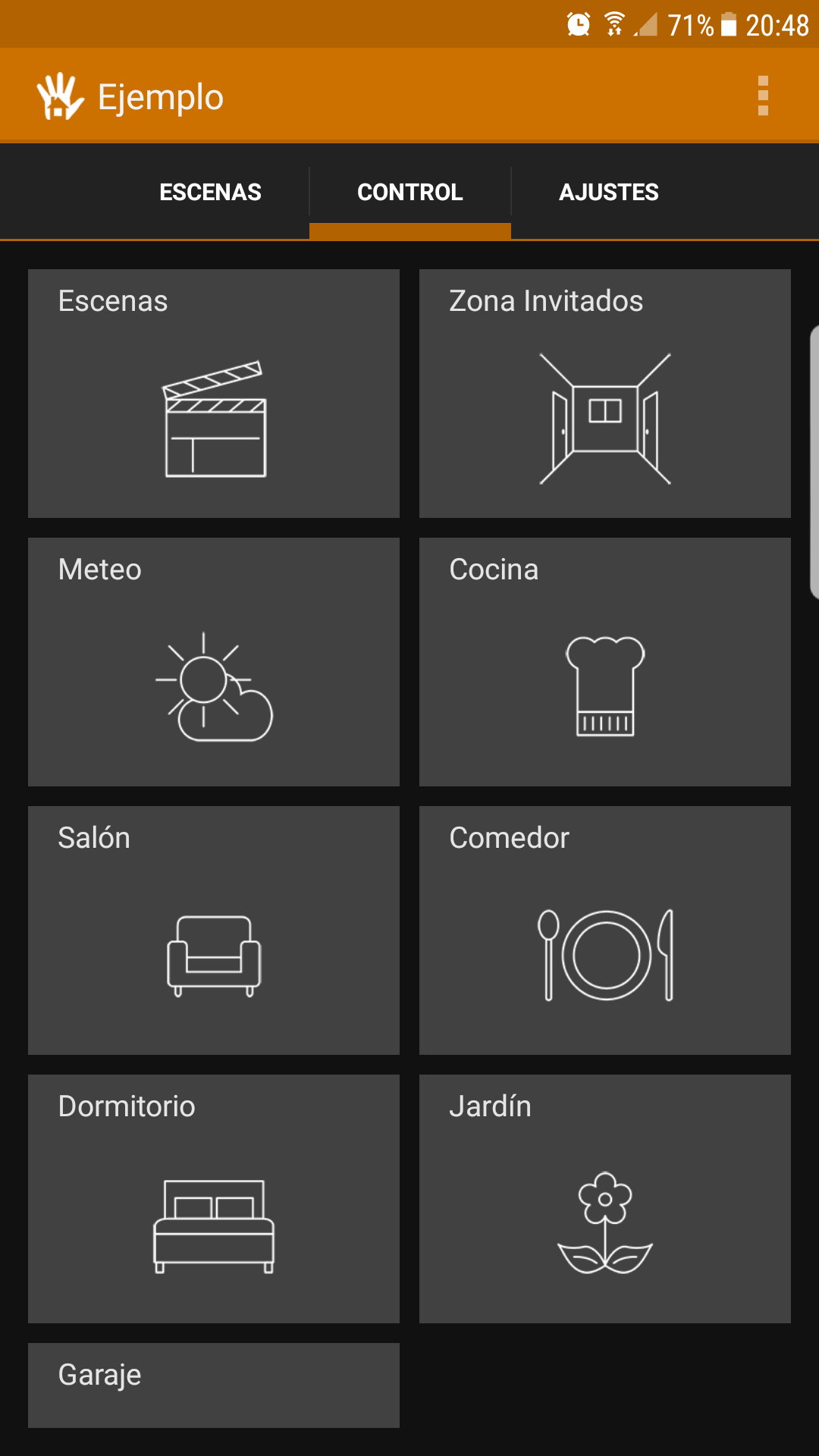
### Imperihome

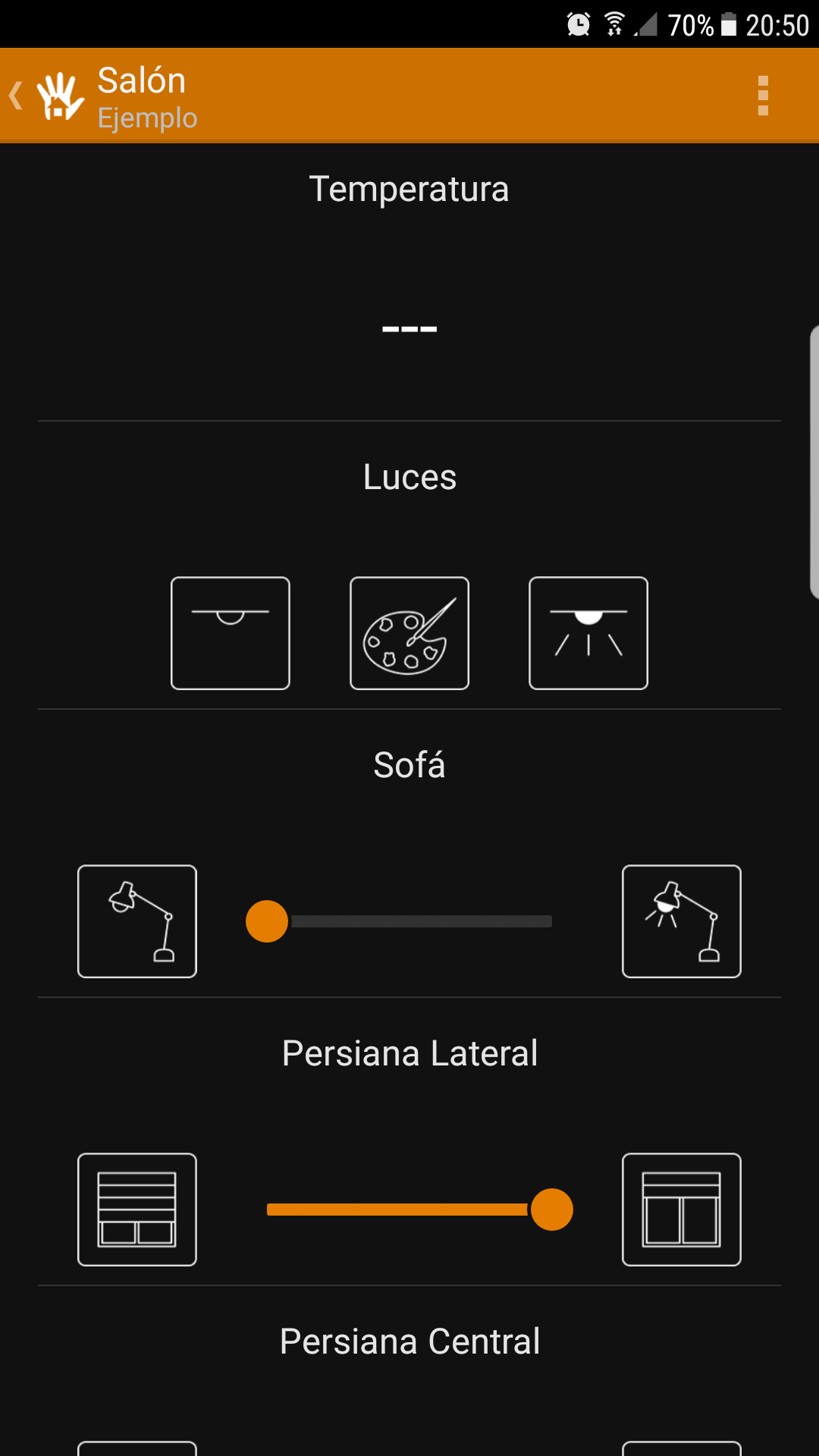




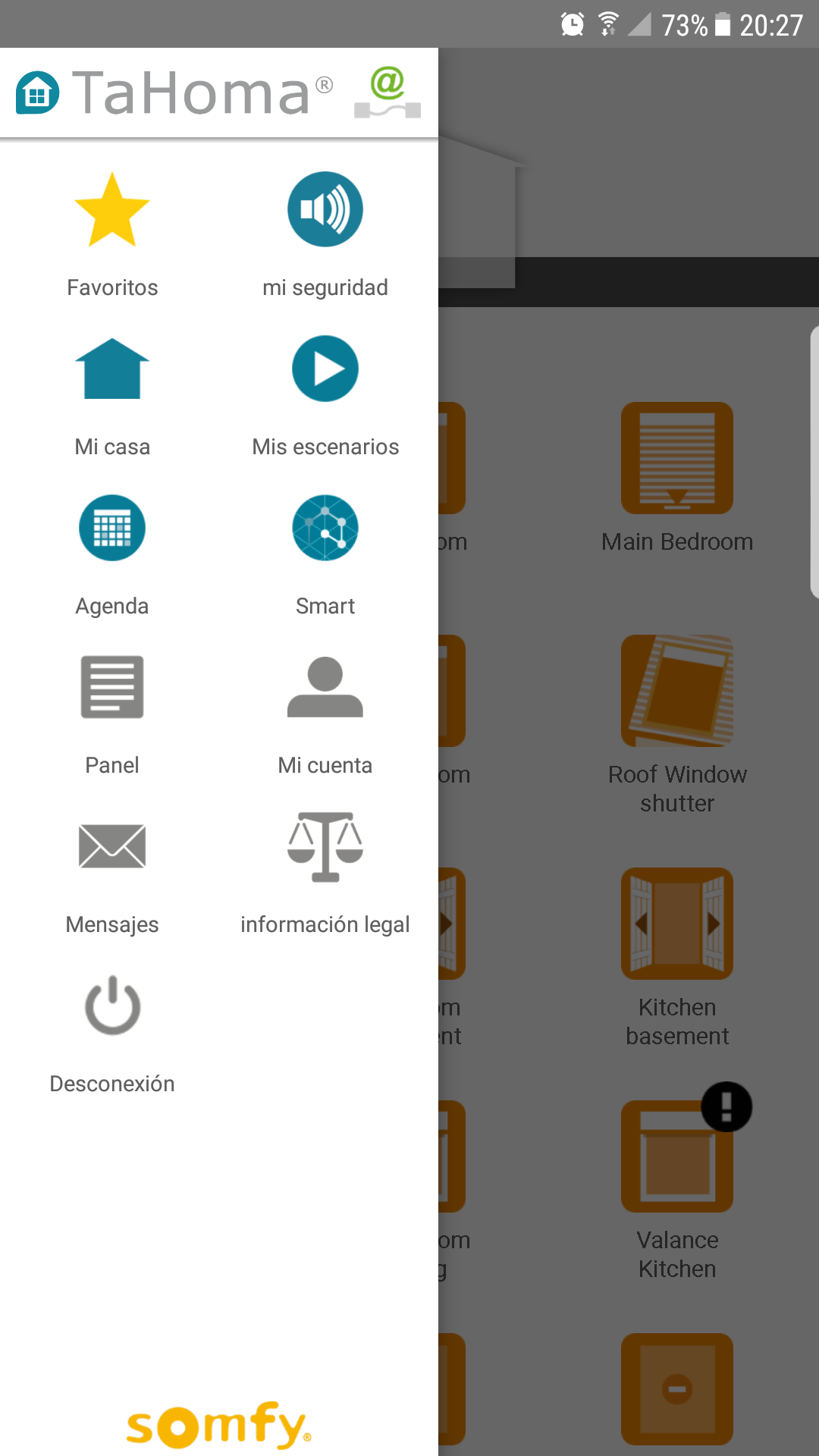


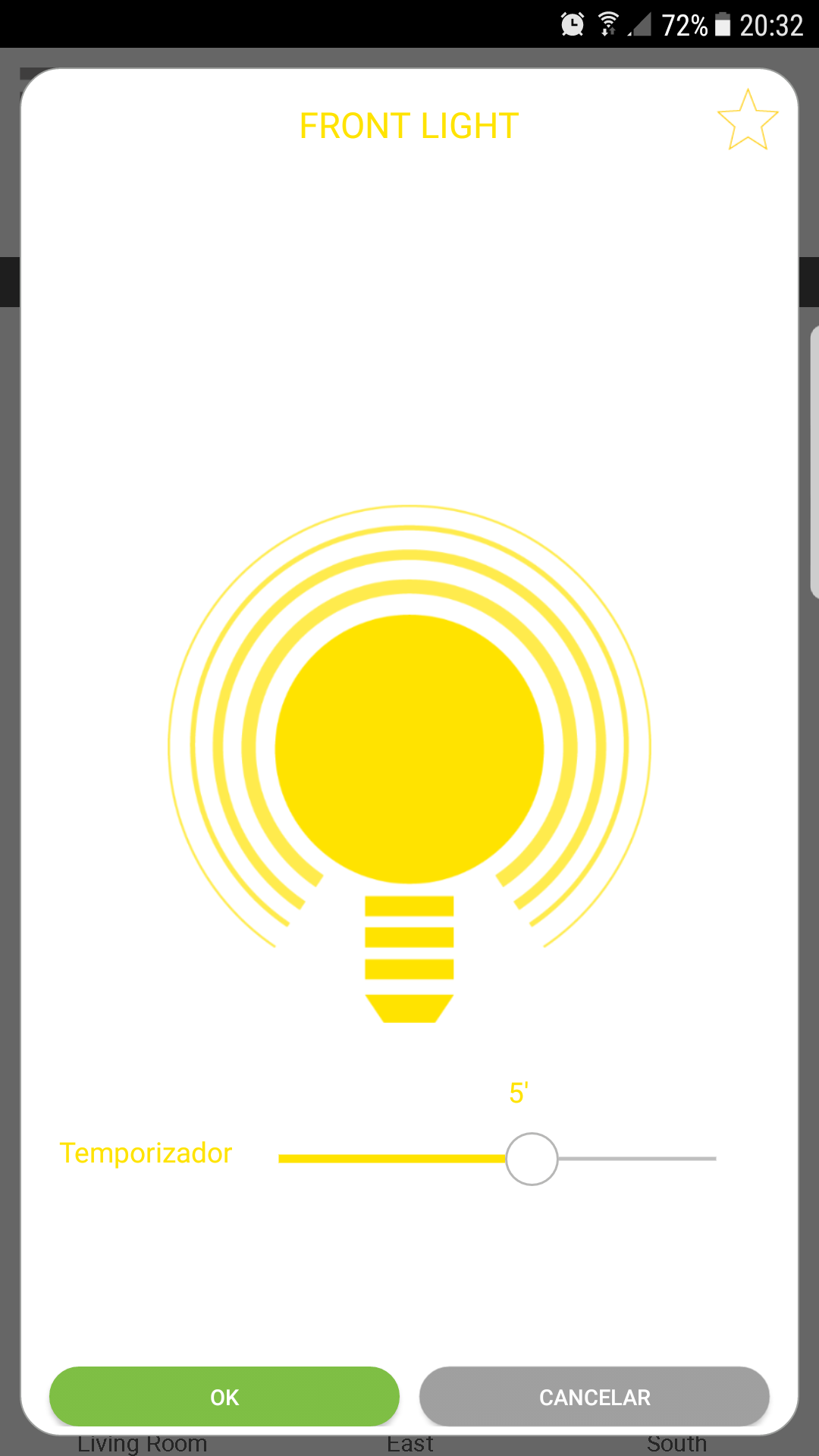
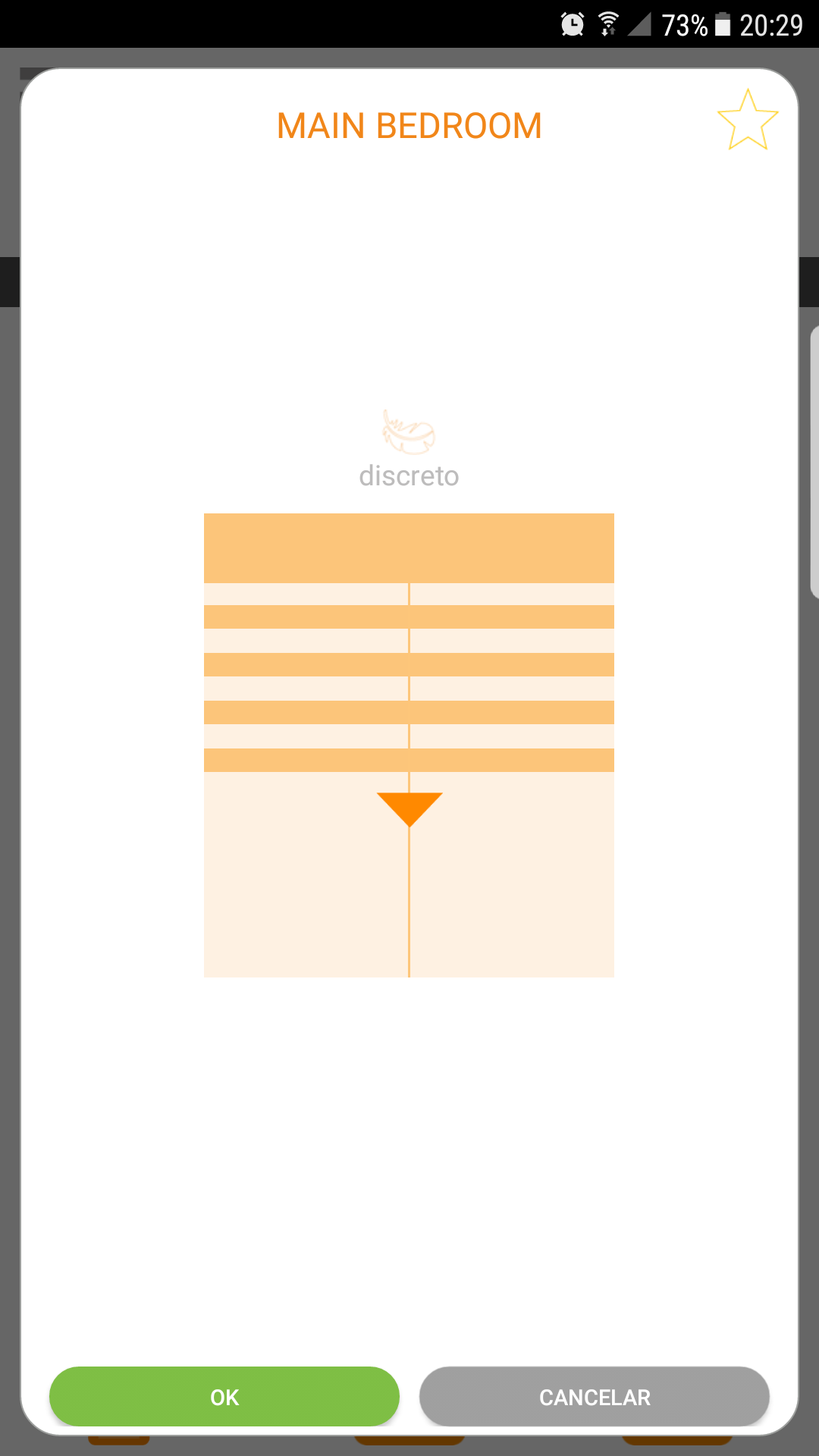
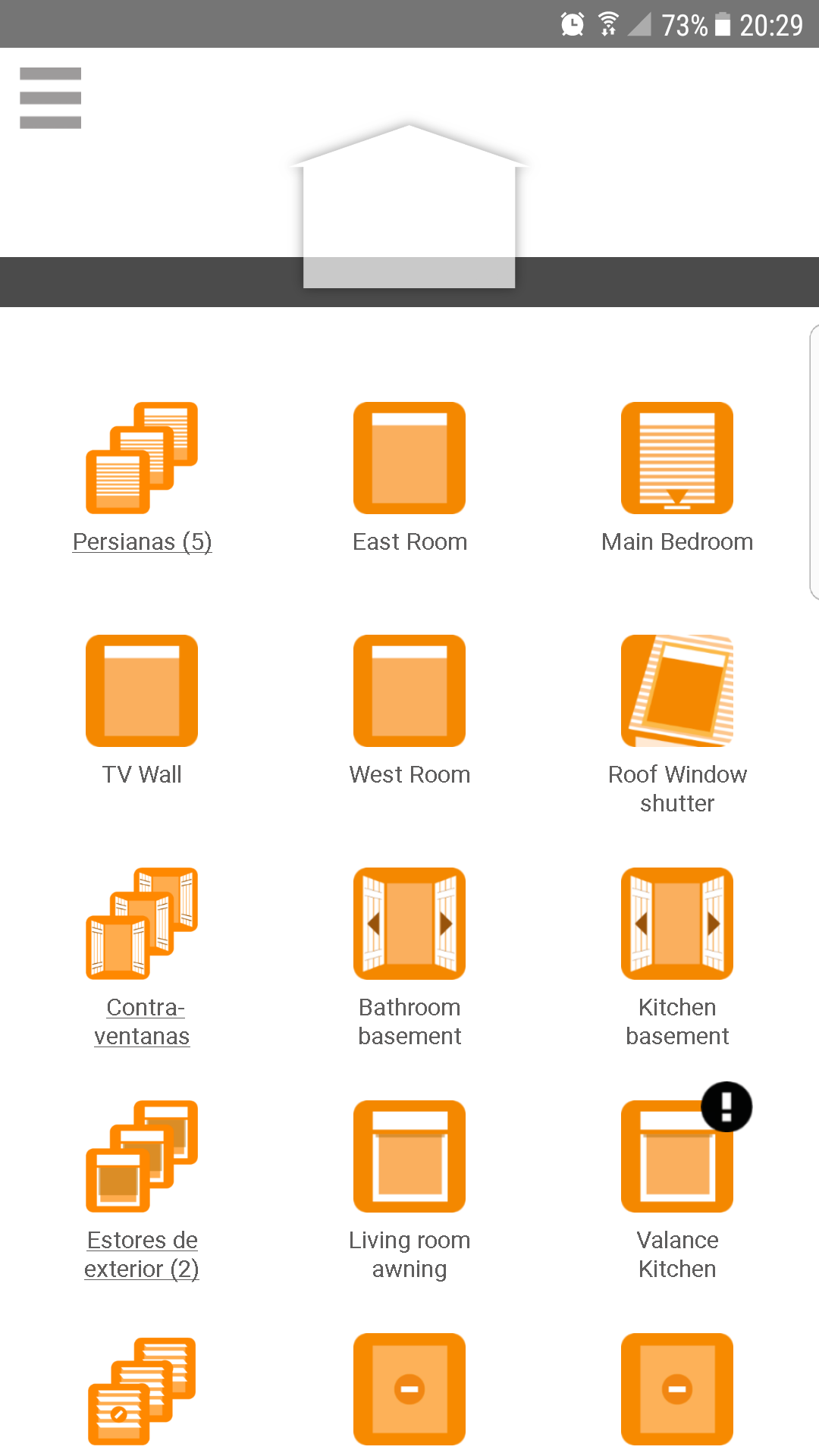
### Houseinhand KXN





### TaHoma by Somfy





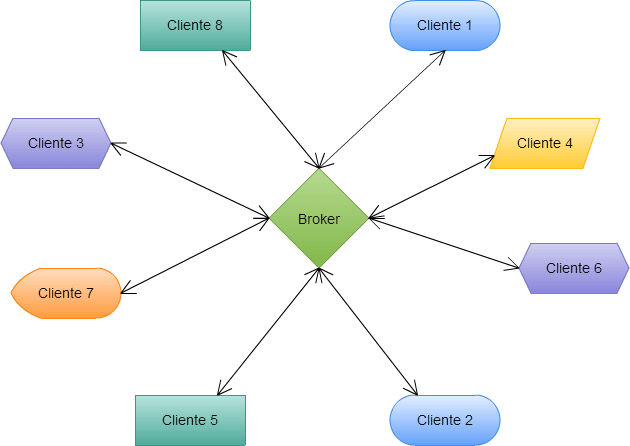
## Protocolos

### MQTT

MQTT (*Message Queue Telemetry Transport*) es un protocolo usado para la comunicación *machine-to-machine* (M2M) en el **Internet of Things**.

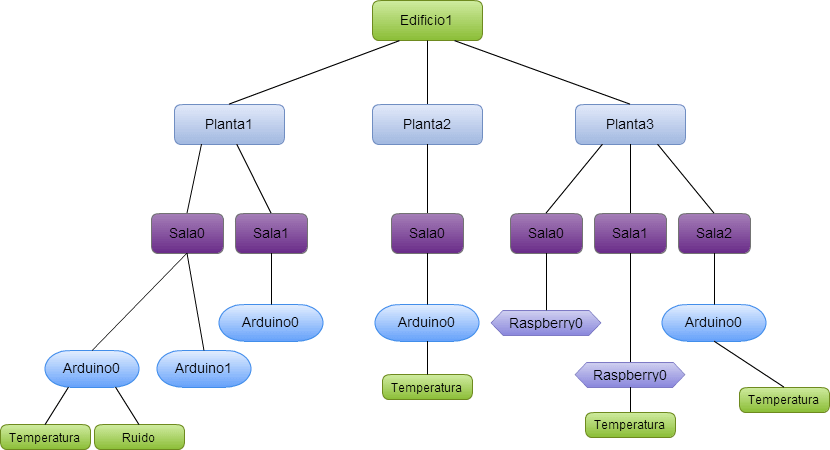
Este protocolo está orientado a la comunicación entre sensores, debido a que consume muy poco ancho de banda y puede ser usado en la mayoría de dispositivos de desarrollo con pocos recursos.

La arquitectura de MQTT sigue una topología de estrella, con un nodo central que hace de servidor o *bróker* con una capacidad de hasta 10000 clientes. El bróker es el encargado de gestionar la red y de transmitir los mensajes, para mantener activo el canal, los clientes mandan periódicamente un paquete *PINGREQ* y esperan la respuesta *PINGRESP* del bróker. Además, la comunicación puede ser cifrada, entre muchas otras opciones, aunque el protocolo no fuera diseñado específicamente pensando en ello.



La comunicación se basa en unos *topics* que el cliente que publica crea, y los nodos que deseen recibirlo deben suscribirse a él. La comunicación puede ser de **uno a uno** o de **uno a muchos** (1:1 o 1:N).

Un topic se representa mediante una cadena y tiene una estructura jerárquica. Cada jerarquía se separa con **‘/’**, por ejemplo “*edificio1/planta5/sala1/arduino2/temperatura*” o “*edificio3/planta0/sala3/arduino4/humedad*”. De esta forma se pueden crear jerarquías de clientes que publican y reciben datos, como observamos a continuación.



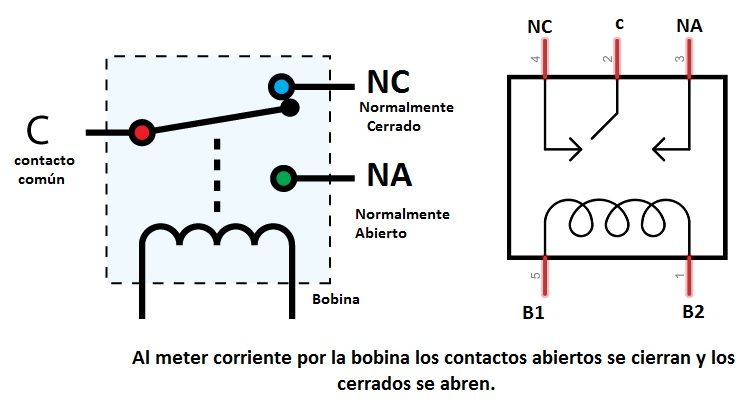
Así, un nodo puede suscribirse a un topic concreto (“*edificio3/planta0/sala3/arduino4/humedad*”) o a varios a la vez (“*edificio2/planta5/#*”).

## Componentes

### Relé

Un *relé* o relevador es un aparato eléctrico que funciona como un interruptor, abriendo y cerrando el paso de la corriente eléctrica. La peculiaridad de este interruptor, es que es **accionado eléctricamente**.

Normalmente un relé está formado por una bobina y 2 contactos. Al hacer pasar corriente por la bobina se crea un campo electromagnético que atrae uno de los contactos y simula el efecto de interruptor.

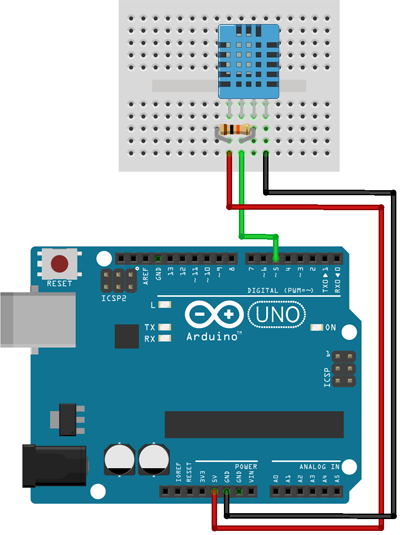
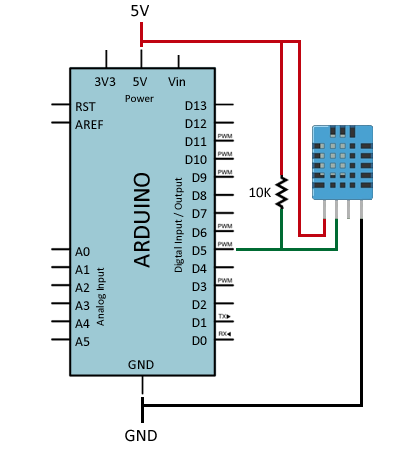


### Sensor DHT22

El DHT22 es un sensor perteneciente a la familia DHT, el cual podríamos considerar el *hermano mayor*, debido a que es el que cuenta con mejores especificaciones. Es muy común en los desarrollos caseros debido a su precisión y bajo coste. Este modelo no solo mide la temperatura, sino que también es capaz de medir la humedad en el ambiente. Sus especificaciones son las siguientes:

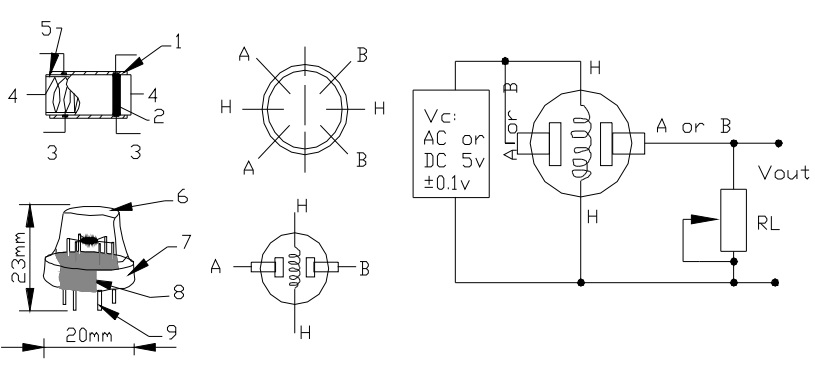
* Medición de temperatura entre -40 y 125, con una precisión de 0.5 grados Celsius.
* Medición de humedad entre 0 y 100, con una precisión entre el 2 y el 5 por ciento.
* Frecuencia de muestreo de 2 mediciones por segundo (2 Hz)

A continuación os mostramos un ejemplo de conexión de este sensor con Arduino.



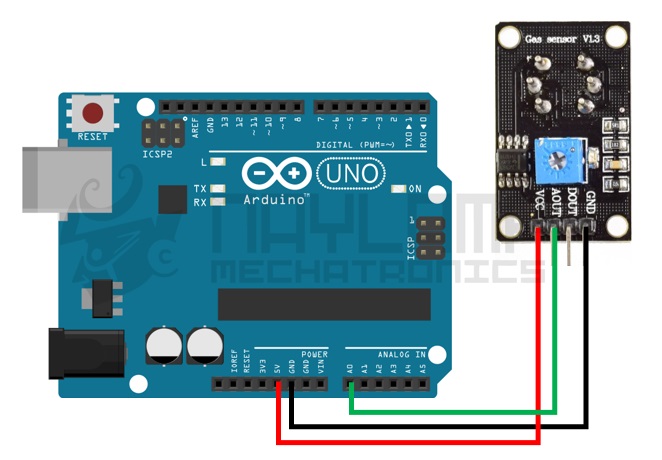
### Sensor MQ9

Este es un sensor de gases, concretamente de monóxido de carbono y gases inflamables. Su funcionamiento es analógico, por lo que es fácil de implementar con la mayoría de microcontroladores. Este tipo de sensor es electroquímico, variando su resistencia cuando se exponen a determinados gases. Internamente tiene un calentador para aumentar la temperatura interna y así obtener mediciones más precisas.



Pertenece a la familia de sensores MQ, donde podemos encontrar gran variedad a la hora de elegir. El *MQ2* es un sensor de gas combustible, el *MQ3* de alcohol, el *MQ7* de monóxido de carbono (es el hermano pequeño del *MQ9*), el *MQ135* mide la calidad del aire, y un largo etc. A continuación dejamos las características del MQ9 y un ejemplo de conexión.

* Medición entre 10-1000 ppm de CO u entre 100-10000 ppm de gas combustible.
* Lectura analógica o digital
* Alta sensibilidad al metano, propano y CO.
* Operativo con temperaturas de -10ºC hasta 50ºC.
* Consumo menor a 150mV con corriente de 5V.

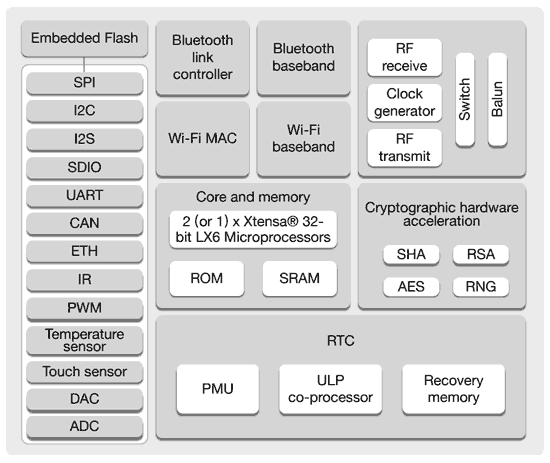


### Microcontrolador ESP32

Este es uno de los mejores microcontroladores que existen en la actualidad, debido a la relación calidad/potencia/precio que posee. Tiene dos núcleos, uno dedicado completamente al Wifi y Bluetooth, lo que reduce los problemas con el código que nosotros le cargamos. Está diseñado para ser compatible con las plataformas de Arduino y con muchas más del mismo estilo, tales como Raspberry y Beaglebone. Pertenece a la familia de los ESP, siendo éste el hermano mayor del ESP8266.

Además, es de código y hardware libre, lo cual favorece a tener una comunidad de desarrolladores mucho más amplia y activa. A continuación dejamos sus especificaciones y un esquema de su diseño de funcionamiento.

* Procesador 32 bits de doble núcleo
* Velocidad de 160 MHz (Con un máximo de 240 MHz)
* Memoria SRAM de 520 KiB y memoria flash externa de hasta 16 MiB con encriptación
* Wifi 802.11 b/g/n 2.4 GHz y Bluetooth v4.2
* 32 pines GPIO con soporte PWM, UART, SPI, I2S, I2C, ADC, DAC y CAN Bus



## Código

### Arduino

#### Inicialización de variables

**#**include **<**IOXhop\_FirebaseESP32.h**>**

**#**include **<**SimpleTimer.h**>**

**#**include **<**DHTesp.h**>**

// Se define la red wifi y la contraseña

**#**define WIFI\_SSID **"**SSID**"**

**#**define WIFI\_PASSWORD **"**PASSWORD**"**

**#**define TEMPERATURE1\_PATH **"/**devices**/**0x00000001**/**temperature1**/**value**"**

**#**define HUMIDITY1\_PATH **"/**devices**/**0x00000001**/**humidity1**/**value**"**

**#**define GAS1\_PATH **"/**devices**/**0x00000001**/**gas1**/**risk**"**

**#**define LIGHT\_PATH **"/**devices**/**0x00000001**/**rele1**/**state**"**

**#**define PLUG\_PATH **"/**devices**/**0x00000001**/**rele2**/**state**"**

// Se define el Timer y cuantos 'hilos' ejecutara el programa

SimpleTimer t;

int id\_timer\_gas1**,**id\_timer\_temperature1**,** id\_timer\_humidity1**,** id\_timer\_light**,** id\_timer\_plug;

// Se declara el stream que comprobara los cambios de Firebase

StreamHandlerCallback stream;

// DHT Variables

const int DHT\_pin **=** 16; // Pin del sensor DHT

DHTesp dht; // Se instancia el sensor DHT

static **char** celsiusTemp**[**7**]**;

static **char** humidityTemp**[**7**]**;

// MQ9 Variables

const int MQ9\_pin **=** 35;

// Reles Variables

const int light\_pin **=** 26;

const int plug\_pin **=** 25;

// Se definen las variables del programa

bool initialized **=** false;

int gas1\_risk;

String light\_state;

String plug\_state;

float temperature1\_value;

float humidity1\_value;

#### Función setup

void **setup()** {

Serial.begin**(**115200**)**;

// Inicializacion de los sensores

dht.setup**(**DHT\_pin**,** DHTesp::DHT22**)**; // Se inicializa el sensor DHT

**pinMode(**MQ9\_pin**,** **INPUT)**; // Se establece el pin de MQ9 como entrada de datos

**pinMode(**light\_pin**,** **OUTPUT)**; // Se inicializa el rele de la luz

**pinMode(**plug\_pin**,** **OUTPUT)**; // Se inicializa el rele del enchufe

initializeWifi**()**; // Se inicializa el Wifi

initializeFirebase**()**; // Se inicializa la monitorizacion de Firebase

}

#### Función loop

void **loop()** {

t.run**()**;

}

#### Funciones update

void updateGas**()** {

// Lectura del sensor de gas

int gas1\_value **=** **analogRead(**MQ9\_pin**)**;

if **(**gas1\_value **>=** 0 && gas1\_value **<** 1365**)** {

gas1\_risk **=** 0; // Se establece un riesgo bajo para la salud

}

else if **(**gas1\_value **>=** 1365 && gas1\_value **<** 2730**)** {

gas1\_risk **=** 1; // Se establece un riesgo medio para la salud

}

else {

gas1\_risk **=** 2; // Se establece un riesgo alto para la salud

}

// Actualizamos el riesgo de gas en Firebase

Serial.println**("**Updated Gas Risk: **"** **+** String**(**gas1\_risk**))**;

Firebase.set**(**GAS1\_PATH**,** gas1\_risk**)**;

}

void updateTemperature**()** {

// Lectura de la temperatura y humedad

TempAndHumidity newValues **=** dht.getTempAndHumidity**()**;

if **(**dht.getStatus**()** **!=** 0**)** {

Serial.println**("**DHT22 error status: **"** **+** String**(**dht.getStatusString**()))**;

}

temperature1\_value **=** dht.computeHeatIndex**(**newValues.temperature**,** newValues.humidity**)**;

// Actualizamos la temperatura en Firebase

Serial.println**("**Updated Temperature Value: **"** **+** String**(**temperature1\_value**))**;

Firebase.set**(**TEMPERATURE1\_PATH**,** temperature1\_value**)**;

}

void updateHumidity**()** {

// Lectura de la temperatura y humedad

TempAndHumidity newValues **=** dht.getTempAndHumidity**()**;

if **(**dht.getStatus**()** **!=** 0**)** {

Serial.println**("**DHT22 error status: **"** **+** String**(**dht.getStatusString**()))**;

}

humidity1\_value **=** dht.computeDewPoint**(**newValues.temperature**,** newValues.humidity**)**;

// Actualizamos la humedad en Firebase

Serial.println**("**Updated Humidity Value: **"** **+** String**(**humidity1\_value**))**;

Firebase.set**(**HUMIDITY1\_PATH**,** humidity1\_value**)**;

}

void updateLight**()** {

if **(**light\_state **==** **"**on**")** {

**digitalWrite(**light\_pin**,** **LOW)**;

}

else if **(**light\_state **==** **"**off**")** {

**digitalWrite(**light\_pin**,** **HIGH)**;

}

}

void updatePlug**()** {

if **(**plug\_state **==** **"**on**")** {

**digitalWrite(**plug\_pin**,** **LOW)**;

}

else if **(**plug\_state **==** **"**off**")** {

**digitalWrite(**plug\_pin**,** **HIGH)**;

}

}

#### Funciones timer

void initializeTimers**()** {

id\_timer\_gas1 **=** t.setInterval**(**5000**,** updateGas**)**;

id\_timer\_temperature1 **=** t.setInterval**(**30000**,** updateTemperature**)**;

id\_timer\_humidity1 **=** t.setInterval**(**30000**,** updateHumidity**)**;

id\_timer\_light **=** t.setInterval**(**1000**,** updateLight**)**;

id\_timer\_plug **=** t.setInterval**(**1000**,** updatePlug**)**;

t.setInterval**(**5000**,** reconnectWifi**)**; // Se monitoriza el estado de la conexion wifi

}

void stopTimers**()** {

t.disable**(**id\_timer\_gas1**)**;

t.disable**(**id\_timer\_temperature1**)**;

t.disable**(**id\_timer\_humidity1**)**;

t.disable**(**id\_timer\_light**)**;

t.disable**(**id\_timer\_plug**)**;

}

void startTimers**()** {

t.enable**(**id\_timer\_gas1**)**;

t.enable**(**id\_timer\_temperature1**)**;

t.enable**(**id\_timer\_humidity1**)**;

t.enable**(**id\_timer\_light**)**;

t.enable**(**id\_timer\_plug**)**;

}

#### Funciones Wifi

void initializeWifi**()** {

WiFi.setAutoReconnect**(**true**)**; // Establecemos que si se pueda usar la funcion de reconnectar si se pierde la conexion

WiFi.begin**(**WIFI\_SSID**,** WIFI\_PASSWORD**)**; // Inicializamos los datos de la conexion

Serial.print**("**connecting**")**;

while **(**WiFi.status**()** **!=** WL\_CONNECTED**)** { // Si no hay conexion

Serial.print**("**.**")**;

WiFi.waitForConnectResult**()**; // Esperamos a que se obtenga un resultado de la conexion

**delay(**3500**)**;

if **(**WiFi.status**()** **!=** WL\_CONNECTED**)** { // Si sigue sin haber conexion

WiFi.reconnect**()**; // Intentamos reconectarnos

}

}

Serial.println**()**;

Serial.print**("**connected: **")**;

Serial.println**(**WiFi.localIP**())**;

}

void reconnectWifi**()** {

if **(**WiFi.status**()** **!=** WL\_CONNECTED**)** { // Si no hay conexion

Serial.println**("**reconnecting...**")**;

WiFi.reconnect**()**; // Intentamos reconectarnos

WiFi.waitForConnectResult**()**; // Esperamos a que se obtenga un resultado de la conexion

}

}

#### Funciones Firebase

void initializeFirebaseStream**()** {

stream **=** **[](**FirebaseStream stream**)** {

String eventType **=** stream.getEvent**()**;

eventType.toLowerCase**()**;

Serial.print**("**event: **")**;

Serial.println**(**eventType**)**;

if **(**eventType **==** **"**put**")** {

// Se inicializan los componentes en el primer GET que se hace a Firebase

if **(!**initialized**)** {

JsonObject& root **=** stream.getData**()**; // Se obtiene un JSON con todos los datos

initializeComponents**(**root**)**; // Se inicializan los componentes a partir del JSON

}

else {

String path **=** stream.getPath**()**; // Se obtiene la ruta donde se ha producido la modificacion

if **(**path **==** **"/**gas1**/**risk**")** { // Se obtiene el nuevo valor para el Gas Risk

//gas1\_risk = stream.getDataInt(); // El valor del gas lo actualiza el ESP32, por lo que se comenta esta linea para que no haya redundancia

Serial.println**("**Gas Risk: **"** **+** String**(**gas1\_risk**))**;

}

else if **(**path **==** **"/**rele1**/**state**")** { // Se obtiene el nuevo valor para el Rele1 State

light\_state **=** stream.getDataString**()**;

updateLight**()**;

Serial.println**("**Light State: **"** **+** light\_state**)**;

}

else if **(**path **==** **"/**rele2**/**state**")** { // Se obtiene el nuevo valor para el Rele2 State

plug\_state **=** stream.getDataString**()**;

updatePlug**()**;

Serial.println**("**Plug State: **"** **+** plug\_state**)**;

}

else if **(**path **==** **"/**temperature1**/**value**")** { // Se obtiene el nuevo valor para el Temperature Value

//temperature1\_value = stream.getDataFloat(); // El valor de la temperatura lo actualiza el ESP32, por lo que se comenta esta linea para que no haya redundancia

Serial.println**("**Temperature Value: **"** **+** String**(**temperature1\_value**))**;

}

else if **(**path **==** **"/**humidity1**/**value**")** { // Se obtiene el nuevo valor para el Humidity Value

//humidity1\_value = stream.getDataFloat(); // El valor de la humedad lo actualiza el ESP32, por lo que se comenta esta linea para que no haya redundancia

Serial.println**("**Humidity Value: **"** **+** String**(**humidity1\_value**))**;

}

}

}

};

}

void initializeFirebase**()** {

Firebase.begin**("**PROYECT\_URL**",** **"**AUTH**")**;

initializeFirebaseStream**()**; // Se inicializa el valor del stream

Firebase.stream**("/**devices**/**0x00000001**",** stream**)**; // Se inicia el stream de Firebase

initializeTimers**()**; // Se inicializan los timers despues de conectarse con Firebase

}

void initializeComponents**(**JsonObject& root**)** {

if **(**root.containsKey**("**gas1**"))** { // Se inicializa gas1

gas1\_risk **=** root**["**gas1**"]["**risk**"]**.as**<**int**>()**;

Serial.println**("**Initial Gas Risk: **"** **+** String**(**gas1\_risk**))**;

}

if **(**root.containsKey**("**rele1**"))** { // Se inicializa rele1

light\_state **=** root**["**rele1**"]["**state**"]**.as**<**String**>()**;

Serial.println**("**Initial Light State: **"** **+** light\_state**)**;

}

if **(**root.containsKey**("**rele2**"))** { // Se inicializa rele2

plug\_state **=** root**["**rele2**"]["**state**"]**.as**<**String**>()**;

Serial.println**("**Initial Plug State: **"** **+** plug\_state**)**;

}

if **(**root.containsKey**("**temperature1**"))** { // Se inicializa temperature1

temperature1\_value **=** root**["**temperature1**"]["**value**"]**.as**<**float**>()**;

Serial.println**("**Initial Temperature Value: **"** **+** String**(**temperature1\_value**))**;

}

if **(**root.containsKey**("**humidity1**"))** { // Se inicializa humidity1

humidity1\_value **=** root**["**humidity1**"]["**value**"]**.as**<**float**>()**;

Serial.println**("**Initial Humidity Value: **"** **+** String**(**humidity1\_value**))**;

}

initialized **=** true;

}

### Android