



**Universidad  
Europea**

LAUREATE INTERNATIONAL UNIVERSITIES

**UNIVERSIDAD EUROPEA DE MADRID**



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:

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

Debemos agradecer el apoyo que nos han prestado nuestros familiares y amigos, siendo un apoyo incondicional en el transcurso del desarrollo del proyecto. El apoyo tanto moral como económico ha permitido que pudiéramos sacar lo mejor de nosotros en todo momento, sirviendo de ayuda y referente en cuestiones básicas que nos planteamos, y aportando su experiencia propia como método de ayuda y comprensión de los problemas que íbamos a enfrentar durante la realización del proyecto.

Agradecemos por otro lado el esfuerzo e implicación por parte del cuerpo docente, no dejándonos dudas sin resolver en ningún momento, y ofreciéndonos su ayuda hasta cuando ellos mismos sabían que no iba a ser necesaria. Esperamos que los resultados de este trabajo estén a la altura de sus expectativas.

Por último, queremos agradecer a la Universidad Europea su implicación con el trabajo y el aprendizaje, así como las ayudas al estudio que ponen a nuestra disposición, valorando el esfuerzo y la actitud de los estudiantes, ya que sin todo ello, este proyecto probablemente no habría sido posible.



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
Planificación.....	18
Descripción del trabajo realizado .....	19
Resultados y validación .....	33
Conclusiones.....	41
Innovación .....	41
Trabajo futuro.....	42
Bibliografía y Webgrafía .....	43
Anexos .....	44
1. Apps domótica.....	44
2. Protocolos.....	54
3. Componentes .....	56
4. Código.....	60

## 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:

- 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](#))

**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](#))

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](#))

## 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](#).

## 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.



## Firestore

*Firestore* 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 *Firestore* 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](#)), 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

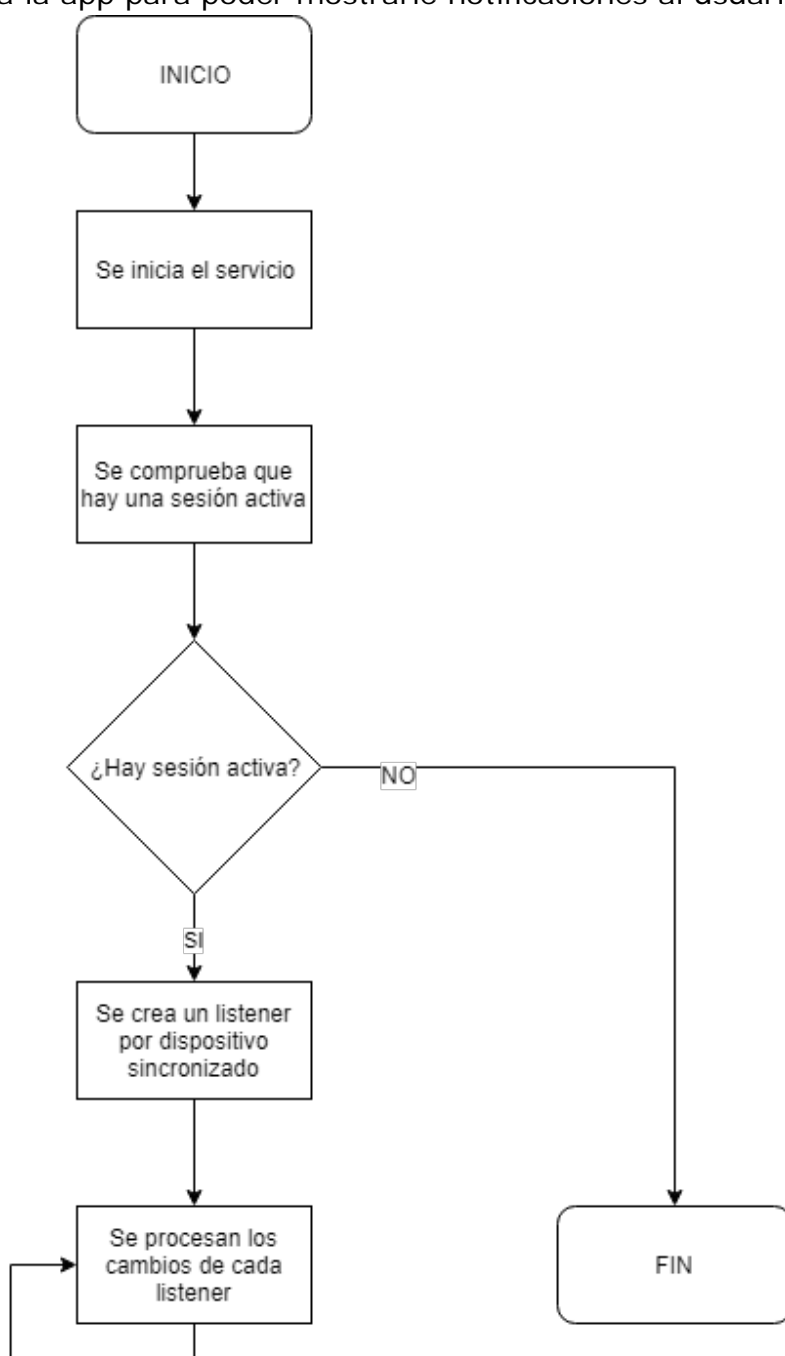
### Documentación técnica

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

### Diagramas de flujo

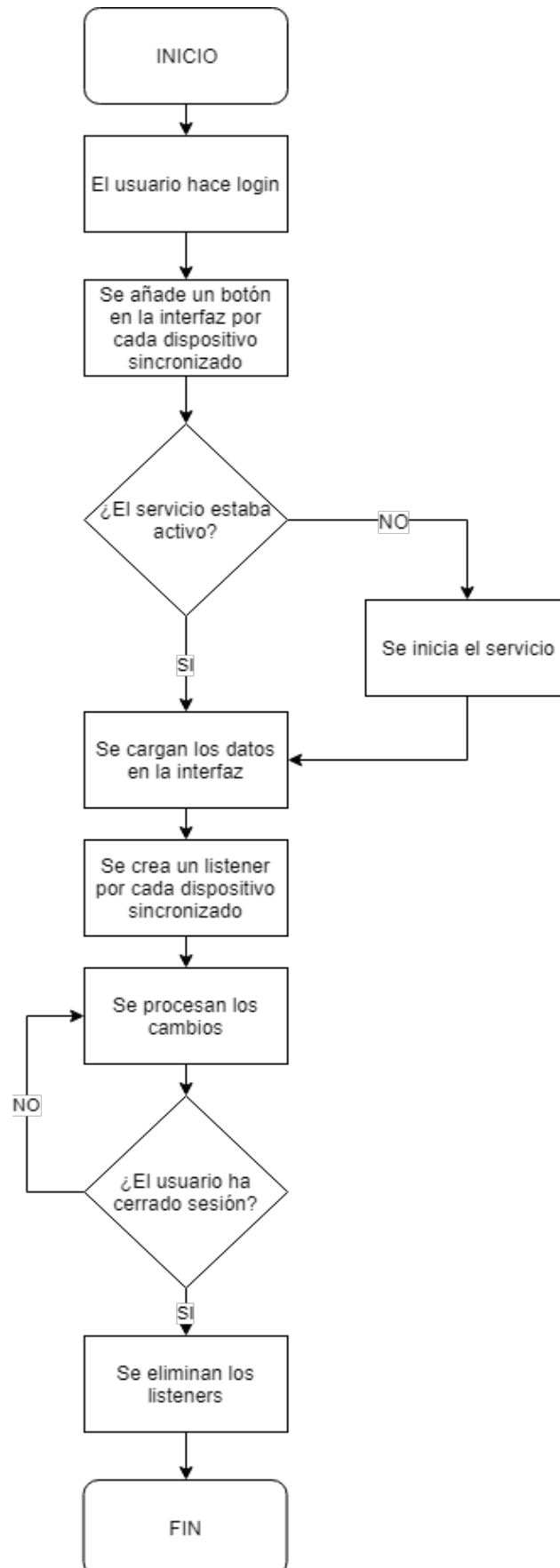
#### Flujo servicio Android

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.



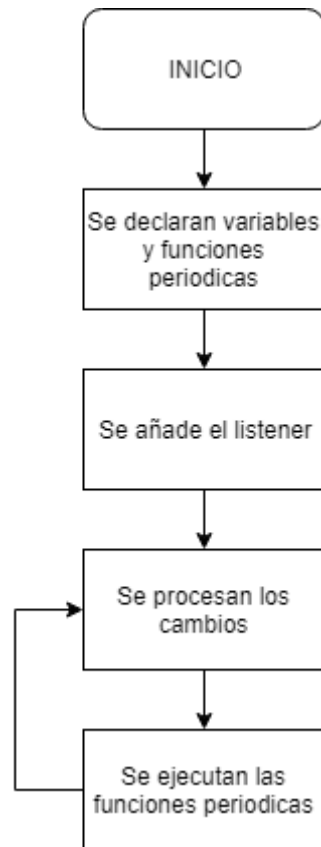
## Flujo aplicación Android

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**.



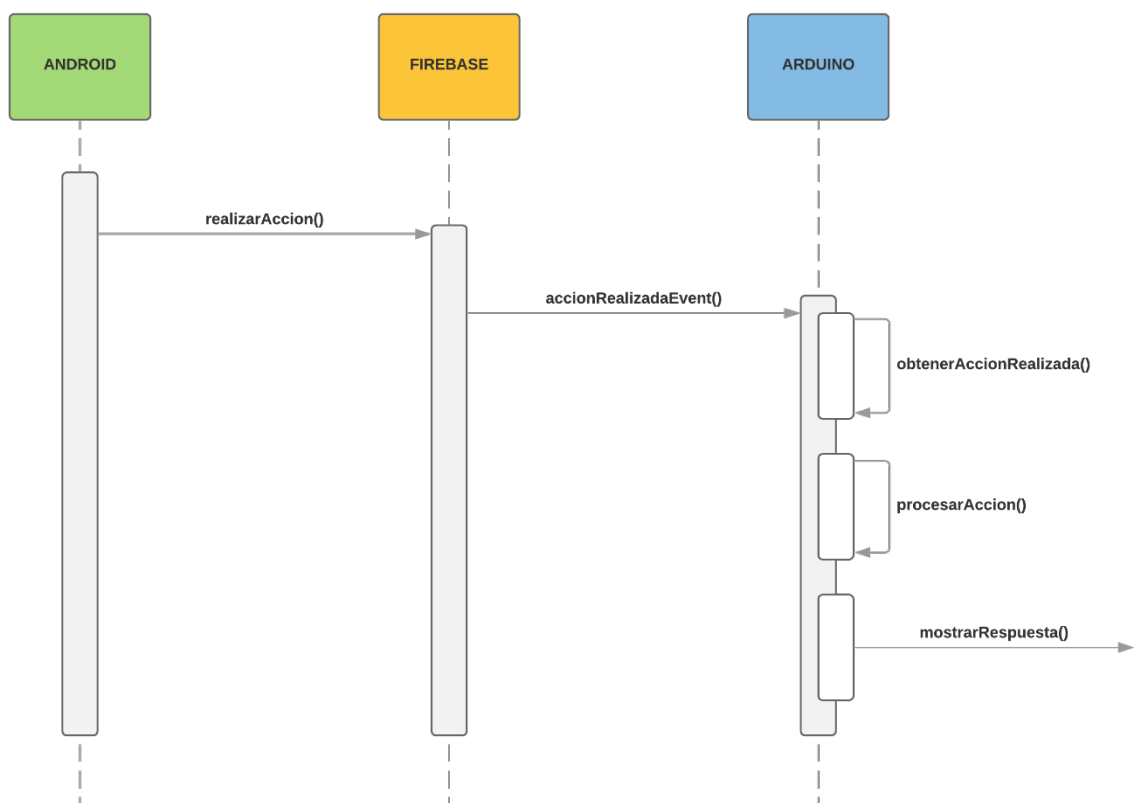
## Diagramas de secuencia

### 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.

#### ANDROID DIAGRAMA DE SECUENCIA

FUTH | May 30, 2018



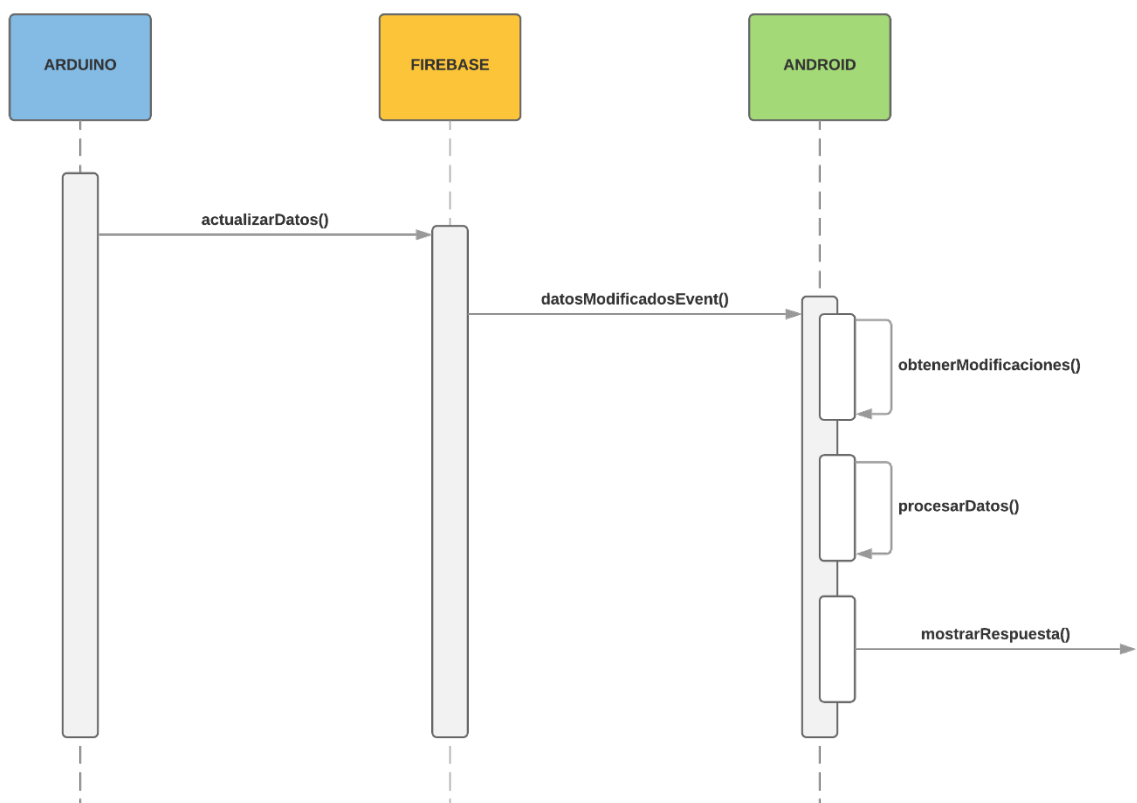
## Secuencia Arduino

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.

### ARDUINO DIAGRAMA DE SECUENCIA

FUTH | May 30, 2018



## 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](#)), por lo que nosotros seguiremos las directrices de **buenas prácticas** de dicho protocolo.

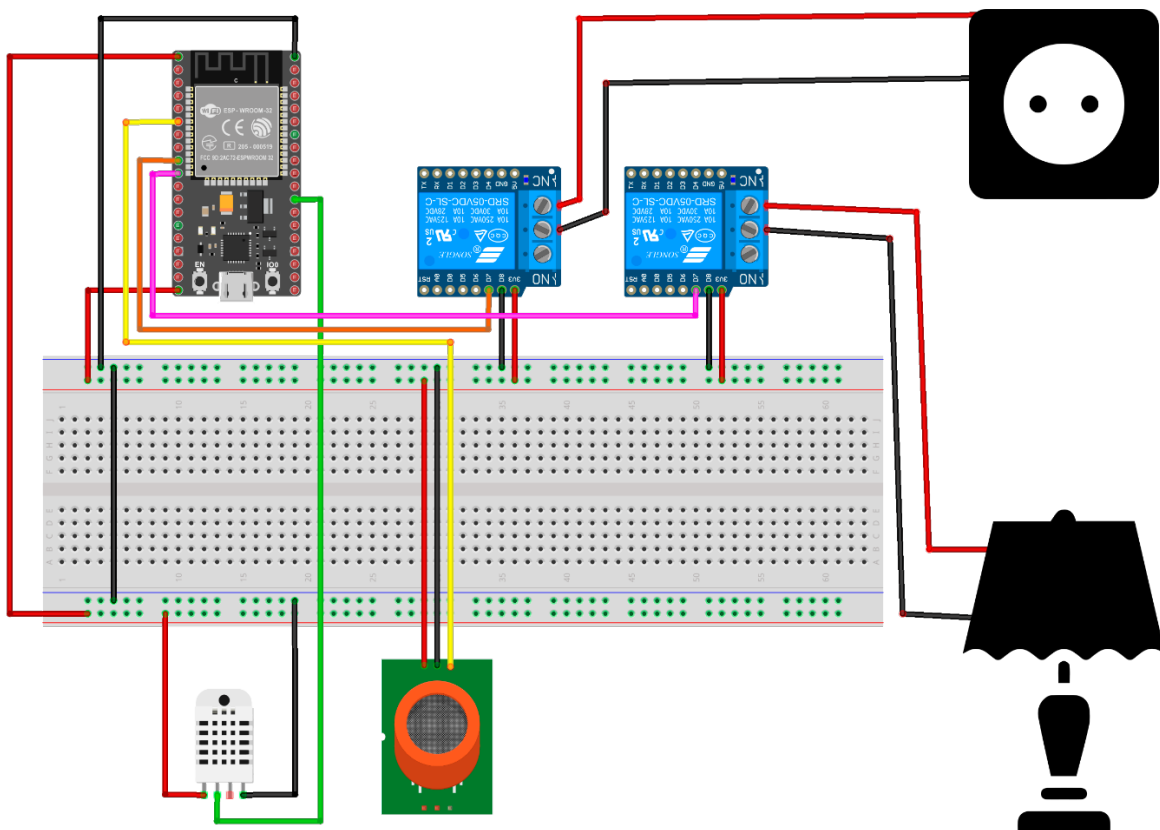




## Esquema de conexiones

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)



# 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 **Ox**, 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 **10<sup>8</sup> (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.

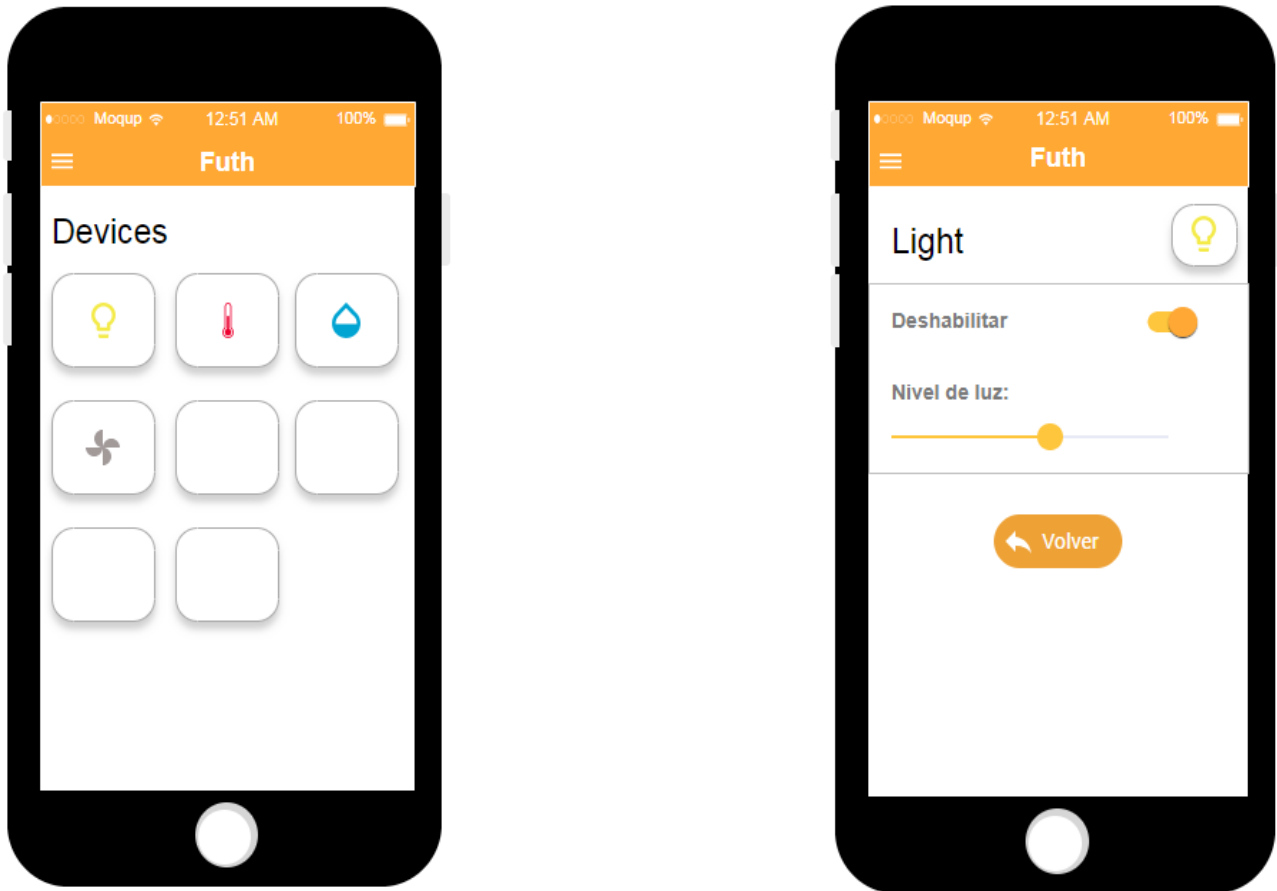
El servicio de notificaciones **solo debe** activarse si hay un usuario con la sesión iniciada, o en el momento de iniciar sesión, en caso contrario, **deberá** estar desactivado.

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

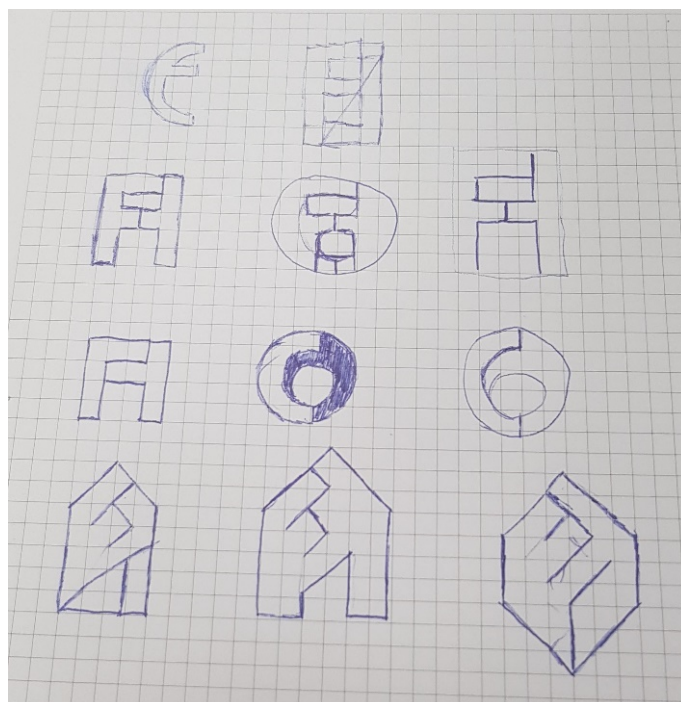
La aplicación **debe** tener una interfaz **clara y sencilla**, con **iconos representativos** de la acción que realizan para la fácil comprensión del usuario.

## 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.

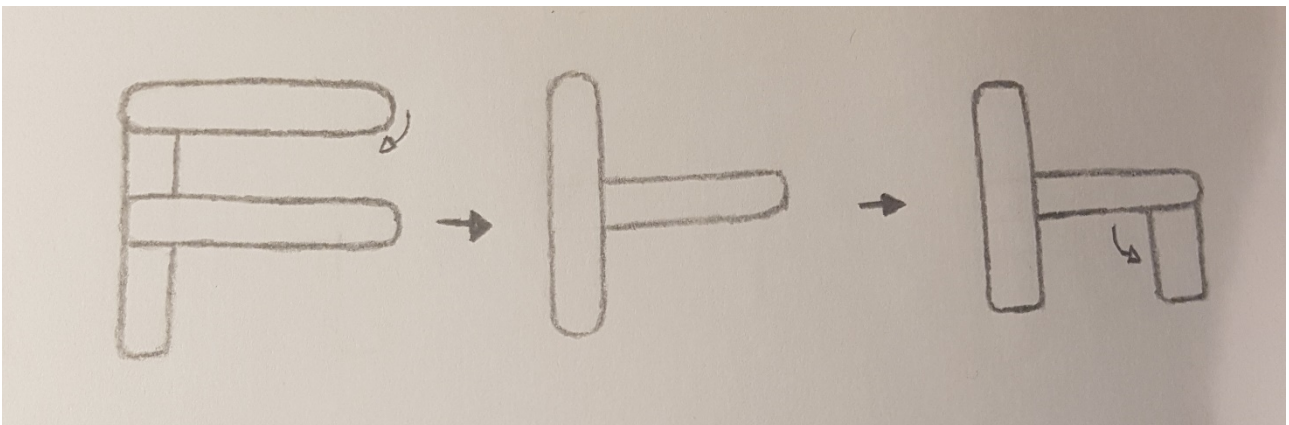


Finalmente tomamos la decisión en elegir un logo menos complejo, acorde con el estilo *material* de la app. Obtuvimos un logo estilo icono y otro de texto para aplicar en títulos y demás.

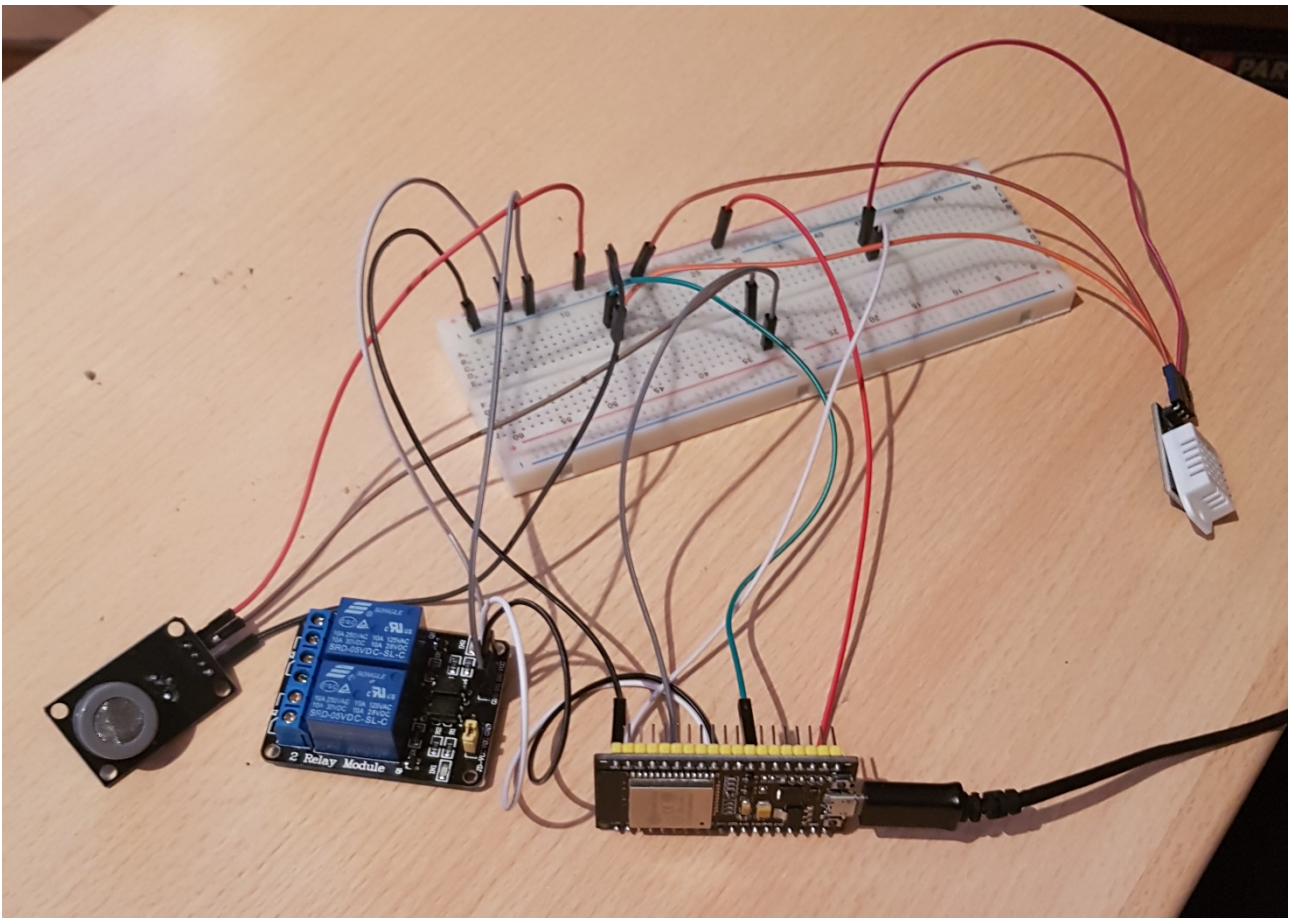
FUTH



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.

### 1. *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*.

### 2. *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.

### 3. *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](#))

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](#))

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](#))

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)

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)

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)

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)

## Desarrollo Android

La clase de *login* de la aplicación se encargará de comprobar si existe una sesión iniciada, y de ser así, nos llevará directamente a la pantalla de *home* cargando los datos del usuario en ella. Antes de ello se iniciará el splash screen, y una vez acabado se volverá a la ejecución normal de la app. (Ver Anexo 4.J.i)

Cuando el usuario haga clic en el botón de añadir, se le mostrará un *pop-up* en el que deberá introducir el **ID** de su dispositivo, una vez lo haga se comprobará que el usuario no se haya sincronizado con él anteriormente, y de ser así, lo añadirá como dispositivo sincronizado. (Ver Anexo 4.J.ii)

Una vez el usuario cierra la aplicación o la sesión, y vuelve a acceder, hemos diseñado una función que recupera automáticamente todos los dispositivos sincronizados que tiene y añade un botón en el menú lateral por cada uno de ellos. (Ver Anexo 4.J.iii)

Hemos creado un *package* de clases *helper*, en el que dispondremos de una clase dedicada a constantes de la app, y otra que facilite la tarea de imprimir el valor de los objetos que creamos por terminal. La segunda clase se creará únicamente con fines de *debug*.

Otro de los *package* que tendrá nuestro proyecto será el de ***model***, donde definiremos los objetos con los que la app deberá trabajar, basados también en la estructura que le hemos dado a los datos en *Firebase*.

Para la funcionalidad de las notificaciones hemos creado otro *package*, llamado ***notification***, destinado a albergar las clases necesarias para gestionar y mostrar las notificaciones de la forma más sencilla posible. En él hemos definido cuatro clases:

- **MessageReadReceiver:** Encargada de gestionar cuando una notificación ha sido leída. (Ver Anexo 4.J.iv)
- **MessageReplyReceiver:** Encargada de gestionar la respuesta que se reciba de la notificación (incluido de *Android Auto* y *Android Wear*). (Ver Anexo 4.J.v)

- **NotificationService:** Es la clase que se encarga de mostrar la notificación en base a los datos que recibe. (Ver Anexo 4.J.vi)
- **NotificationHelper:** Con esta clase invocamos a *NotificationService*, permitiéndonos operar con ella de una manera sencilla. Sin entrar en cálculos complicados, recibe los datos, y se los pasa a *NotificationService*, que se encarga de hacer el trabajo pesado. (Ver Anexo 4.J.vii)

Dada la naturaleza del diseño de nuestra app, y teniendo en cuenta que pretendemos mostrar una serie de **notificaciones personalizadas** al usuario en base a los datos que se procesan, hemos tenido que desarrollar un servicio en **segundo plano**, que esté constantemente monitorizando los sensores que el usuario tenga sincronizados; y que además, se ejecute **automáticamente** con el sistema. Para ello, hemos creado un *package* que se llama *service*, en el que se encuentran dos clases:

- **ServiceStartup:** Esta clase se ha registrado como receiver en el *AndroidManifest* de la app, lo que le permite **autoejecutarse** al iniciar el sistema (evento *BOOT\_COMPLETED*). Se encarga de iniciar *ServiceListener* una vez se ha iniciado el sistema. (Ver Anexo 4.J.viii)
- **ServiceListener:** Esta clase es básicamente el código que va a estar corriendo en **segundo plano** todo el rato monitorizando los datos de los sensores y mostrando las notificaciones que considere oportuna en base a ellos. (Ver Anexo 4.J.ix)

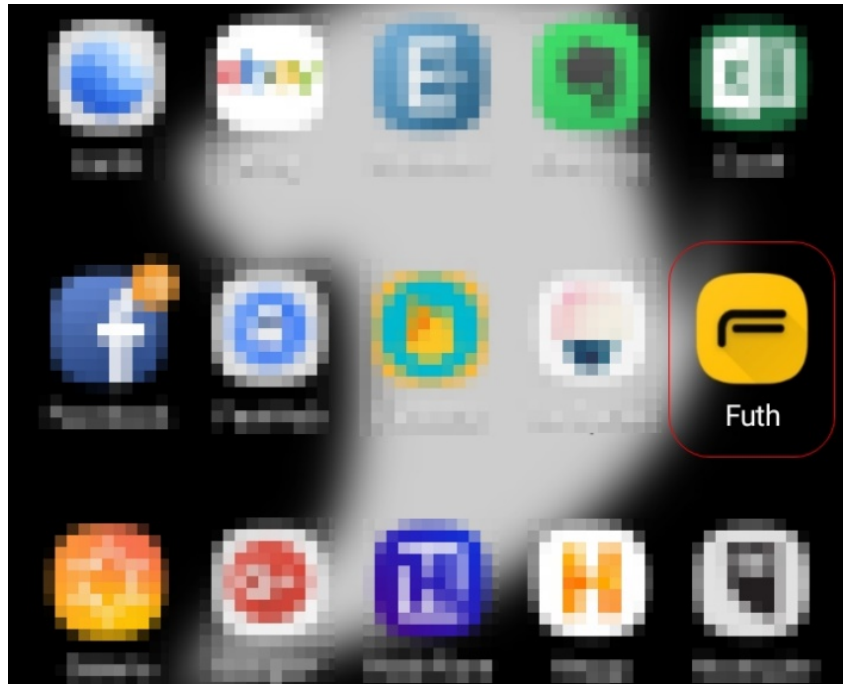
Para más información acerca de los permisos, *receivers* y servicios que hemos necesitado declarar en el *AndroidManifest* puedes consultar el Anexo 4.J.x.



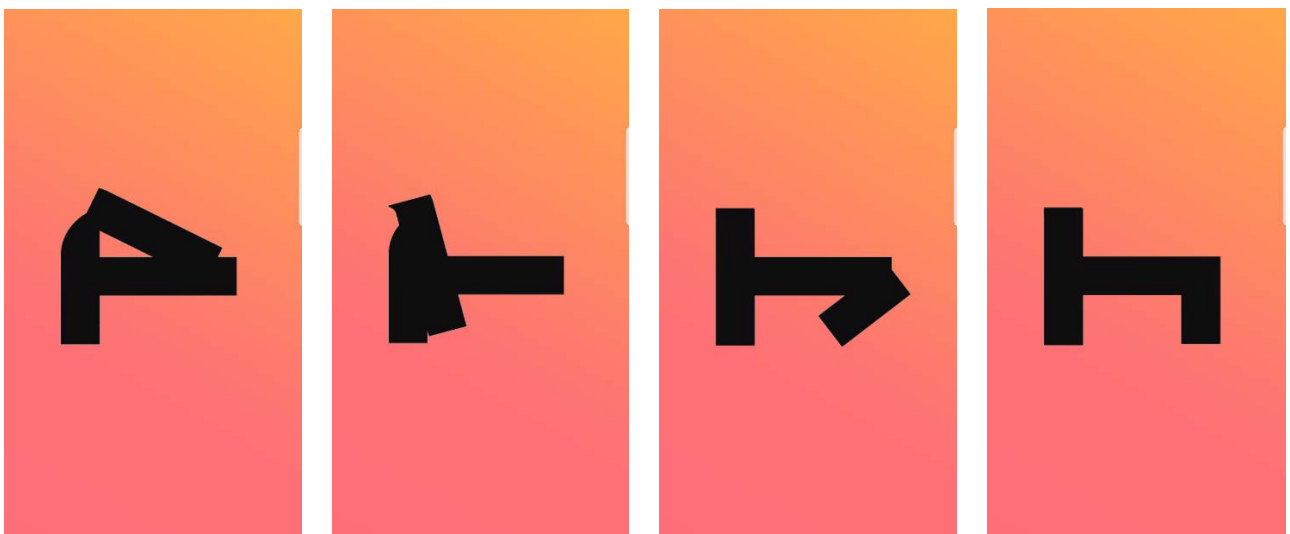
## Resultados y validación

En este apartado vamos a proceder a mostrar los **resultados obtenidos** del desarrollo del proyecto.

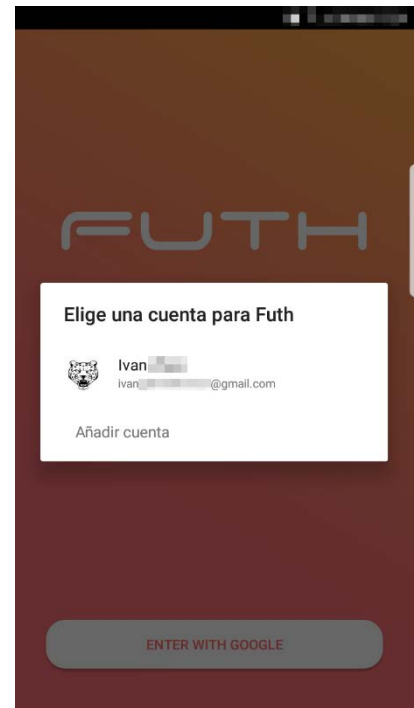
Una vez instalada la app en nuestro *smartphone* se vería así en el menú de aplicaciones.



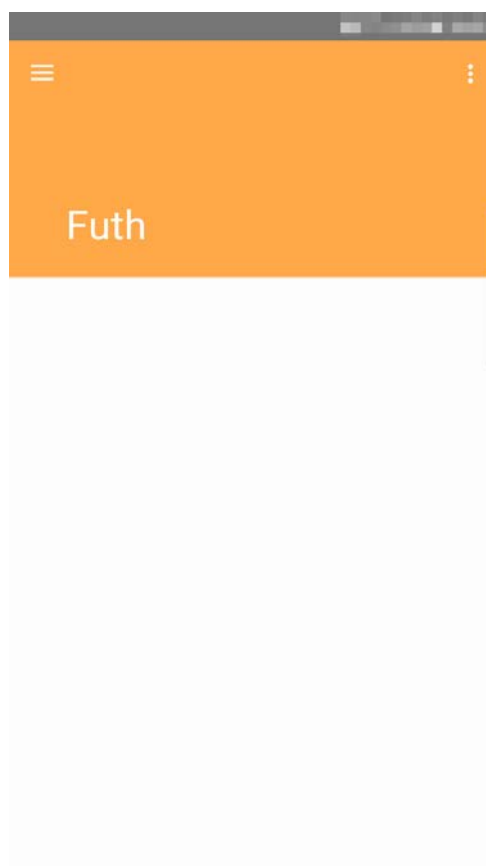
Al hacer clic sobre ella, se nos iniciará el *splash*, con la animación que hemos desarrollado a pantalla completa.



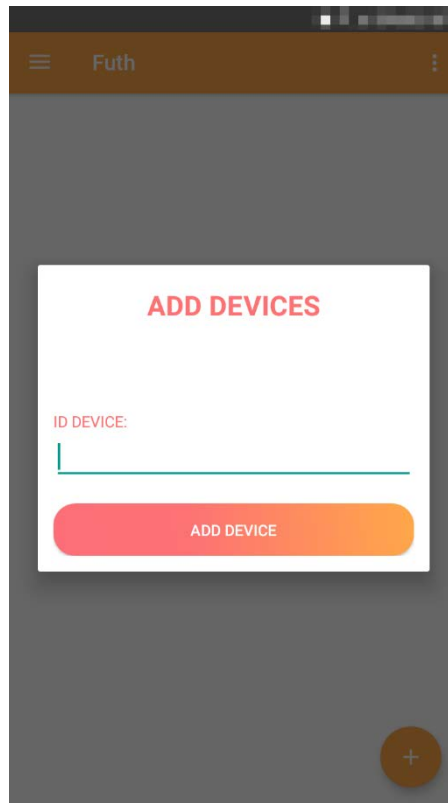
En la pantalla de *login* veremos el título de la aplicación con la fuente que elegimos para diseñar el icono de la aplicación. Una vez hagamos clic en el botón para **iniciar sesión** podremos elegir entre las cuentas que tenemos disponibles para ello.



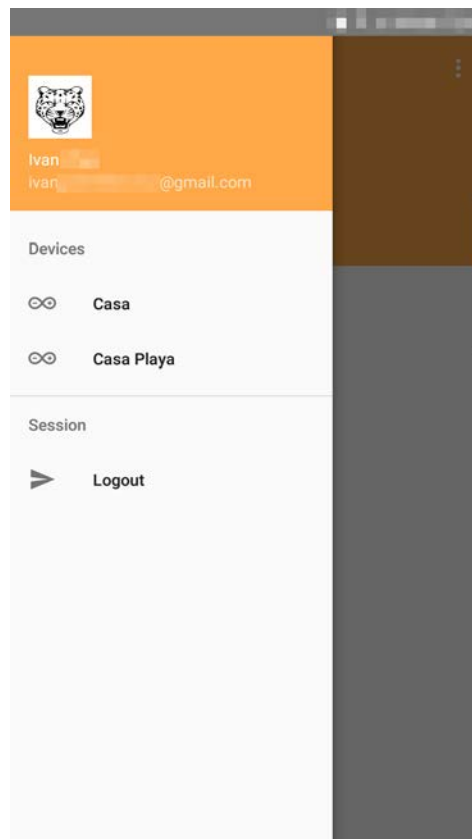
Seleccionamos la cuenta con la que vayamos a iniciar sesión, e inmediatamente después se nos redirigirá al *home* de la aplicación, por ahora **vacío** debido a que aún no se ha sincronizado ningún dispositivo.



Para comenzar a usar nuestro dispositivo debemos hacer *swipe-up*, en ese momento aparecerá un *float-button* con un símbolo de más. Al hacer clic sobre él se nos mostrará un diálogo donde deberemos introducir el *ID* único de nuestro dispositivo (que se le facilitaría al cliente en el momento de la compra junto con el producto).

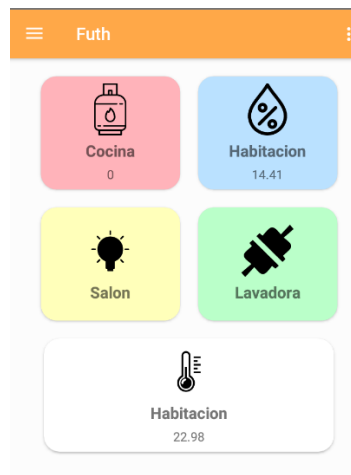


A medida que vayamos añadiendo más dispositivos, irán apareciendo en el menú lateral para poder acceder a sus funciones.

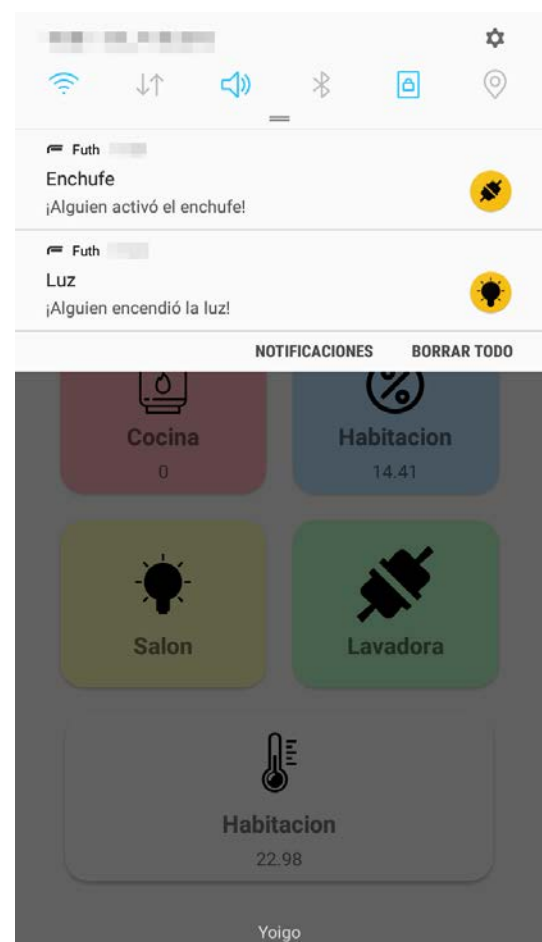
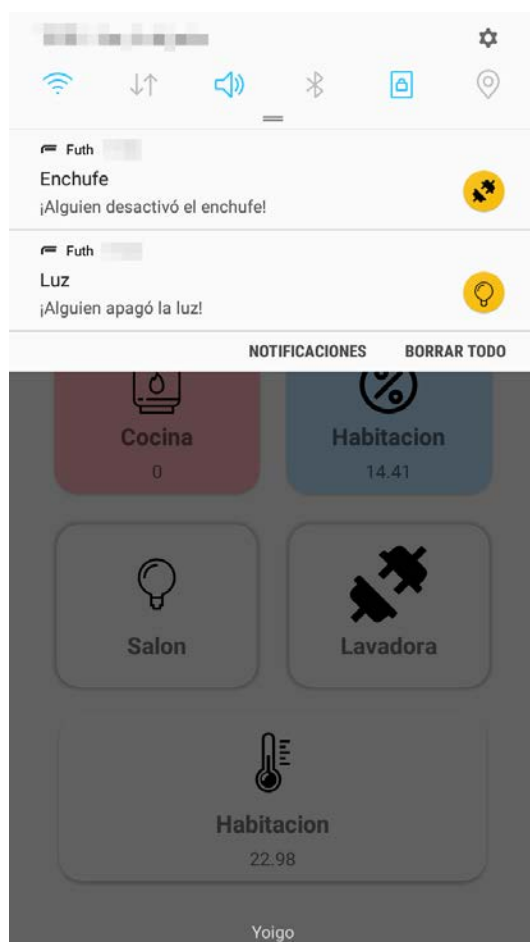


Al hacer clic sobre uno de los dispositivos que hemos sincronizado se nos abrirá el panel de configuración del mismo, en el que tendremos acceso a los datos de los sensores que tiene asociados.

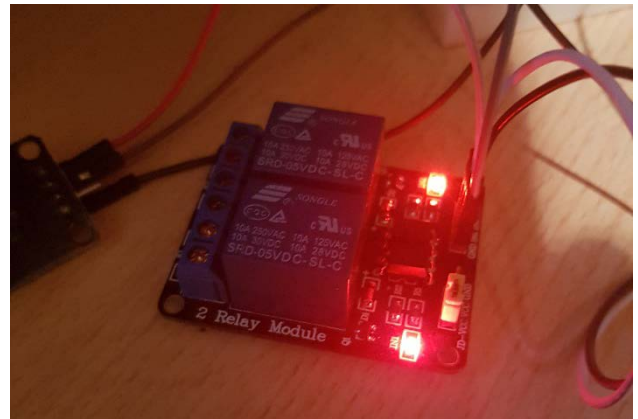
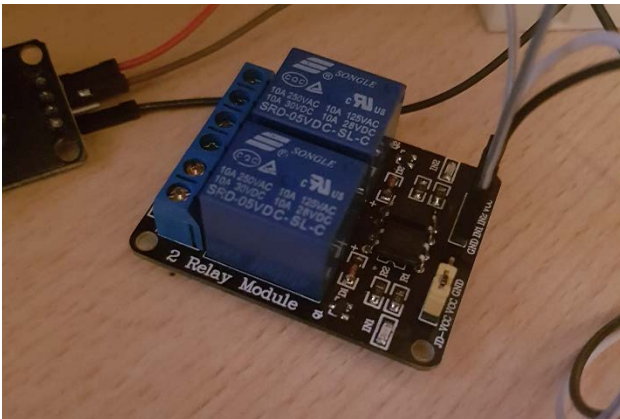
- **Rojo:** Sensor que mide el nivel de gas
- **Azul:** Sensor que mide el nivel de humedad
- **Amarillo:** Botón que permite encender y apagar una bombilla
- **Verde:** Botón que permite activar y desactivar un enchufe
- **Blanco:** Sensor que mide la temperatura



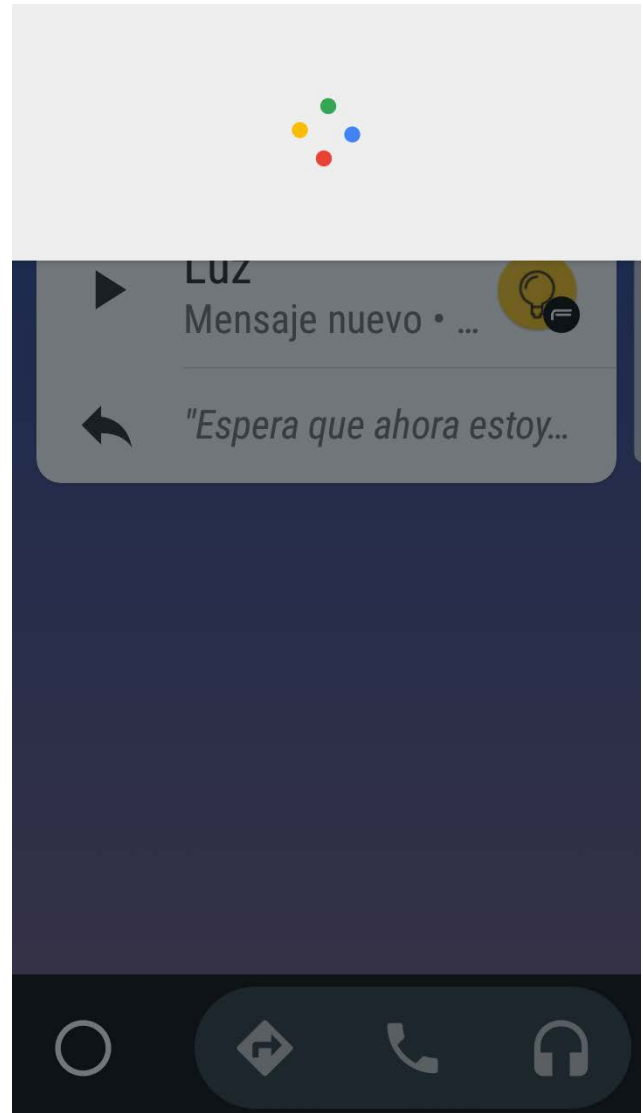
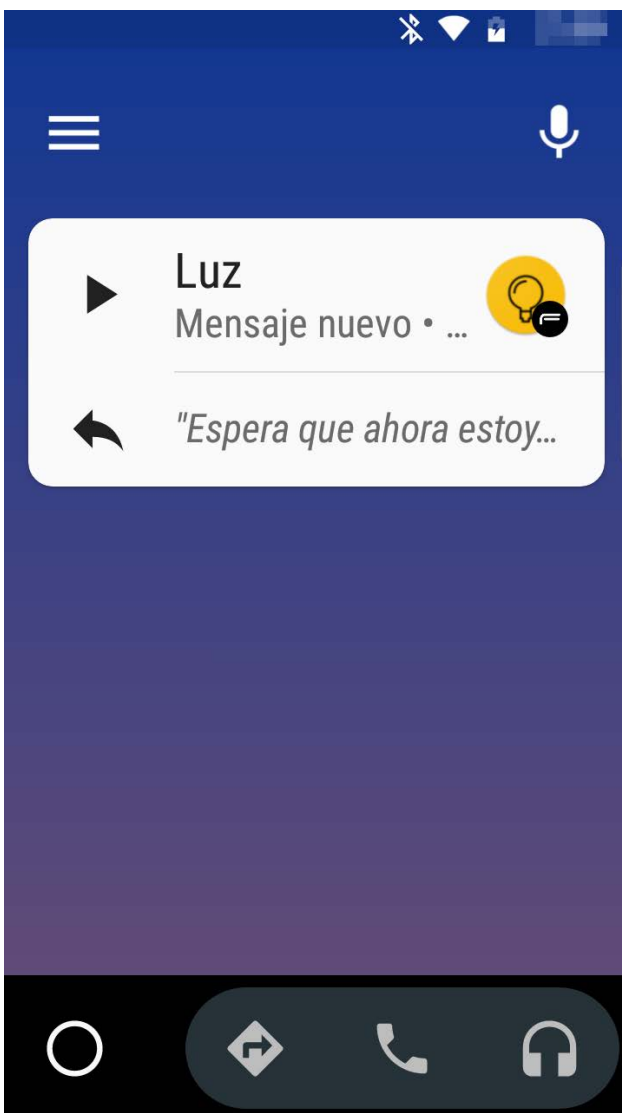
En el momento en que nosotros o alguien con el mismo dispositivo sincronizado **encienda/apague** y/o **active/desactive** la bombilla y el enchufe, recibiremos una notificación en nuestro móvil, y los iconos cambiarán de color e imagen.



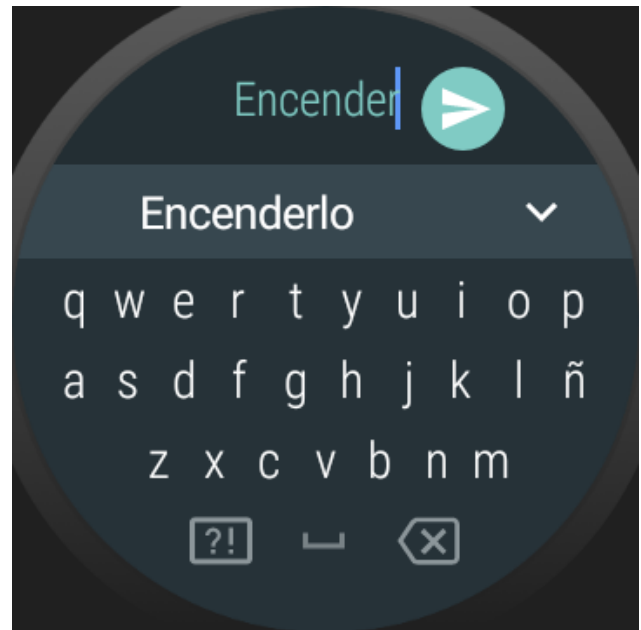
En nuestro dispositivo podremos comprobar como los **relés** de control de la bombilla y el enchufe se apagan y encienden conforme a las acciones que se realizan desde la aplicación.



Suponiendo que en el momento de recibir las notificaciones tenemos nuestro smartphone conectado con **Android Auto**, éste mostrará las notificaciones. Si hacemos clic sobre ella, el sistema nos la leerá en voz alta, y si le damos al icono del micrófono tendremos la opción de responder por voz (por ejemplo, un comando de voz implementado es *"Apaga la luz de nuevo"*, con el que la aplicación la apagará otra vez).



Si en lugar de estar emparejado con *Android Auto* lo está con un *smartwatch* que lleve **Android Wear**, éste mostrará las notificaciones de la misma manera, ofreciendo la posibilidad de responder por voz también.

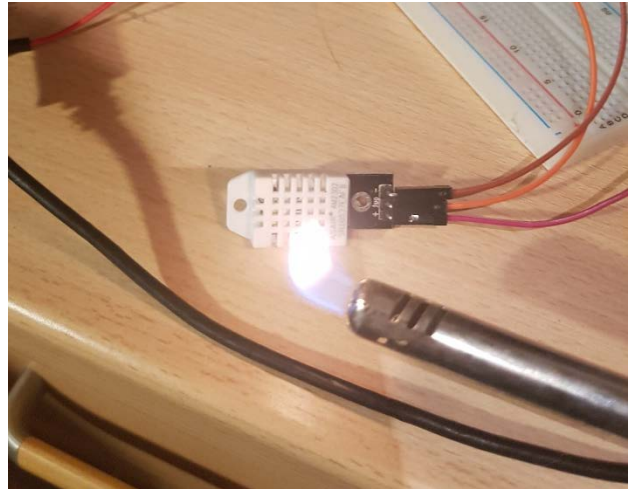
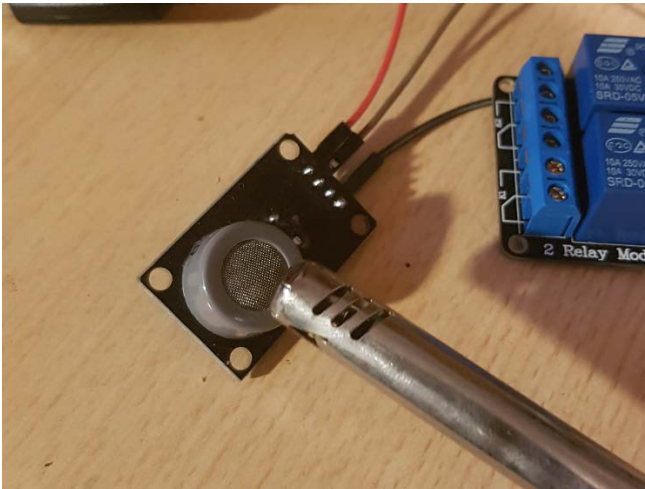


Hemos probado los sensores de nuestro dispositivo sometiéndolo a un test sencillo para comprobar que funciona correctamente.

- Inicialmente tenemos unos valores de **riesgo de gas 0** y de temperatura sobre los **23 grados**.



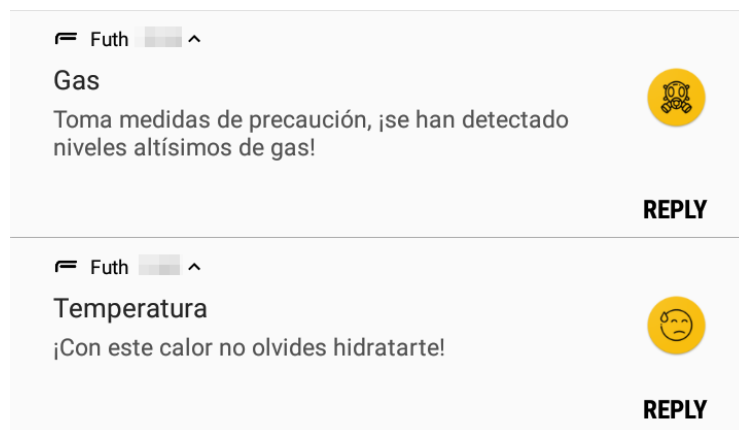
- Posteriormente **aplicamos gas** de un mechero al sensor de gas, y **acercamos la llama** del mismo un momento al sensor de temperatura.



- Podemos ver como los valores de los sensores dentro de la aplicación **han cambiado**.

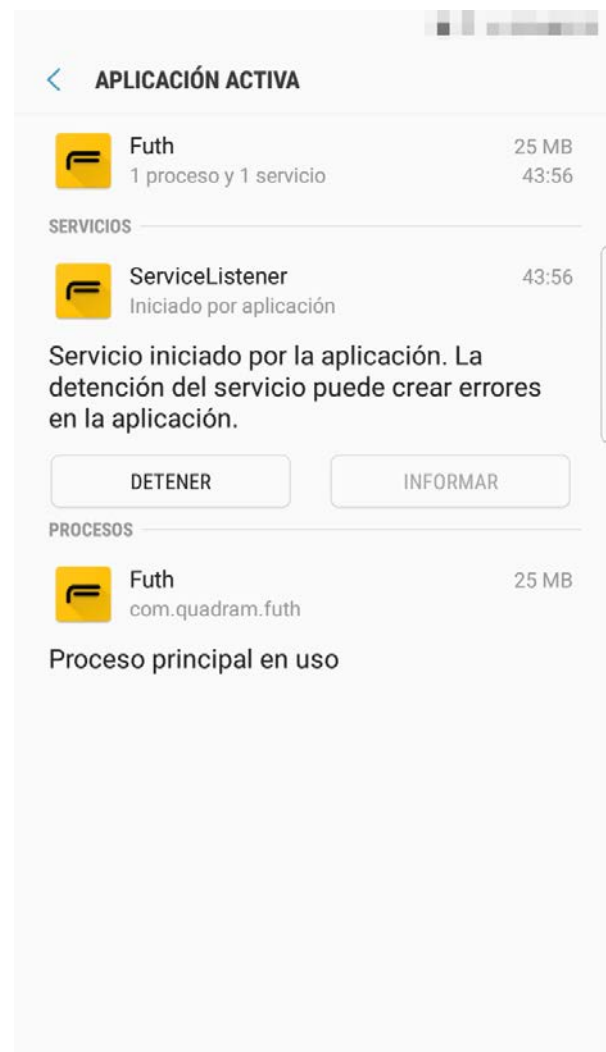
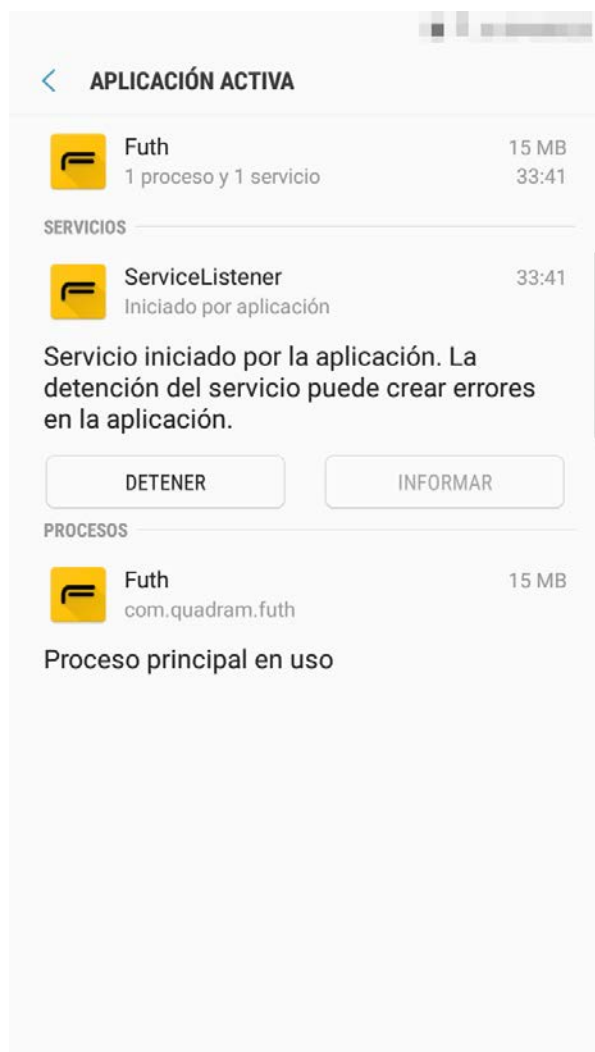


- Comprobamos que al poco tiempo la app recibe una **notificación de aviso** de ambos sensores debido al cambio que se ha registrado en los sensores.



A su vez, al haber planificado un diseño de la app **enfocado al rendimiento**, hemos intentado controlar al detalle la cantidad de recursos que tanto la app y el servicio de notificaciones consumen del sistema, estando realmente satisfechos con los resultados obtenidos.

A la izquierda los resultados del servicio corriendo *standalone*, y a la derecha los resultados del servicio corriendo junto con la aplicación abierta.





## Conclusiones

En este proyecto ambos integrantes hemos puesto a prueba nuestros conocimientos y habilidades adquiridas durante los dos años de duración del *CFGS de Desarrollo de Aplicaciones Multiplataforma*.

Debido a que los objetivos y metas que nos colocamos al elegir esta idea eran **muy exigentes**, hemos tenido que trabajar muy duro para conseguir una integración perfecta (o casi perfecta) tanto del software como del hardware que desarrollamos, haciendo uso de métodos y técnicas que no habíamos visto antes.

Hemos puesto a prueba nuestra **creatividad e imaginación** llevando más allá cada uno de los requisitos que establecimos en un principio, y no dándonos tiempo a realizar muchas otras implementaciones de las que nos hubiera gustado contar en este proyecto.

El **trabajo en equipo** y el **reparto de tareas** definitivamente ha sido crucial para poder cumplir los plazos en el tiempo que disponíamos. Siendo éste uno de los motivos del éxito de haber conseguido un proyecto a nuestros ojos satisfactorio y con buenos resultados, superando con creces los que esperábamos obtener en un principio.

## Innovación

Teniendo en cuenta el crecimiento exponencial en el cual se está desarrollando *Internet*, y la civilización a su alrededor, nosotros hemos pretendido "*seguir la moda*" de conseguir un producto que sea totalmente personalizable según las necesidades del usuario, como se está viendo cada vez más en las grandes empresas.

Hemos desarrollado un servicio que se ejecuta en **segundo plano**, y que no consume apenas recursos del sistema, para que el usuario esté constantemente enterado de los datos que generan sus sensores. Hemos establecido una serie de rangos en base a los valores que se obtienen, y en base a ellos se le facilitan una serie de notificaciones **amigables y desenfadadas** al usuario.

Lo realmente novedoso de este servicio es que hemos dado soporte a **Android Auto** y **Android Wear**, por lo que todos los usuarios de nuestra app que tengan al menos un dispositivo sincronizado, podrán recibir las notificaciones en su *smartwatch* o en su coche mientras conducen. Además, hemos dado soporte a respuesta mediante **comandos de voz**, lo que supone una gran comodidad y evita distracciones al volante. En el caso de *Android Auto*, es necesario destacar que al pulsar sobre la notificación, el propio sistema la lee en voz alta por los altavoces del coche, evita la distracción de tener que leerla uno mismo.

Otra novedad que hemos introducido, es que permitimos al usuario guardar un **sinfín de dispositivos**, pudiendo tener uno para la casa, otro del trabajo, otro de la finca, y todos los que quiera. Y facilitamos el acceso a ellos al no requerir de los típicos registros interminables que existen en otras aplicaciones o página web. En el momento en el que el usuario está usando nuestra app, se asume que la ha descargado del *Play Store*, con su cuenta de *Gmail* (obligatorio tenerla para poder descargar apps), y con esa misma u

otra cuenta de *Gmail* que tenga asociada a su dispositivo de procederá a iniciar sesión (registrando automáticamente al usuario si no lo estaba).

Mediante la introducción de **relés** en nuestro proyecto, otra novedad que pretendemos mostrar es que sería posible actualizar la instalación de una vivienda antigua para integrarla con un dispositivo como el que hemos desarrollado, sin necesidad de hacer obras, cambios de potencia, ni cosas por el estilo.

El diseño que hemos desarrollado permite al dispositivo no depender de un solo usuario, y viceversa, consiguiendo una **relación N:N**. Un dispositivo puede estar sincronizado por ninguno o muchos usuarios, y un usuario puede tener sincronizado ninguno o muchos dispositivos. Esto, sumado al servicio de notificaciones, permite que múltiples *smartphones* con distintos usuarios reciban las notificaciones de un mismo dispositivo, y que un único *smartphone* con un solo usuario, reciba las notificaciones de múltiples dispositivos.

## Trabajo futuro

En este proyecto principalmente se pretende evaluar de qué manera se puede mejorar la calidad de vida de las personas mediante la tecnología y el uso de las herramientas que disponemos hoy a nuestro alcance. Siendo ésta una visión "*pequeña*", al no tener en cuenta el gran número de ámbitos y sectores a los que se podría aplicar la **domótica** y el **hardware personalizado** según necesidades para mejorar la calidad de vida de las personas, productividad de empleados, accesibilidad a personas con discapacidad, seguridad, confort y comunicación.

En un futuro se podría dar soporte a otras plataformas como *Apple* u ordenadores de sobremesa, ampliando así el público objetivo del producto.

También se pretende implementar funcionalidades que permiten al usuario elegir un **nombre personalizado** del dispositivo que sincronizan, como también la opción de **eliminarlo** si ya no desean estar sincronizados más con él.

El diseño que se ha elaborado no permite al servicio ejecutarse en segundo plano en *smartphones* con *Android* igual o superior a *Oreo* sin que la aplicación esté abierta, debido a los nuevos límites de ejecución en *background* que se han impuesto a partir de estas versiones, por lo que la implementación de dicho servicio en los terminales que cumplan esa condición también forma parte del trabajo futuro.

Dar la opción **drag & drop** sobre los sensores dentro de la aplicación, para que el usuario los acomode a su gusto, y/o los elimine si lo desea, es otra característica que formaría parte de desarrollos posteriores.

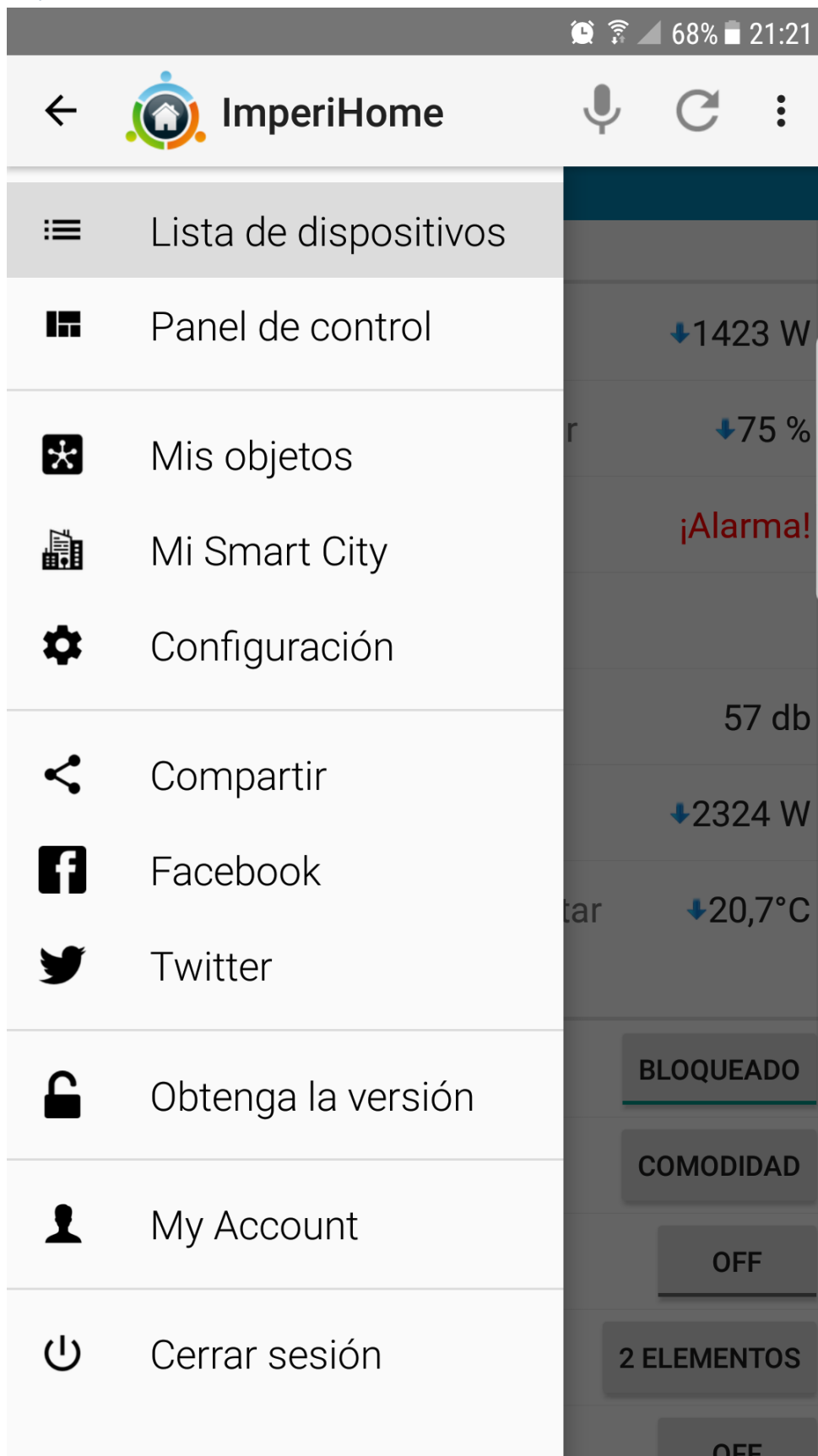
## Bibliografía y Webgrafía

- [Arduino](#)
- [¿Qué es Firebase?](#)
- [Productos Firebase](#)
- [Firebase](#)
- [¿Qué es MQTT?](#)
- [MQTT](#)
- [C++ Wikipedia](#)
- [Arduino Wikipedia](#)
- [¿Cómo funciona un relé?](#)
- [Sensores MQ](#)
- [Sensor MQ9](#)
- [Sensor DHT22](#)
- [¿Qué es el ESP32?](#)

## Anexos








### 1. Apps domótica

#### A. Imperihome








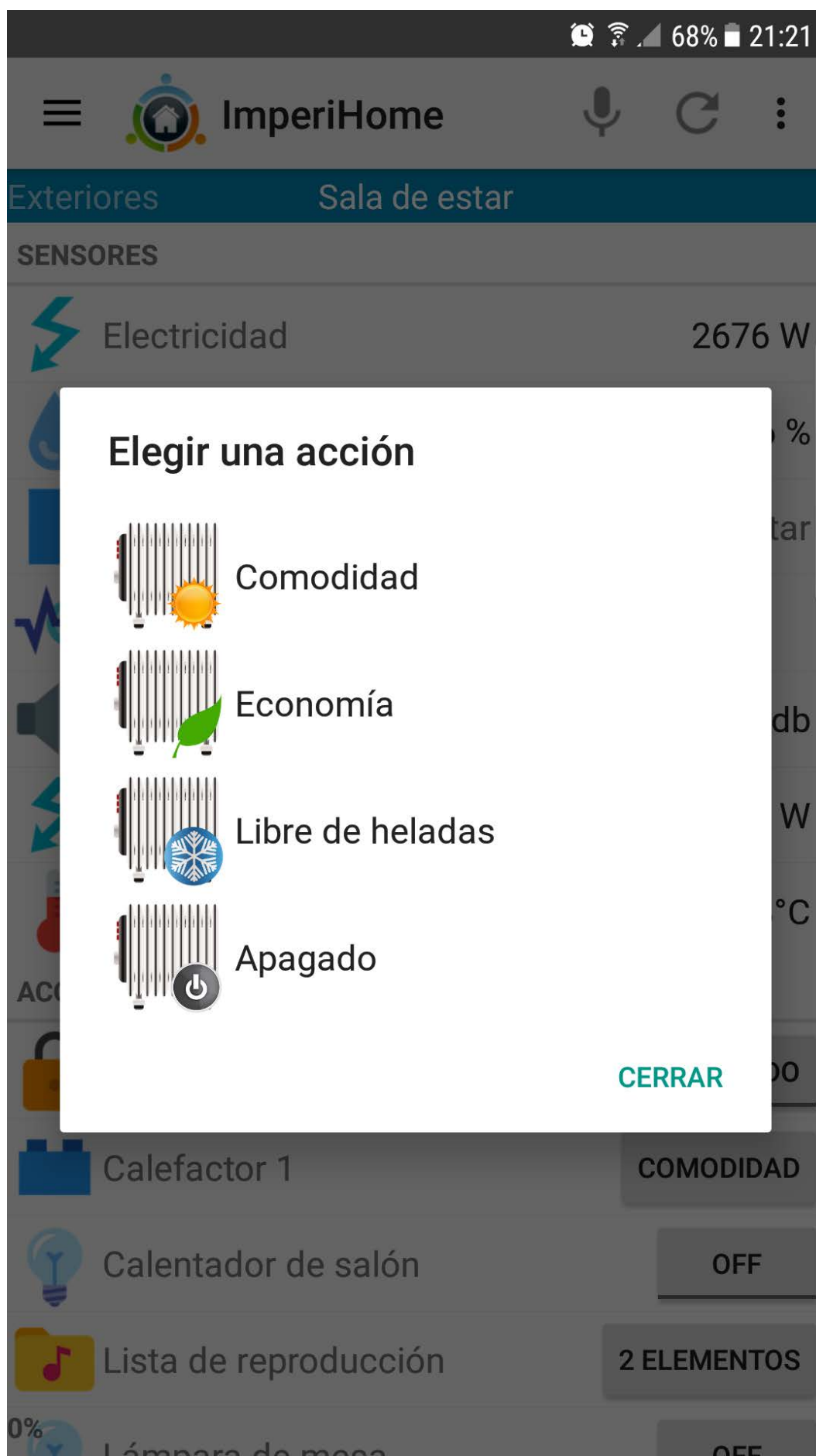
Exteriores Sala de estar

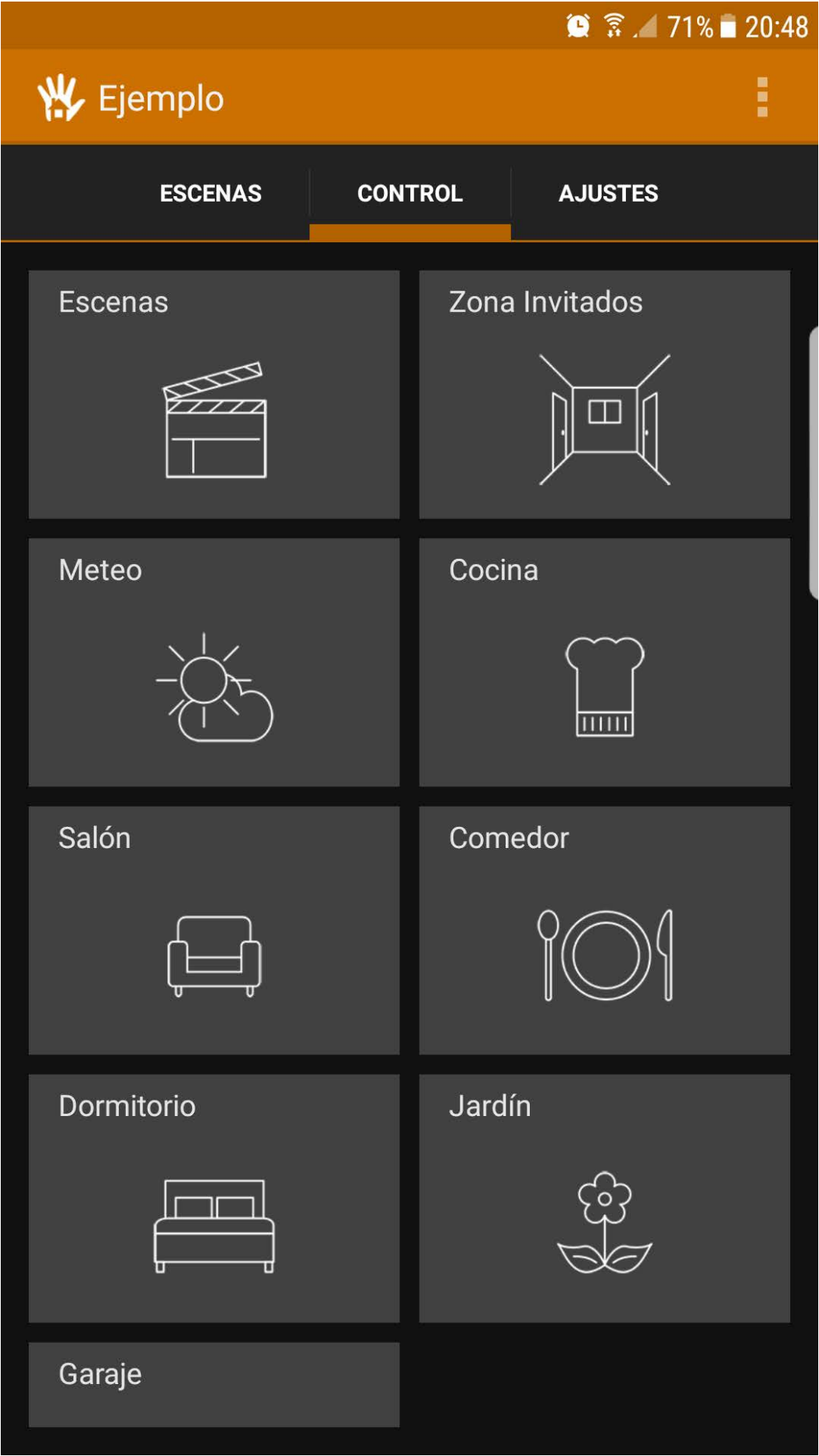
SENSORES

	Electricidad	2473 W
	Higrografía de la sala de estar	75 %
	Puerta delantera	¡Alarma!
	Rocker Switch	
	Ruido	54 db
	Teleinformación	2104 W
	Temperatura de la sala de estar	20,1°C

ACCIONADORES

	Bloquear puerta	DESBLOQUEADO
	Calefactor 1	COMODIDAD
	Calentador de salón	OFF
	Lista de reproducción	2 ELEMENTOS
	Lámpara de mesa	OFF





## Apagado General



## Bajar Persianas



## Simulador Presencia



## Escena KNX

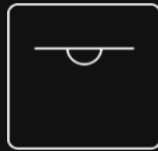




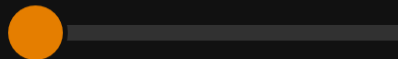
## Temperatura

---

## Luces



## Sofá

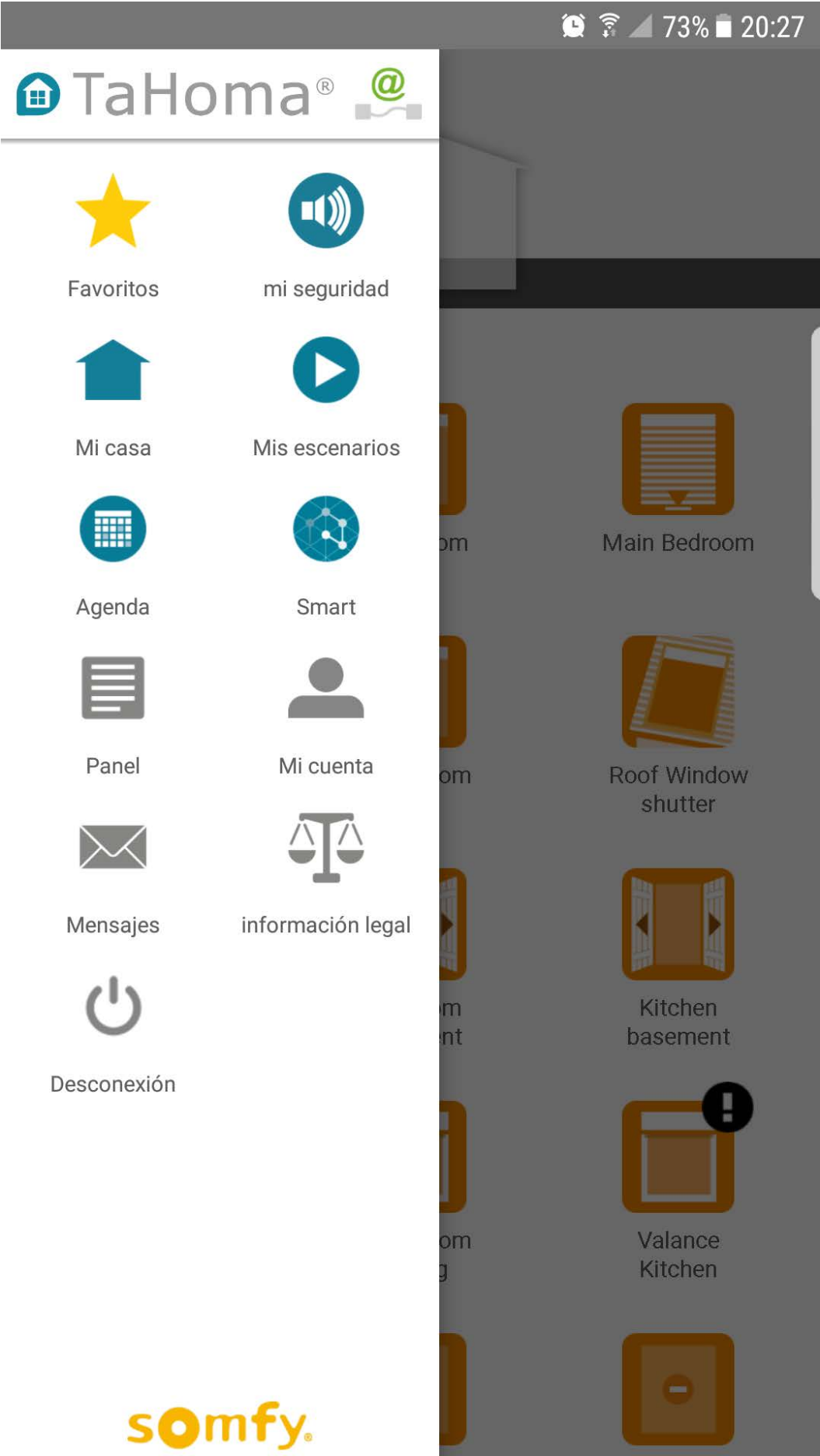


## Persiana Lateral



## Persiana Central

C. TaHoma by Somfy





Persianas (5)



East Room



Main Bedroom



TV Wall



West Room



Roof Window  
shutter



Contra-  
ventanas



Bathroom  
basement



Kitchen  
basement



Estores de  
exterior (2)



Living room  
awning



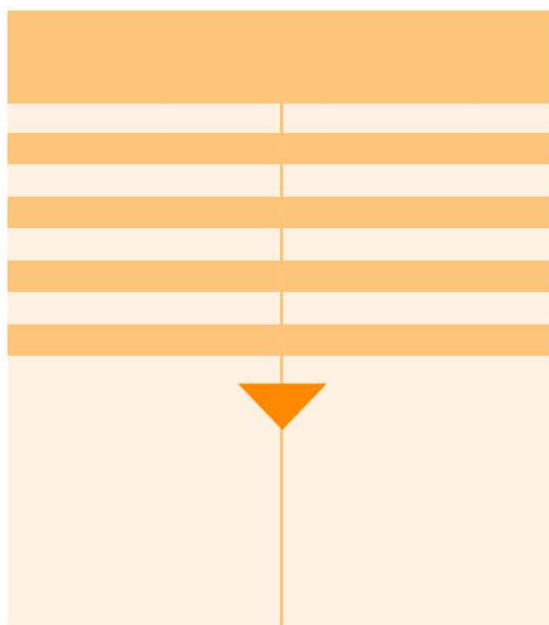
Valance  
Kitchen



## MAIN BEDROOM



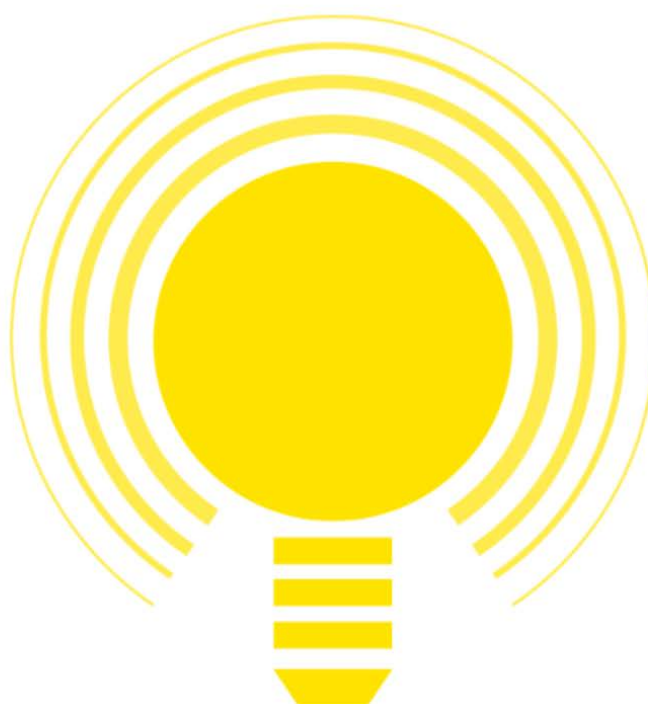
discreto



OK

CANCELAR

## FRONT LIGHT



5'

Temporizador



OK

CANCELAR

Living Room

East

South

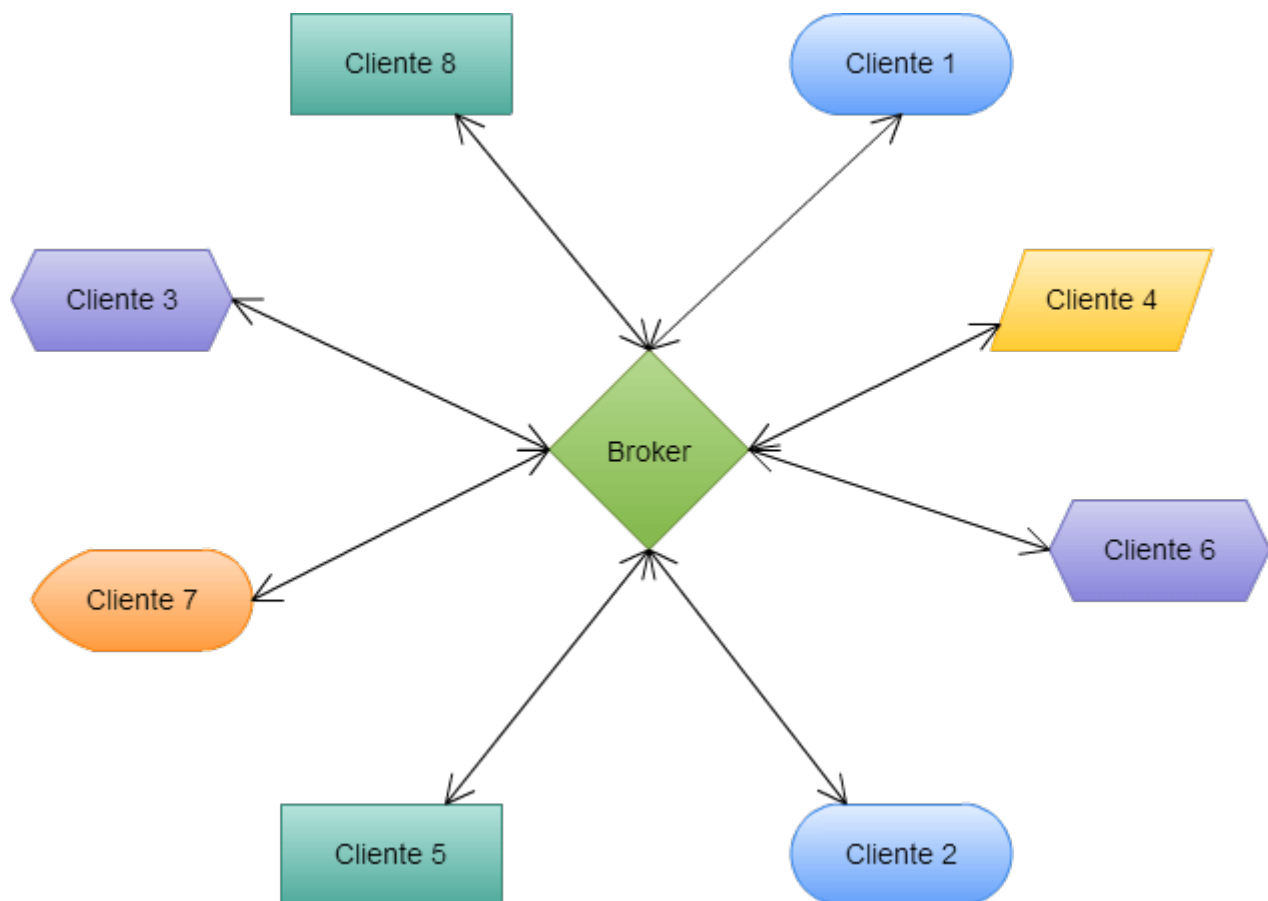
## 2. Protocolos

### D. 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.

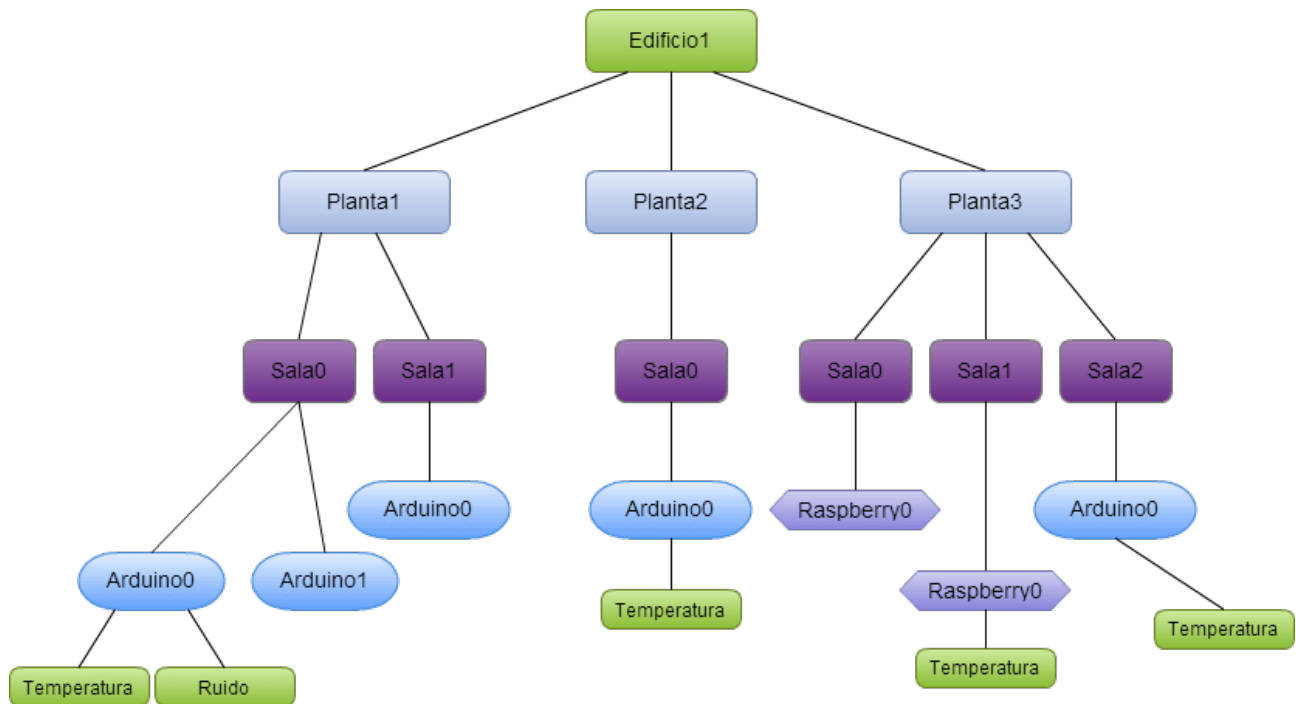
La arquitectura de MQTT sigue una topología de estrella, con un nodo central que hace de servidor o *bróker* con una capacidad de hasta 10000 clientes. El bróker es el encargado de gestionar la red y de transmitir los mensajes, para mantener activo el canal, los clientes mandan periódicamente un paquete *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.



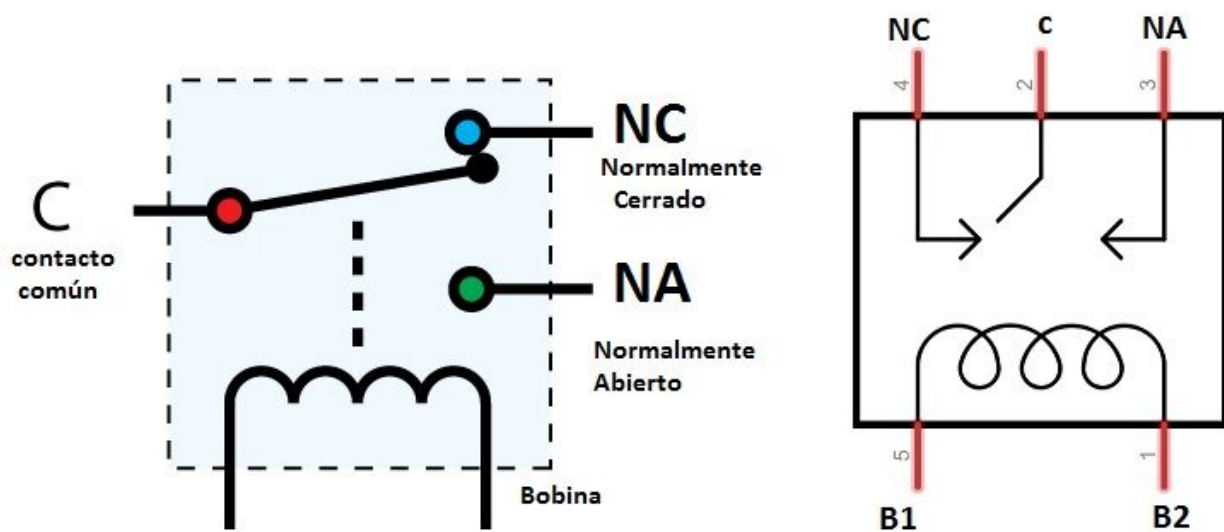
Así, un nodo puede suscribirse a un topic concreto ("edificio3/planta0/sala3/arduino4/humedad") o a varios a la vez ("edificio2/planta5/#").

### 3. Componentes

#### E. 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.



**Al meter corriente por la bobina los contactos abiertos se cierran y los cerrados se abren.**

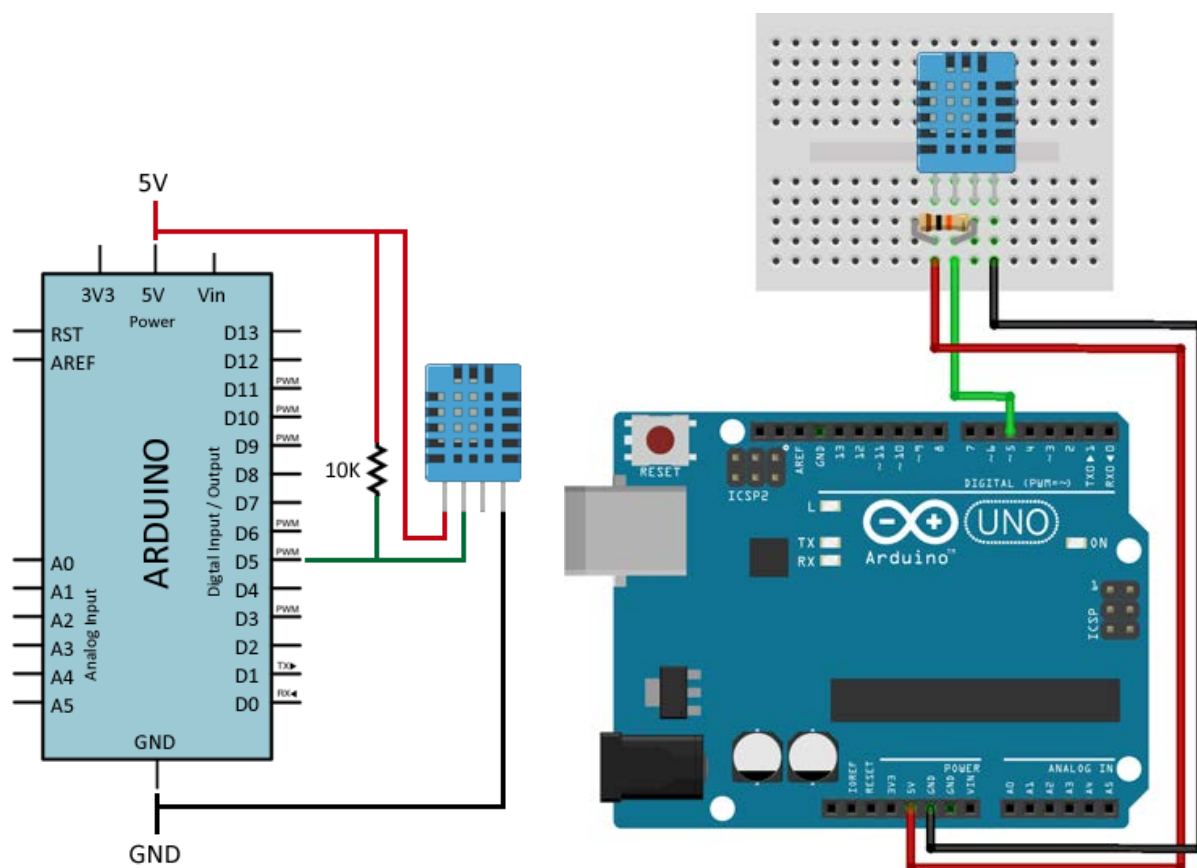


## F. Sensor DHT22

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:

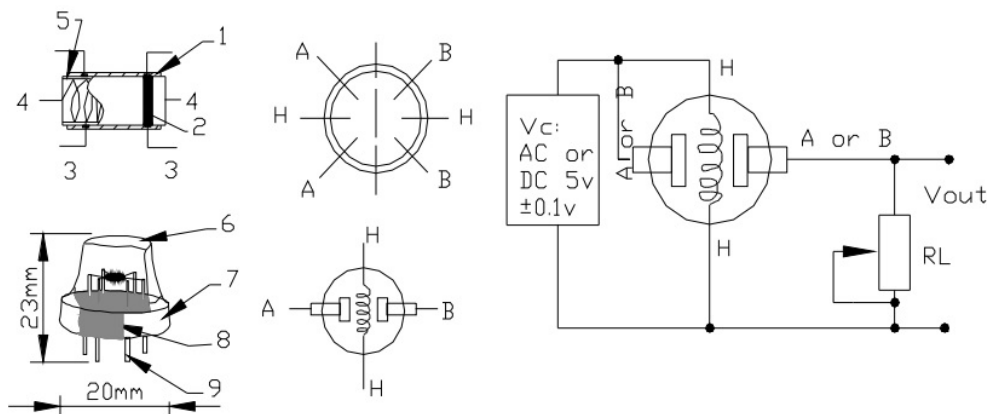
- Medición de temperatura entre -40 y 125, con una precisión de 0.5 grados Celsius.
- Medición de humedad entre 0 y 100, con una precisión entre el 2 y el 5 por ciento.
- Frecuencia de muestreo de 2 mediciones por segundo (2 Hz)

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



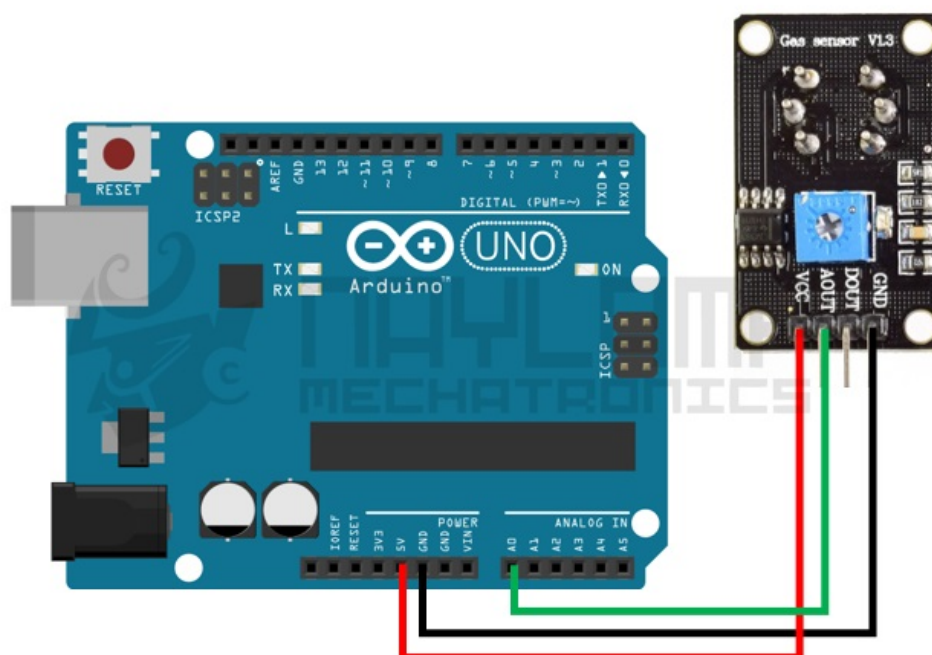
## G. Sensor MQ9

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 -10°C hasta 50°C.
- Consumo menor a 150mV con corriente de 5V.

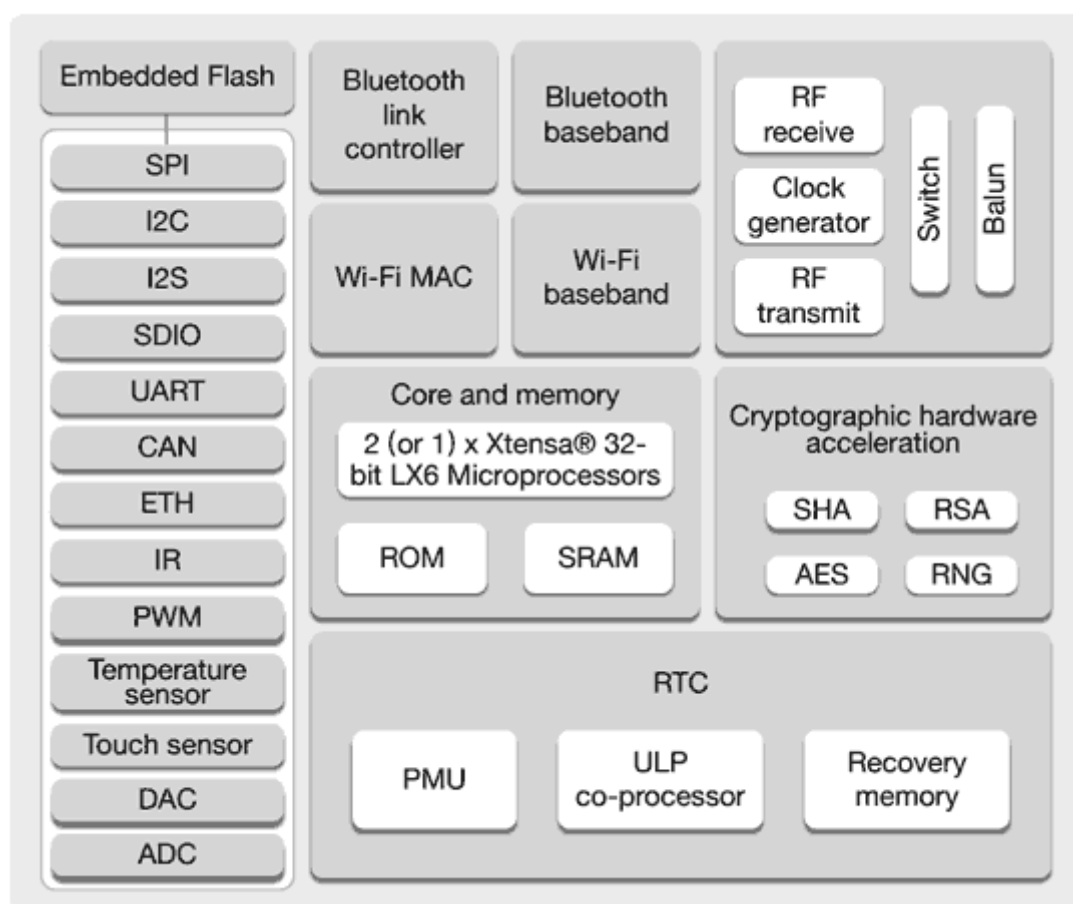


## H. Microcontrolador ESP32

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



## 4. Código

### I. Arduino

#### *i. 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;
```

## ii. 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
}
```

## iii. Función loop

```
void loop() {
  t.run();
}
```

## iv. 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);
    }
}

```

## v. 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);
}

```

## vi. Funciones Wifi

```

void initializeWifi() {
    WiFi.setAutoReconnect(true); // Establecemos que si se pueda usar la funcion
    de reconectar 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
    }
}

```

## vii. 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;
}

```

## J. Android

### *i. Login y Main*

```

if (!isServiceRunning(ServiceListener.class)) {
    Intent i = new Intent(this, ServiceListener.class);
    startService(i);
}

startSplash();

firebaseAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
        FirebaseUser user = firebaseAuth.getCurrentUser();
        if (user != null) {
            setUserData(user);
        } else {
            goLoginActivity();
        }
    }
};

private void setUserData(FirebaseUser user) {
    txvNameGoogle.setText(user.getDisplayName());
    txvGmail.setText(user.getEmail());
    Glide.with(this)
        .load(user.getPhotoUrl())
        .into(imgGoogle);
}

private void goLoginActivity() {
    Intent i = new Intent(this, LoginActivity.class);
    i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);
    startActivity(i);
}

```

## ii. *Añadir dispositivo*

```
private void addDevices() {
    final DatabaseReference reference =
        FirebaseDatabase.getInstance().getReference();

    AlertDialog.Builder aBuilder = new AlertDialog.Builder(MainActivity.this);
    View mView = getLayoutInflater().inflate(R.layout.dialog_add_devices,
        null);

    final EditText etIdDevice = mView.findViewById(R.id.etIdDevice);
    Button btnAddDevice = mView.findViewById(R.id.btnAdd);
    aBuilder.setView(mView);
    final AlertDialog dialog = aBuilder.create();
    dialog.show();

    btnAddDevice.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            final String idDevice =
                etIdDevice.getText().toString().trim(); // Se obtiene el ID introducido

            reference.child("users").child(currentUser.getId()).child("devices").child(idDevice).addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(DataSnapshot snapshot) {
                    if (snapshot.getValue() != null) { // Si en
                        // Firebase ya existe no le dejamos al usuario añadirlo de nuevo
                        Toast.makeText(getApplicationContext(), "El
                        ID introducido ya ha sido añadido", Toast.LENGTH_SHORT).show();
                    }
                    else if(idDevice.equals("")){ // Si no se
                        // introdujo nada se le muestra al usuario un Toast indicandolo
                        Toast.makeText(getApplicationContext(),
                        "Debes introducir un ID", Toast.LENGTH_SHORT).show();
                    }
                    else { // Sincronizamos el dispositivo con el
                        // usuario
                        addDeviceFirebase(idDevice, dialog);
                    }
                    reference.removeEventListener(this); // Se
                    // elimina el listener para liberar memoria
                }
                @Override
                public void onCancelled(DatabaseError databaseError) {
                    reference.removeEventListener(this); // Se
                    // elimina el listener para liberar memoria
                }
            });
        }
    });

    private void addDeviceFirebase(String idDevice, AlertDialog dialog) {
        Menu menu = navigationView.getMenu();
        menu.add(idDevice).setIcon(R.drawable.ic_arduino);
        DatabaseReference refRaiz = FirebaseDatabase.getInstance().getReference();
        DatabaseReference refUsers =
            refRaiz.child("users").child(firebaseAuth.getCurrentUser().getId());
        refUsers.child("devices").child(idDevice).setValue(idDevice);
        dialog.dismiss();
    }
}
```

### iii. Consultar dispositivos sincronizados

```
private void checkDevices() {
    // Se comprueba si es la primera vez que inicia sesion y se le asigna un
    // perfil en Real-Time Database
    final DatabaseReference reference =
    FirebaseDatabase.getInstance().getReference();
    ValueEventListener vel = new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot snapshot) {
            if (snapshot.getValue() != null) {
                devicesMap = (HashMap<String,String>)
                snapshot.getValue(); // Se obtienen los dispositivos en formato clave:valor
                ArrayList<String> devices = new
                ArrayList<>(devicesMap.values()); // Se meten los valores en un ArrayList

                for (int i = 0; i < devices.size(); i++) {
                    Menu menu = navigationView.getMenu();

                    menu.add(devices.get(i)).setIcon(R.drawable.ic_arduino);
                    Log.d("DEVICE", devices.get(i));
                }
            }
            reference.removeEventListener(this);
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {
            Log.d("USER", databaseError.getMessage());
            reference.removeEventListener(this);
        }
    };
    reference.child("users").child(currentUser.getUid()).child("devices").addL
    istenerForSingleValueEvent(vel);
}
```

### iv. Clase MessageReadReceiver

```
public class MessageReadReceiver extends BroadcastReceiver {
    private static final String TAG = MessageReadReceiver.class.getSimpleName();

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "ONRECEIVE");
        int conversationId =
        intent.getIntExtra(Constants.NOTIFICATION_ID_STRING, -1);
        if (conversationId != -1) {
            Log.d(TAG, "Notification " + conversationId + " was read");
            NotificationManagerCompat.from(context).cancel(conversationId);
        }
    }
}
```

## v. Clase MessageReplyReceiver

```
public class MessageReplyReceiver extends BroadcastReceiver {
    private static String KEY_NOTIFICATION_ID = "key_noticiation_id";
    private static String KEY_MESSAGE_ID = "key_message_id";

    public static Intent getReplyMessageIntent(Context context, int
notificationId, int messageId) {
        Intent intent = new Intent(context, MessageReplyReceiver.class);
        intent.setAction(Constants.REPLY_ACTION);
        intent.putExtra(KEY_NOTIFICATION_ID, notificationId);
        intent.putExtra(KEY_MESSAGE_ID, messageId);
        return intent;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        if (Constants.REPLY_ACTION.equals(intent.getAction())) {
            CharSequence message = getMessageText(intent);
            String channel = intent.getStringExtra("channel");

            processReply(context, channel, message); // Se procesa el tipo de
notificacion y la respuesta obtenida
        }
    }

    private CharSequence getMessageText(Intent intent) {
        Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);
        if (remoteInput != null) {
            return remoteInput.getCharSequence(Constants.VOICE_REPLY); // Se
obtiene la respuesta de voz
        }
        return "";
    }

    private void processReply(final Context context, String channel,
CharSequence message) {
        final DatabaseReference mReference =
FirebaseDatabase.getInstance().getReference("devices/0x00000001/");

        if (channel.equalsIgnoreCase(Constants.CHANNEL_LIGHT)) { // Si la
notificacion es de luz
            if
(Arrays.asList(Constants.COMANDOS_ENCENDER_LUZ).contains(message)) { // Si la
respuesta obtenida coincide con algun comando de encender luz
                //Toast.makeText(context, "Encendiendo luz...",
Toast.LENGTH_LONG).show();
                mReference.child("rele1/state").setValue("on"); // Encendemos
la luz
            }
            else if
(Arrays.asList(Constants.COMANDOS_APAGAR_LUZ).contains(message)) { // Si la
respuesta obtenida coincide con algun comando de apagar luz
                //Toast.makeText(context, "Apagando luz...",
Toast.LENGTH_LONG).show();
                mReference.child("rele1/state").setValue("off"); // Apagamos la
luz
            }
        }
        else if (channel.equalsIgnoreCase(Constants.CHANNEL_PLUG)) { // Si la
notificacion es de enchufe
```

```

        if
        (Arrays.asList(Constants.COMANDOS_ACTIVAR_ENCHUFE).contains(message)) { // Si
        la respuesta obtenida coincide con algun comando de activar enchufe
            //Toast.makeText(context, "Activando enchufe...",
            Toast.LENGTH_LONG).show();
            mReference.child("rele2/state").setValue("on"); // Activamos el
            enchufe
        }
        else if
        (Arrays.asList(Constants.COMANDOS_DESACTIVAR_ENCHUFE).contains(message)) { //
        Si la respuesta obtenida coincide con algun comando de desactivar enchufe
            //Toast.makeText(context, "Desactivando enchufe...",
            Toast.LENGTH_LONG).show();
            mReference.child("rele2/state").setValue("off"); //
            Desactivamos el enchufe
        }
    }
    else if (channel.equalsIgnoreCase(Constants.CHANNEL_GAS)) { // Si la
    notificacion es de gas
        if (Arrays.asList(Constants.COMANDOS_ESTADO_GAS).contains(message))
        { // Si la respuesta obtenida coincide con algun comando para conocer el nivel
        de riesgo de gas
            //Toast.makeText(context, "Leyendo estado del gas...",
            Toast.LENGTH_LONG).show();
            mReference.child("gas1/risk").addValueEventListener(new
            ValueEventListener() {
                @Override
                public void onDataChange(DataSnapshot snapshot) {
                    int risk =
                    Integer.parseInt(snapshot.getValue().toString()); // Se obtiene el valor
                    solicitado

                    // Mostramos una nueva notificacion con los datos
                    solicitados

                    new
                    NotificationHelper(context).showNotification(Constants.CHANNEL_GAS, "El riesgo
                    de gas es de nivel "+risk, Constants.CHANNEL_GAS, R.mipmap.gas_risk_one_icon,
                    false);

                    mReference.removeEventListener(this); // Se elimina el
                    listener una vez se recupero el dato correspondiente
                }
                @Override
                public void onCancelled(DatabaseError databaseError) {}
            });
        }
    }
    else if (channel.equalsIgnoreCase(Constants.CHANNEL_HUMIDITY)) { // Si
    la notificacion es de humedad
        if
        (Arrays.asList(Constants.COMANDOS_VALOR_HUMEDAD).contains(message)) { // Si la
        respuesta obtenida coincide con algun comando para conocer el porcentaje de
        humedad
            //Toast.makeText(context, "Leyendo valor de humedad...",
            Toast.LENGTH_LONG).show();
            mReference.child("humidity1/value").addValueEventListener(new
            ValueEventListener() {
                @Override
                public void onDataChange(DataSnapshot snapshot) {
                    float value =
                    Float.parseFloat(snapshot.getValue().toString()); // Se obtiene el valor
                    solicitado

```

```

        // Mostramos una nueva notificacion con los datos
solicitados

        new
NotificationHelper(context).showNotification(Constants.CHANNEL_HUMIDITY, "La
humedad es del "+value+" por ciento", Constants.CHANNEL_HUMIDITY,
R.mipmap.humidity_notification_icon, false);

        mReference.removeEventListener(this); // Se elimina el
listener una vez se recupero el dato correspondiente
    }
    @Override
    public void onCancelled(DatabaseError databaseError) {}
    });
}
}
else if (channel.equalsIgnoreCase(Constants.CHANNEL_TEMPERATURE)) { //
Si la notificacion es de temperatura
    if
(Array.asList(Constants.COMANDOS_VALOR_TEMPERATURA).contains(message)) { //
Si la respuesta obtenida coincide con algun comando para conocer la temperatura
        //Toast.makeText(context, "Leyendo valor de temperatura...",
Toast.LENGTH_LONG).show();
        mReference.child("temperature1/value").addValueEventListener(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot snapshot) {
                float value =
Float.parseFloat(snapshot.getValue().toString()); // Se obtiene el valor
solicitado

                // Mostramos una nueva notificacion con los datos
solicitados

                new
NotificationHelper(context).showNotification(Constants.CHANNEL_TEMPERATURE, "La
temperatura es de "+value+" grados", Constants.CHANNEL_TEMPERATURE,
R.mipmap.notification_temperature_icon, false);

                mReference.removeEventListener(this); // Se elimina el
listener una vez se recupero el dato correspondiente
            }
            @Override
            public void onCancelled(DatabaseError databaseError) {}
        });
    }
}
//Toast.makeText(context, "Reply: " + message,
Toast.LENGTH_LONG).show();
Log.d("REPLY", message.toString());

NotificationManagerCompat.from(context).cancel(Constants.NOTIFICATION_ID_INT);
// Se cancela la notificacion una vez ha sido respondida
    }
}

```

## vi. Clase NotificationService

```
public class NotificationService extends IntentService {
    private static final String TAG = NotificationService.class.getSimpleName();

    public NotificationService() {
        super(NotificationService.class.getSimpleName());
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        // Handle intent to send a new notification.
        if (intent != null &&
Constantes.SHOW_NOTIFICATION.equals(intent.getAction())) {

            sendNotification(
                Constantes.NOTIFICATION_ID_INT,
                intent.getStringExtra("title"),
                intent.getStringExtra("text"),
                intent.getStringExtra("channel"),
                intent.getIntExtra("icon",
android.R.drawable.sym_def_app_icon),
                intent.getBooleanExtra("hasReply", false),
                System.currentTimeMillis());
        }

        // Creates an intent that will be triggered when a message is read.
        private Intent getMessageReadIntent(int id) {
            return new
Intent().setAction(Constantes.READ_ACTION).putExtra(Constantes.NOTIFICATION_ID_S
TRING, id);
        }

        // Creates an Intent that will be triggered when a voice reply is received.
        private Intent getMessageReplyIntent(int conversationId, String channel) {
            return new Intent()
                .setAction(Constantes.REPLY_ACTION)
                .putExtra("channel", channel)
                .putExtra(Constantes.NOTIFICATION_ID_STRING, conversationId);
        }

        private void sendNotification(
            int conversationId,
            String sender,
            String message,
            String channel,
            int icon,
            boolean hasReply,
            long timestamp) {

            createNotificationChannel();

            // A pending Intent for reads.
            PendingIntent readPendingIntent = PendingIntent.getBroadcast(
                getApplicationContext(),
                conversationId,
                getMessageReadIntent(conversationId),
                PendingIntent.FLAG_UPDATE_CURRENT);

            // Building a Pending Intent for the reply action to trigger.
            PendingIntent replyIntent = PendingIntent.getBroadcast(
```

```

        getApplicationContext(),
        conversationId,
        getMessageReplyIntent(conversationId, channel),
        PendingIntent.FLAG_UPDATE_CURRENT);

    /// TODO: Add the code to create the UnreadConversation.
    // Build a RemoteInput for receiving voice input from devices.
    RemoteInput remoteInput = new
RemoteInput.Builder(Constants.VOICE_REPLY).build();

    // Create the UnreadConversation and populate it with the participant
name,
    // read and reply intents.
    NotificationCompat.CarExtender.UnreadConversation.Builder
unreadConversationBuilder =
        new
NotificationCompat.CarExtender.UnreadConversation.Builder(sender)
            .setLatestTimestamp(timestamp)
            .setReadPendingIntent(readPendingIntent)
            .setReplyAction(replyIntent, remoteInput);

    // Note: Add messages from oldest to newest to the
UnreadConversation.Builder.
    // Since we are sending a single message here we simply add the message.
    // In a real world application there could be multiple unread messages
which
    // should be ordered and added from oldest to newest.
unreadConversationBuilder.addMessage(message);
    /// End create UnreadConversation

    // TODO: Add the code to allow inline reply on Wear 2.0.
    // Wear 2.0 allows for inline actions, which will be used for "reply"
NotificationCompat.Action.WearableExtender inlineActionForWear2 =
        new NotificationCompat.Action.WearableExtender()
            .setHintDisplayActionInline(true)
            .setHintLaunchesActivity(false);
    /// End inline action for Wear 2.0.

    // 2. Build action
    NotificationCompat.Action replyAction = new
NotificationCompat.Action.Builder(
        android.R.drawable.sym_action_chat, "Reply",
getReplyPendingIntent(channel))
        .addRemoteInput(remoteInput)
        .extend(inlineActionForWear2) // TODO: Add better Wear support.
        .setAllowGeneratedReplies(true)
        .build();

    // Creamos la notificacion
NotificationCompat.Builder builder =
        new NotificationCompat.Builder(getApplicationContext(), channel)
// Obtenemos el contexto y le asignamos a que grupo de notificaciones pertenece
        .setDefaults(NotificationCompat.DEFAULT_VIBRATE)
        .setSmallIcon(R.mipmap.notification_small_icon_white)

.setLargeIcon(BitmapFactory.decodeResource(getApplicationContext().getResources(
), icon))
        .setWhen(timestamp)
        .setColor(Color.BLACK) // Se establece el color de
fondo de la notificacion cuando se use el smallIcon
        .setAutoCancel(true)
        .setContentTitle(sender)
        .setContentText(message)
        .setContentIntent(readPendingIntent)

```



```

        /// TODO: Extend the notification with CarExtender.
        .extend(new NotificationCompat.CarExtender())

.setUnreadConversation(unreadConversationBuilder.build());
    builder.setStyle(new
NotificationCompat.BigTextStyle().bigText(message));

    if (hasReply) { // Si se ha indicado que la notificacion debe tener
respuesta se le añade
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N)
builder.addAction(replyAction); // Aunque se quiera respuesta, solo se permite
para versiones superiores a Android N
    }

    Log.d(TAG, "Sending notification " + conversationId + " conversation: "
+ message);
    NotificationManagerCompat.from(this).notify(conversationId,
builder.build());
}

private void createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channelGas = new
NotificationChannel(Constants.CHANNEL_GAS, Constantes.CHANNEL_GAS,
NotificationManager.IMPORTANCE_HIGH);
        channelGas.setDescription("Recibir notificaciones de los sensores de
gas");

        NotificationChannel channelHumidity = new
NotificationChannel(Constants.CHANNEL_HUMIDITY, Constantes.CHANNEL_HUMIDITY,
NotificationManager.IMPORTANCE_HIGH);
        channelHumidity.setDescription("Recibir notificaciones de los
sensores de humedad");

        NotificationChannel channelLight = new
NotificationChannel(Constants.CHANNEL_LIGHT, Constantes.CHANNEL_LIGHT,
NotificationManager.IMPORTANCE_HIGH);
        channelLight.setDescription("Recibir notificaciones si se enciende o
se apaga la luz");

        NotificationChannel channelPlug = new
NotificationChannel(Constants.CHANNEL_PLUG, Constantes.CHANNEL_PLUG,
NotificationManager.IMPORTANCE_HIGH);
        channelPlug.setDescription("Recibir notificaciones si se enciende o
se apaga el enchufe");

        NotificationChannel channelTemperature = new
NotificationChannel(Constants.CHANNEL_TEMPERATURE,
Constantes.CHANNEL_TEMPERATURE, NotificationManager.IMPORTANCE_HIGH);
        channelTemperature.setDescription("Recibir notificaciones de los
sensores de temperatura");

        NotificationManager notificationManager =
getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channelGas);
        notificationManager.createNotificationChannel(channelHumidity);
        notificationManager.createNotificationChannel(channelLight);
        notificationManager.createNotificationChannel(channelPlug);
        notificationManager.createNotificationChannel(channelTemperature);
    }
}

private PendingIntent getReplyPendingIntent(String channel) {
    Intent intent;

```

```

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
            // start a
            // (i) broadcast receiver which runs on the UI thread or
            // (ii) service for a background task to be executed , but for the
            purpose of this code lab, will be doing a broadcast receiver
            intent =
            MessageReplyReceiver.getReplyMessageIntent(getApplicationContext(),
            Constantes.NOTIFICATION_ID_INT, 1);
            intent.putExtra("channel", channel);
            return PendingIntent.getBroadcast(getApplicationContext(), 100,
            intent, PendingIntent.FLAG_UPDATE_CURRENT);
        }
        else {
            // start your activity
            intent =
            MessageReplyReceiver.getReplyMessageIntent(getApplicationContext(),
            Constantes.NOTIFICATION_ID_INT, 1);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            intent.putExtra("channel", channel);
            return PendingIntent.getActivity(getApplicationContext(), 100,
            intent, PendingIntent.FLAG_UPDATE_CURRENT);
        }
    }
}

```

### *vii. Clase NotificationHelper*

```

public class NotificationHelper extends ContextWrapper {
    private Intent notification;

    public NotificationHelper(Context c) {
        super(c);

        notification = new Intent(getApplicationContext(),
        NotificationService.class);
    }

    public void showNotification(String title, String text, String channel, int
    icon, boolean hasReply) {
        notification.setAction(Constantes.SHOW_NOTIFICATION);
        notification.putExtra("title", title);
        notification.putExtra("text", text);
        notification.putExtra("channel", channel);
        notification.putExtra("icon", icon);
        notification.putExtra("hasReply", hasReply);
        startService(notification);
    }
}

```

### *viii. Clase Service Startup*

```
public class ServiceStartup extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d("ServiceStartup", "ONBOOTCOMPLETED");

        Intent startServiceIntent = new Intent(context, ServiceListener.class);
        context.startService(startServiceIntent);
    }
}
```

### *ix. Clase ServiceListener*

```
public class ServiceListener extends Service {
    private final IBinder mBinder;
    private static ServiceListener INSTANCE = null;
    private static FirebaseDatabase database;
    private static FirebaseAuth mAuth;
    private static FirebaseUser currentUser;
    private static DatabaseReference myRef;
    private static ValueEventListener listener;
    private static Device dispositivoOld;
    private static Device dispositivoNew;
    private static boolean isFirstRead;
    private static Trace trace;
    private static NotificationHelper nh;

    public ServiceListener() {
        mBinder = new LocalBinder();
        database = FirebaseDatabase.getInstance();
        dispositivoOld = new Device();
        dispositivoNew = new Device();
        isFirstRead = true;
    }

    public static ServiceListener getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new ServiceListener();
            listener = new ValueEventListener() {
                @Override
                public void onDataChange(DataSnapshot dataSnapshot) {
                    if (isFirstRead) {

dispositivoOld.setGas(dataSnapshot.child("gas1").getValue(Gas.class));

dispositivoOld.setHumidity(dataSnapshot.child("humidity1").getValue(Humidity.class));

dispositivoOld.setLight(dataSnapshot.child("rele1").getValue(Relay.class));

dispositivoOld.setPlug(dataSnapshot.child("rele2").getValue(Relay.class));

dispositivoOld.setTemperature(dataSnapshot.child("temperature1").getValue(Temperature.class));

                        trace.logDevice(dispositivoOld);
                    }
                }
            };
        }
    }
}
```

```

        isFirstRead = false;
    }
    else {

dispositivoNew.setGas(dataSnapshot.child("gas1").getValue(Gas.class));

dispositivoNew.setHumidity(dataSnapshot.child("humidity1").getValue(Humidity.class));

dispositivoNew.setLight(dataSnapshot.child("rele1").getValue(Relay.class));

dispositivoNew.setPlug(dataSnapshot.child("rele2").getValue(Relay.class));

dispositivoNew.setTemperature(dataSnapshot.child("temperature1").getValue(Temperature.class));

        trace.logDevice(dispositivoNew);
        processChanges(whatChanged(dispositivoOld,
dispositivoNew));
    }
}

@Override
public void onCancelled(DatabaseError error) {}
};
myRef = database.getReference("devices/0x00000001");
}
return INSTANCE;
}

private static String whatChanged(Device dOld, Device dNew) {
    if (dOld.getGas().getRisk() != dNew.getGas().getRisk()) {
        return "gas";
    }
    else if (dOld.getHumidity().getValue() != dNew.getHumidity().getValue())
{
        return "humidity";
    }
    else if
(!dOld.getLight().getState().equalsIgnoreCase(dNew.getLight().getState())) {
        return "light";
    }
    else if
(!dOld.getPlug().getState().equalsIgnoreCase(dNew.getPlug().getState())) {
        return "plug";
    }
    else if (dOld.getTemperature().getValue() !=
dNew.getTemperature().getValue()) {
        return "temperature";
    }
    else {
        return "nochanges";
    }
}

private static void processChanges(String changes) {
    if (changes.equalsIgnoreCase("gas")) {
        if (dispositivoNew.getGas().getRisk() == 0) {
            nh.showNotification("Gas", "No se han detectado gases nocivos
para la salud", Constantes.CHANNEL_GAS, R.mipmap.gas_risk_zero_icon, false);
        }
        else if (dispositivoNew.getGas().getRisk() == 1) {
            nh.showNotification("Gas", "¡Es posible que tengas una fuga de
gas!", Constantes.CHANNEL_GAS, R.mipmap.gas_risk_one_icon, false);

```

```

    }
    else if (dispositivoNew.getGas().getRisk() == 2) {
        nh.showNotification("Gas", "Toma medidas de precaución, ¡se han
detectado niveles altísimos de gas!", Constantes.CHANNEL_GAS,
R.mipmap.gas_risk_two_icon, false);
    }
    dispositivoOld.getGas().setRisk(dispositivoNew.getGas().getRisk());
    // Se guarda el nuevo valor en el objeto antiguo para ser comaparado de nuevo
}
    else if (changes.equalsIgnoreCase("humidity")) {
        if (dispositivoNew.getHumidity().getValue() <= 25) {
            nh.showNotification("Humedad", "¿También has notado lo seco que
está el ambiente?", Constantes.CHANNEL_HUMIDITY,
R.mipmap.humidity_notification_icon, false);
        }
        else if (dispositivoNew.getHumidity().getValue() <= 50) {
            nh.showNotification("Humedad", "¡Hace la humedad perfecta para
jugar a la play!", Constantes.CHANNEL_HUMIDITY,
R.mipmap.humidity_notification_icon, false);
        }
        else if (dispositivoNew.getHumidity().getValue() <= 75) {
            nh.showNotification("Humedad", "Noto que estoy empezando a sudar
y soy un móvil...", Constantes.CHANNEL_HUMIDITY,
R.mipmap.humidity_notification_icon, false);
        }
        else if (dispositivoNew.getHumidity().getValue() <= 100) {
            nh.showNotification("Humedad", "¡Esto es peor que el Amazonas!",
Constantes.CHANNEL_HUMIDITY, R.mipmap.humidity_notification_icon, false);
        }
    }

    dispositivoOld.getHumidity().setValue(dispositivoNew.getHumidity().getValue());
    // Se guarda el nuevo valor en el objeto antiguo para ser comaparado de nuevo
}
    else if (changes.equalsIgnoreCase("light")) {
        if (dispositivoNew.getLight().getState().equalsIgnoreCase("on")) {
            nh.showNotification("Luz", "¡Alguien encendió la luz!",
Constantes.CHANNEL_LIGHT, R.mipmap.light_on_notification_icon, false);
        }
        else if
(dispositivoNew.getLight().getState().equalsIgnoreCase("off")) {
            nh.showNotification("Luz", "¡Alguien apagó la luz!",
Constantes.CHANNEL_LIGHT, R.mipmap.light_off_notification_icon, false);
        }
    }

    dispositivoOld.getLight().setState(dispositivoNew.getLight().getState()); // Se
guarda el nuevo valor en el objeto antiguo para ser comaparado de nuevo
}
    else if (changes.equalsIgnoreCase("plug")) {
        if (dispositivoNew.getPlug().getState().equalsIgnoreCase("on")) {
            nh.showNotification("Enchufe", "¡Alguien activó el enchufe!",
Constantes.CHANNEL_PLUG, R.mipmap.connected_icon, false);
        }
        else if
(dispositivoNew.getPlug().getState().equalsIgnoreCase("off")) {
            nh.showNotification("Enchufe", "¡Alguien desactivó el enchufe!",
Constantes.CHANNEL_PLUG, R.mipmap.disconnected_icon, false);
        }
    }

    dispositivoOld.getPlug().setState(dispositivoNew.getPlug().getState()); // Se
guarda el nuevo valor en el objeto antiguo para ser comaparado de nuevo
}
    else if (changes.equalsIgnoreCase("temperature")) {
        float tempNew = dispositivoNew.getTemperature().getValue();
        if (tempNew > 40.0) {

```

```

        nh.showNotification("Temperatura", "¿Es posible que tu casa esté
ardiendo?", Constantes.CHANNEL_TEMPERATURE, R.mipmap.temperature_fire_icon,
false);
    }
    else if (tempNew > 25.0) {
        nh.showNotification("Temperatura", "¡Con este calor no olvides
hidratarte!", Constantes.CHANNEL_TEMPERATURE, R.mipmap.temperature_hot_icon,
false);
    }
    else if (tempNew > 15.0) {
        nh.showNotification("Temperatura", "¡Hace una temperatura que da
gusto!", Constantes.CHANNEL_TEMPERATURE, R.mipmap.temperature_ideal_icon,
false);
    }
    else if (tempNew >= 5.0) {
        nh.showNotification("Temperatura", "Creo que deberías
abrigarte...", Constantes.CHANNEL_TEMPERATURE, R.mipmap.temperature_cold_icon,
false);
    }
    else if (tempNew < 5.0) {
        nh.showNotification("Temperatura", "Me da la sensación de estar
en la Antártida...", Constantes.CHANNEL_TEMPERATURE,
R.mipmap.temperature_ice_icon, false);
    }
}

dispositivoOld.getTemperature().setValue(dispositivoNew.getTemperature().getValu
e()); // Se guarda el nuevo valor en el objeto antiguo para ser comaparado de
nuevo
    }
}

// Clase para poder consumir datos de este servicio desde otras activities
public class LocalBinder extends Binder {
    public ServiceListener getService() {
        return getInstance(); // Return this instance of ServiceListener so
clients can call public methods
    }
}

@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

@Override
public void onCreate() {
    super.onCreate();
    INSTANCE = getInstance(); // Se inicializa el singleton del servicio
cuando se completa el boot del sistema
    trace = new Trace();
    nh = new NotificationHelper(getApplicationContext());

    // Configure Firebase modules
    mAuth = FirebaseAuth.getInstance();
    currentUser = mAuth.getCurrentUser(); // Se obtiene el usuario actual
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // Si no hay algun usuario con la sesion iniciada se limpia la sesion y
se detiene el servicio
    if (currentUser == null) {
        mAuth.signOut();
        stopSelf();
    }
}

```

```

        Log.d("ServiceStartup", "NOSESSION");
    }
    else {
        myRef.addValueEventListener(listener); // Se añade el listener de
Firebase
    }

    return super.onStartCommand(intent, flags, startId);
}

@Override
public void onDestroy() {
    super.onDestroy();
    Log.d("ServiceListener", "ONDESTROY");
    myRef.removeEventListener(listener); // Se elimina el listener de
Firebase cuando se detiene el servicio
}
}

```

## x. AndroidManifest

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.quadram.futh">

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <!-- <uses-feature android:name="android.hardware.type.watch" /> Comentar y
descomentar esta línea para instalar la app en AndroidWear -->

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher_square"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_rounded"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.Light.NoActionBar">

        <meta-data
            android:name="com.google.android.gms.car.application"
            android:resource="@xml/automotive_app_desc" />

        <activity
            android:name=".LoginActivity"
            android:theme="@style/Theme.AppCompat.Light.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main2"
            android:theme="@style/Theme.AppCompat.Light.NoActionBar" />

        <!-- Servicio que inicia el listener de Firebase cuando arranca Android
-->

        <receiver
            android:name=".service.ServiceStartup"

```

```

        android:enabled="true"
        android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>

    <!-- Servicio que monitoriza los cambios en Firebase y muestra
notificaciones -->
    <service
        android:name=".service.ServiceListener"
        android:enabled="true"
        android:exported="true"
        android:permission="android.permission.BIND_JOB_SERVICE" />

    <receiver
        android:name=".notification.MessageReadReceiver"
        android:exported="false">
        <intent-filter>
            <action android:name="com.quadram.futh.ACTION_MESSAGE_READ" />
        </intent-filter>
    </receiver>
    <receiver
        android:name=".notification.MessageReplyReceiver"
        android:exported="false">
        <intent-filter>
            <action android:name="com.quadram.futh.ACTION_MESSAGE_REPLY" />
        </intent-filter>
    </receiver>

    <service
        android:name=".notification.NotificationService"
        android:enabled="true"
        android:exported="false"/>
</application>

</manifest>

```