PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Escuela de Ingeniería Eléctrica
PONTIFICIA UNIVERSIDAD CATOLICA DE VALPARAISO

Marcelo Antonio Figueroa González

Estudio y desarrollo de una aplicación movil (Android) para el monitoreo de sensores aplicados a domótica en base al protocolo MQTT

Informe Proyecto de Título de Ingeniero Civil Electrónico

Escuela de Ingeniería Eléctrica
Facultad de Ingeniería

Valparaíso, 17 de agosto de 2018

Estudio y desarrollo de una aplicación móvil (Android) para el monitoreo de sensores aplicados a domótica en base al protocolo MQTT

Marcelo Antonio Figueroa González

Informe Final para optar al título de Ingeniero Civil Electrónico,

aprobada por la comisión de la

Escuela de Ingeniería Eléctrica de la

Facultad de Ingeniería de la

Pontificia Universidad Católica de Valparaíso

conformada por

Sr. Francisco Javier Alonso Villalobos

Profesor Guía

Sr. David Velasco López

Segundo Revisor

Sr. Sebastián Fingerhuth Massmann

Secretario Académico

Valparaíso, 17 de agosto de 2018

Agradecimientos

Doy gracias, en primer lugar, a mis padres Liliana González y Marco Figueroa, por todo su cariño, apoyo incondicional, por todos los esfuerzos realizados a lo largo de mi vida para educarme y entregarme los valores que hacen de mí la persona que soy. A mis hermanos por acompañarme y ayudarme en todo momento.

A mis amigos y compañeros de universidad, por los momentos de alegría y risas, por ayudarme y acompañarme en momentos difíciles de mi vida. Agradezco enormemente a la Universidad y a la Escuela de Ingeniería Eléctrica por formarme y apoyarme en las dificultades. A los profesores que me brindaron el conocimiento y sabiduría para seguir estudiando profesionalmente.

También a los profesores David Velasco y Francisco Alonso, por sus múltiples consejos y apoyos para realizar el proyecto. Un agradecimiento en especial a mi profesor guía, sin el cual no sería posible esto, al darme las herramientas, posibilidad y motivación a desarrollar un proyecto personal.

Valparaíso, 17 de agosto de 2018

M. F. G

Resumen

Se realiza un estudio del estado del arte del protocolo de comunicación MQTT, Internet de las Cosas, Android y Domótica. Se profundiza en cada uno de los conceptos mencionados para poder tener los conocimientos necesarios de base para realizar y proponer el proyecto.

Se analiza la arquitectura del protocolo MQTT, los fundamentos y arquitectura de Android, el lenguaje de programación Python y Java. Se realiza un estudio de la herramienta de desarrollo Android Studio para la creación de aplicaciones móviles y se presenta de forma detallada los principales actores del proyecto junto a una estructura a desarrollar.

Luego se trabajó con la placa Raspberry Pi, su configuración e instalación de servicios necesarios para la comunicación por medio de MQTT. Se dan a conocer los programas desarrollados para la lectura de los datos obtenidos con los sensores, su conexión y pruebas realizadas. Además de la automatización del funcionamiento de la Raspberry Pi.

Posteriormente se presenta el desarrollo de la aplicación móvil para Android, planteando los requisitos funcionales, no funcionales y diseño que tendrá. Se estudia el uso del protocolo MQTT en Java y la forma de establecer la comunicación por medio de este protocolo. Tras finalizar el desarrollo se da a conocer la aplicación creada en Android Studio.

Se realiza un diseño 3D para el soporte del hardware involucrado en el proyecto, se muestra su modelado, resultados y montaje con los dispositivos trabajados. Por último se presentan los resultados obtenidos del proyecto, como los del dispositivo portátil, aplicación móvil y uso del protocolo MQTT.

Se finaliza con las conclusiones, discusiones y comentarios del proyecto realizado. Además de presentar los posibles trabajos futuros para nuevas iteraciones del mismo.

Palabras claves: Protocolo MQTT, Internet de las Cosas, Android, Domótica, Java, Python, Aplicación móvil, Raspberry Pi.

Abstract

A study of the state of the art of the MQTT communication protocol, Internet of Things, Android and Domotica. Each of the concepts mentioned above is studied in depth in order to have the necessary basic knowledge to carry out and propose the project.

It analyzes the architecture of the MQTTT protocol, the fundamentals and architecture of Android, the Python programming language and Java. A study of the Android Studio development tool for the creation of mobile applications is carried out and the main actors of the project are presented in detail together with a structure to be developed.

After worked with the Raspberry Pi board, its configuration and installation of services necessary for communication via MQTT. The programs developed for the reading of the data obtained with the sensors, their connection and the tests carried out are presented. In addition to the automation of the operation of the Raspberry Pi

Subsequently, the development of the mobile application for Android is presented, setting out the functional, non-functional and design requirements it will have. The use of the MQTT protocol in Java and how to establish communication through this protocol is studied. After the development is finished, the application created in Android Studio is released.

A 3D design is made to support the hardware involved in the project, showing its modeling, results and assembly with the devices worked. Finally, the results obtained from the project are presented, such as those of the portable device, mobile application and use of the MQTT protocol.

It ends with the conclusions, discussions and comments on the project carried out. In addition to presenting the possible future work for new iterations of it.

Key words: MQTT Protocol, Internet of Things, Android, Domotica, Java, Python, Mobile application, Raspberry Pi.

Índice general

Introducción

En la última década la tecnología ha ido evolucionando estrepitosamente de tal forma que cada día se inventan nuevas y mejores cosas con mucha más capacidad, menor tamaño y mejor rendimiento. Ejemplo claro de esto son los dispositivos móviles inteligentes o Smartphone, a los cuales se les pueden agregar distintas aplicaciones (Apps) que sirven para todo tipo de tareas, desde escuchar música hasta hacer una compleja transacción bancaria, haciendo a estos dispositivos indispensables en nuestra vida.

Los dispositivos móviles inteligentes son la respuesta tecnológica a la necesidad inherente del ser humano a estar comunicado frente a un mundo más globalizado. Desde la aparición del internet han sucedido cambios profundos en la manera de comunicarnos y acceder a información que está al otro lado del mundo con un solo click. Con esto nació la idea de poder reunir en un solo dispositivo todas las funciones de los medios de comunicación personales, beneficios de internet adicionando espacio para el entretenimiento, el ocio, la información y una conexión continua. Todo en un aparato que cabe perfectamente en un bolsillo.

El éxito de estos aparatos ha sido inminente, generando grandes cambios en la vida cotidiana, nuevos mercados y nuevas aplicaciones para los Smartphone. Actualmente los dispositivos móviles tienen un gran impacto en el país, siendo Chile el líder Latinoamericano en el uso de internet y Smartphone.

Uno de los conceptos que más se han desarrollado debido a los avances tecnológicos, es que más dispositivos se conectan entre sí a través de internet en lo que hoy se conoce como el “Internet of Things (IoT)”. Los mensajes enviados de un lugar a otro, pueden ser descifrados y ejecutados de forma inteligente por una máquina, en lugar de un humano, lo que facilita enormemente la vida de las personas. Básicamente IoT es la interconexión, monitoreo y control de dispositivos, maquinas o sensores, tareas las cuales se pueden usar para distintas aplicaciones de las que se puede destacar la domótica.

La domótica es el conjunto de tecnologías aplicadas al control, monitoreo y automatización inteligente de una vivienda, que permite gestionar de forma eficiente diversas tareas, además de aportar seguridad, confort, y comunicación didáctica entre el usuario y el sistema. Para la realización de lo anterior, se hace necesario que se recoja la información del entorno con diversos sensores y que se disponga de actuadores en base a la información recogida.

Por ello el aspecto de las comunicaciones o intercambio de información, en un sistema de domótica se requiere que esta sea confiable, segura, rápida y ligera, todo lo anterior se puede cumplir con un protocolo conocido como MQTT.

MQTT (Message Queue Telemetry Transport) es un protocolo que se enfoca en la conectividad Machine-to-Machine (M2M), esta tecnología permite a los dispositivos la comunicación entre ellos de forma inalámbrica o por cable siguiendo una topología estrella. El protocolo se destaca principalmente en que consume muy poco ancho de banda haciéndolo ideal para la comunicación de sensores utilizando muy pocos recursos [1].

La combinación de las ideas mencionadas, se verán reflejadas en el desarrollo de una aplicación móvil para el sistema operático Android, en el cual se pueda monitorear una serie de sensores que se usan comúnmente para domótica. Esto estará acompañado de un producto físico, el cual contara con un dispositivo Raspberry Pi, donde estarán los sensores requeridos junto a su conexión y programación permitiendo la operación de monitoreo de un lugar en específico.

Un aspecto importante en el proyecto son los protocolos de comunicación, que corresponden al conjunto de pautas que posibilitan a distintos elementos a formar parte de un sistema, estableciendo la comunicación permitiendo el intercambio de información. La Raspberry Pi es un computador de placa reducida de bajo coste y es del tamaño de una tarjeta de crédito en comparación a otras tarjetas de desarrollo, tiene un gran poder de procesamiento de datos, capaz de realizar múltiples tareas, al igual que su capacidad de almacenar información, que permite usar este computador portátil para diversas aplicaciones. Esta utiliza el sistema operativo Raspbian, por lo que se puede programar en diversos lenguajes de programación, si se cuenta con el compilador indicado. Además cuenta con los protocolos de comunicación más importantes, pudiendo incluir mediante software el uso del protocolo MQTT.

Para llegar a un resultado final que cumpla con las funciones y especificaciones mencionadas anteriormente, se hace necesario realizar un estudio exhaustivo, teniendo en consideración las características y funciones de cada uno de los componentes a utilizar. En el resultado también se debe considerar y enfocar para el usuario final. Por lo que el dispositivo físico debe ser agradable e intuitivo en su funcionamiento, al igual que la aplicación móvil para Android. La aplicación móvil debe contar con un nombre, icono e información de su uso.

Otro aspecto importante para el desarrollo del proyecto es la utilización de distintos lenguajes de programación, por lo que se debe realizar un estudio minucioso de estos. Los cuáles serán principalmente en Java para la aplicación móvil y en Python para la lectura y envió de los datos tomados por los sensores en la Raspberry Pi. Con este estudio se logra desarrollar los distintos programas que harán funcionar de manera óptima la lectura y comunicación.

Se requiere de una caja que sea cómoda, resistente, ergonómica y agradable a la vista para el usuario final. Para esto se realizó un diseño 3D, que corresponde a un representación tridimensional de datos geométricos permitiendo un proyección visual en dos dimensiones para ser mostrada en pantalla o en un papel. Esto ayuda de gran manera en la actualidad para distintos proyectos permitiendo plasmar y diseñar la caja que contendrá el hardware usado en el proyecto.

Se cuentas con distintos softwares que permiten el modelado en 3D, uno de ellos y el cual se utilizara para modelar es SketchUp. Este modelo puede ser impreso de forma física gracias a una impresora 3D, que es una maquita capaz de transformar diseños en softwares de modelado 3D y llevarlos a un plano tridimensional. Esto mediante el uso de materiales plásticos como PLA (ácido poli-láctico), el cual se funde y se hace un barrido tridimensional por medio de la impresión en 3D.

Por lo dicho, en este proyecto se mostrara la combinación de manera íntegra los principales aspectos de las aplicaciones móviles, Internet de las cosas y domótica, teniendo como eje el protocolo MQTT. Dando solución a problemáticas reales que afectan al mundo y en especial a Chile.

Objetivo General

- Desarrollar una aplicación móvil para la plataforma Android, con el fin de monitorear un conjunto de sensores usados en domótica por medio del protocolo MQTT.

Objetivos Específicos

- Revisión del estado del arte acerca del protocolo MQTT, Internet de las cosas, Android, Domótica y sus aplicaciones.
- Comprender y analizar la arquitectura del protocolo MQTT junto a su funcionamiento.
- Estudio del desarrollo de aplicaciones móviles para Android y sus fundamentos.
- Estudio electrónico e informático del funcionamiento de la Raspberry Pi y de sensores aplicados a domótica.
- Comprender y analizar la comunicación en Java e Python mediante el protocolo MQTT.
- Desarrollar una App móvil y un dispositivo portátil que pueda competir en el mercado de la domótica y que sea escalable para futuras mejoras.

1 Antecedentes del proyecto

El objetivo principal de este capítulo es lograr contextualizar al lector con respecto al trabajo presentado. Se expone la problemática que se quiere resolver, un estado del arte de los principales aspectos del proyecto y la solución propuesta para lograrlo.

1.1 Problemática.

Actualmente en nuestro país se encuentra una gran cantidad de dispositivos móviles inteligentes, siendo Chile el líder Latinoamericano en el uso de internet y Smartphone [2], por ello se hace necesario la búsqueda de aplicaciones que mediante el uso de los Smartphone, puedan dar solución a problemas que afectan a las personas en nuestro país. Los problemas que más se ven afectada la población son los de calidad de vida, prevención de desastres y resguardo al medio ambiente.

Los principales problemas de calidad de vida son los de seguridad, comodidad y bienestar en el hogar. En la prevención de desastres se destacan los accidentes en las viviendas debidos a incendios, los cuales son la principal causa en muertes en niños según un estudio realizado en el año 2013 por la policía de investigación (PDI) [3]. Y finalmente en los problemas al medio ambiente están los del malgasto de energético y fuga de gases peligrosos al ambiente.

Debido a los problemas que se presentan surge la necesidad de implementar nuevas y distintas tecnologías que permitan resolver a mayor medida los problemas planteados. Nuevas tecnologías como el protocolo de comunicación MQTT y su potencial aplicación para la domótica. Con el desarrollo de este proyecto, se busca mejorar la calidad de vidas de las personas y aportar una solución que da un uso a los teléfonos inteligentes mediante una App.

1.2 Estado del arte

En este apartado, se expone una visión general del estado en que se encuentra el protocolo MQTT, internet de las cosas, la plataforma Android y domótica. También se expondrá las principales aplicaciones de estos.

1.2.1 Protocolo MQTT

El protocolo MQTT cuenta con diversas aplicaciones en dispositivos móviles, pero una de las más conocidas es la aplicación de chat Facebook Messenger. Esta App se encuentra disponible en las plataformas de iOS, Windows Phone y Android, con la funcionalidad de poder intercambiar mensajes en tiempo real.

Facebook Messenger [4] fue creada por Ben Daveport, Jon Perlow y Lucy Zhang, quienes indicaron que un comienzo, hubo problemas al crear la aplicación debido a la latencia al enviar mensajes, era confiable pero lento. Por lo anterior es que se optó por un nuevo mecanismo de comunicación, utilizar el protocolo MQTT en su aplicación de chat, ya que esta cuenta con un buen flujo y manejo de mensajes, se logró alcanzar la entrega de estos en cientos de miles de segundos. Cabe destacar que esta App cuenta con una base de 1.200 millones de usuarios activos mensualmente, demostrando lo cotidiano que se hace usar las aplicaciones móviles.

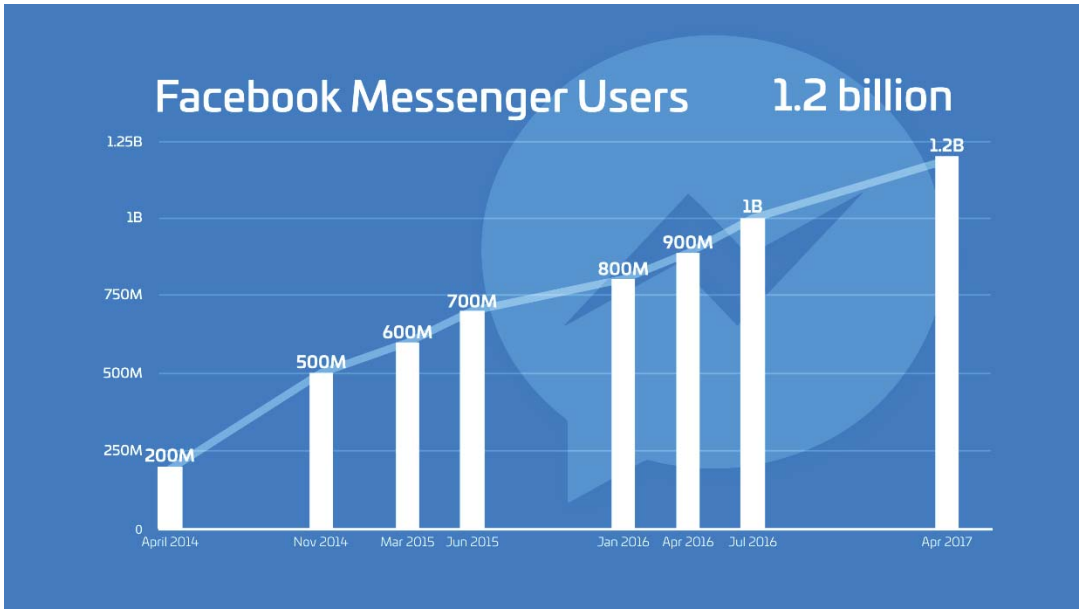

Figura 1-1 Histograma de la base de usuarios de Facebook Messenger.

Otras de las aplicaciones más comunes realizadas con el protocolo MQTT que se debe mencionar, son las de medición de la temperatura con un sensor conectado a una placa Arduino. Los datos enviados por el sensor (publicador) son enviados de forma segura por el protocolo MQTT a un servidor “Broker” que puede ser local o estar en la nube. Con todos estos elementos se puede observar en tiempo real y de forma gráfica la temperatura de algún ambiente en particular. Como ejemplo de esta aplicación se puede ver en la *Figura 1-2*, utilizando la nube de IoT Foundation.

Figura 1-2 Diseño secuencia de medición de temperatura usando MQTT [1].

También están las aplicaciones de MQTT relacionadas con el ámbito de la medicina, ya que es posible utilizar la telemetría para supervisar la salud de un paciente estando en su hogar. Por ejemplo para una persona con problemas al corazón se le puede crear una solución domestica de marcapaso cardiaco, la cual permita una supervisión al estado actual del paciente, una mejora en la eficiencia de chequeos posteriores y que cumpla con estándares de seguridad requeridos. Esta solución incorpora un cliente MQTT en un dispositivo supervisor domestico para recopilar datos de diagnóstico, estos se enviaran a través de internet a un servidor central, donde se hace una lectura y análisis de los datos para alertar al personal médico ante cualquier eventualidad. Mejorando aspectos económicos tanto para el paciente como la organización, cómo de salud y seguridad al paciente.

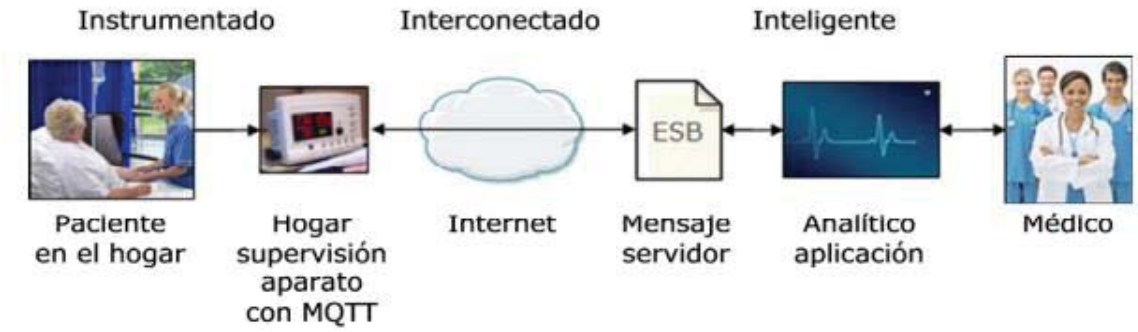

Figura 1-3 Diseño secuencia de solución domestica de marcapasos usando MQTT [1].

1.2.2 Internet de las cosas

Las aplicaciones de Internet de las cosas (IoT), se han propagado de manera considerable en el sector industrial. Una de las características más comunes de IoT es que los objetos (cosas) deben estar instrumentados, interconectados y ser procesados de manera inteligente (esto se le conoce como las 3I [instrumented, interconnected, intelligently]) en cualquier lugar, cualquier momento, de cualquier forma y cualquier modo (esto se le conoce como las 5A [anything, anywhere, anytime, anyway, and anyhow]).

Se calcula que en el año 2020 habrá en el mundo 30.000 millones de dispositivos conectados gracias al fenómeno del internet de las cosas. Para tener una idea más clara sobre la explosión tecnológica que se está viviendo, en 2009 había aproximadamente 2.500 millones, lo que muestra

una vez más el gran crecimiento tecnológico y el aumento del mercado de todo lo relacionado con IoT.

Existen diversas aplicaciones en las que podemos usar Internet de las cosas, la mayoría de estas buscan mejorar y facilitar la vida de las personas. Las de aplicaciones de IoT se pueden encontrar en áreas de:

Figura 1-4 Iconografía de algunas áreas donde se usa Internet de las cosas [5].

- Automotores: Cuando se lo vincula a IoT, el automóvil convierte los datos en una perspectiva que permite actuar, tanto dentro del automóvil como en el mundo que lo rodea.
- Energía: Mediante IoT, los innumerables dispositivos de la red eléctrica pueden compartir información en tiempo real para distribuir y manejar la energía en forma más eficiente.
- Atención médica: Desde dispositivos vestibles de uso clínico hasta tablets para servicios de emergencia y equipos quirúrgicos sofisticados, IoT está transformando los servicios de salud.
- Fabricación inteligente: La tecnología de IoT permite que las fábricas de hoy liberen la eficacia operacional, optimicen la producción y aumenten la seguridad de los trabajadores.
- Comercio minorista: Para los comerciantes minoristas, IoT ofrece oportunidades ilimitadas que aumentan la eficacia de la cadena de suministro, desarrollan nuevos servicios y rediseñan la experiencia del cliente.
- Edificios inteligentes: IoT está dando respuesta a los costos crecientes de la energía, la sustentabilidad y la conformidad con códigos conectando, administrando y asegurando los dispositivos que recopilan datos de los sistemas centrales.
- Casas inteligentes: Desde reconocer su voz hasta saber quién está en la puerta principal, la tecnología IoT está convirtiendo en realidad el sueño de una casa inteligente segura.
- Transporte inteligente: Desde automóviles conectados o de autoconducción hasta sistemas de logística y transporte inteligentes, IoT puede salvar vidas, reducir el tráfico y minimizar el impacto de los vehículos en el ambiente.

1.2.3 Android

Android constituye una pila de software pensada especialmente para dispositivos móviles y que incluye tanto un sistema operativo, como middleware y diversas aplicaciones de usuario. Representa la primera incursión seria de Google en el mercado móvil y nace con la pretensión de extender su filosofía a dicho sector.

Todas las aplicaciones para Android se programan en lenguaje Java y son ejecutadas en una máquina virtual especialmente diseñada para esta plataforma, que ha sido bautizada como Dalvik. El núcleo de Android está basado en Linux 2.6. También cuenta con un constate mejoramiento el cual se refleja en diversas versiones periódicamente.

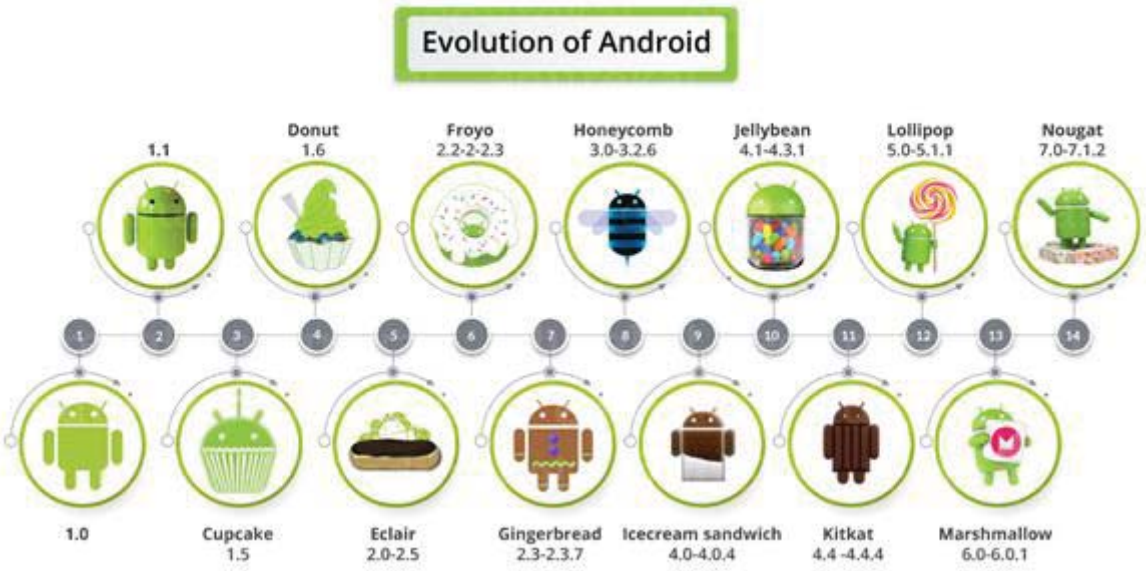

Figura 1-5 Línea evolutiva de las diversas versiones de Android [6].

La licencia de distribución elegida para Android ha sido Apache 2.0, lo que lo convierte en software de libre distribución. Gracias a lo anterior Android ha creado una red de dispositivos conectados entre sí por lo que fluye una gran cantidad de datos. Es por ello que con este sistema operativo (S.O) se ha logrado hacer una realidad el concepto de Internet de las cosas, además de hacerlo útil.

Una de las razones por las que Android está relacionado con IoT, es el hecho de que cualquier dispositivo que cuente con este S.O, puede actuar como una aplicación front-end para cualquier dispositivo o sensor capaz de emitir datos a través de internet mediante protocolos de comunicación.

Otra de las razones es que las aplicaciones Android obtendrá un mayor provecho de los dispositivos electrónicos de IoT. Además Java está muy bien integrado con protocolos de comunicación usados en Internet de las cosas.

1.2.4 Domótica

La domótica es un término usado para hacer referencia a viviendas inteligentes, esto es, al uso de las tecnologías de automatización e informática aplicadas al hogar. Busca mejorar la calidad de vida aumentada la comodidad, la seguridad, el confort y al mismo tiempo lograr ahorro energético. Dada la definición anterior, los campos de aplicación que se le están dando a la domótica se pueden ordenar en las siguientes categorías:

Figura 1-6 Ejemplo de aplicación usada en domótica [7].

- Gestión energética: Es la acción de administrar las energías que se utilizan en un hogar, esta administración se apoya en tres pilares fundamentarles que son, el ahorro energético, la eficiencia energética y la generación de energía. La domótica es importante es esos pilares ya que cuenta con la inteligencia suficiente para realizar dichas acciones.
- Confort: Cuando la vivienda se adecua por si misma a sus necesidades se mejora la calidad de vida. En este aspecto la domótica juega un papel importante ya que puede realizar tareas de forma controlada y automática. Básicamente el confort es el control de los dispositivos de iluminación, control de clima, control de aberturas, control de riego y otros más.
- Seguridad: Consiste en una red encargada de proteger a las personas y bienes de un hogar, la cual se apoya de los pilares de la prevención y detección. Como con la domótica se tiene conocimiento pleno del estado de puertas, ventanas y sensores, se puede de manera sencilla y eficiente, tomar control de dicha información y proteger el hogar.
- Comunicación: Esta aplicación es vital, ya que va de la mano con el resto de aplicaciones, si ella sería imposible conocer el estado y controlar el sistema a distancia. Lo que recae en esta aplicación es la posibilidad de conectarse con el hogar, pudiendo de esta forma controlar la vivienda a distancia y aumentar la interactividad.
- Accesibilidad: En esta aplicación la domótica busca posibilitar el acceso de cualquier persona a cualquier entorno. En la actualidad se persigue la accesibilidad universal, que se permita disfrutar de bienes o servicios según el contexto dado, con el fin de hacerlos adecuados a las capacidades, necesidades y expectativa de sus potenciales usuarios, independientemente de su edad, sexo, origen cultural o grado de capacidad.

1.3 Solución propuesta

La solución que se presenta para la problemática en este proyecto, es la utilización del protocolo MQTT como eje principal de comunicación en un dispositivo móvil inteligente con el S.O Android, mediante una App. Esta aplicación contara con una interfaz con la cual el usuario podrá monitorear diversos sensores conectados a una plataforma Raspberry Pi.

1.3.1 Protocolo de comunicación MQTT

El protocolo MQTT fue inventado por el Dr. Andy Stanford de IBM y Arlen Nipper de Arcom (ahora Eurotech) en el año 1999. Es un protocolo de conectividad abierto máquina a máquina y su funcionamiento se basa en mensajes del tipo publicar/suscribir haciéndolo extremadamente simple y liviano. Este se localiza en las capas superiores del modelo OSI [8] y se apoya en TCP/IP. Esto significa que los participantes de una aplicación MQTT deben tener una pila TCP/IP.

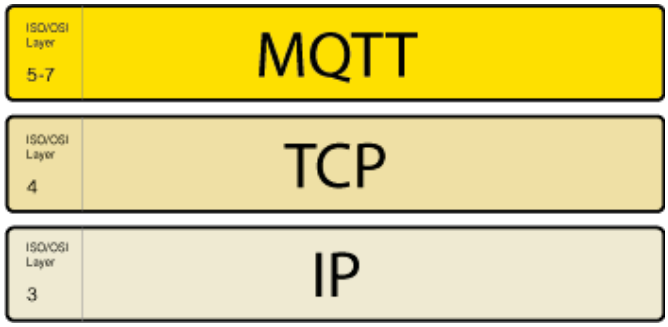

Figura 1-7 El protocolo MQTT dentro del modelo OSI.

Este protocolo presenta una serie de características que lo hacen muy atractivo para el mundo IoT y para resolver la problemática. Entre estas características se encuentran:

- Es open source, con lo cual es fácilmente integrable al universo de tecnologías, protocolos y aplicaciones de Internet de las Cosas.
- Se basa en la publicación/suscripción de mensajes, no se necesita saber para quien van los mensajes o de donde vienen, reduciendo bastante la complejidad de la red.
- Es un protocolo ligero (cabeceras reducidas, comunicación bajo demanda, etc), con lo que se ajusta a dispositivos con pocos recursos y baja velocidad de transmisión.
- Se basa en comando sencillos y cuenta con varios modos de gestión de mensajes, además, no necesita que el contenido del mensaje esté en un formato específico.
- Tiene mecanismos para garantizar una comunicación fiable, retransmitiendo o guardando para más tarde los mensajes cuando se pierda la conexión entre servidor y cliente. Para esto implementa tres niveles de calidad, lo que permite garantizar la entrega de los mensajes más importantes.

El funcionamiento del protocolo en palabras sencillas, se trata que un servidor llamado “Broker” con una capacidad hasta 10.000 clientes, se encarga de gestionar la red y de transmitir los mensajes de un “Topic” específico.

Comienza con la recopilación de los datos que los “Clientes, Publishers/Subscribers” (objetos comunicantes) envían o reciben periódicamente. Los datos pueden ser enviados a los clientes que previamente se lo hayan solicitado al servidor.

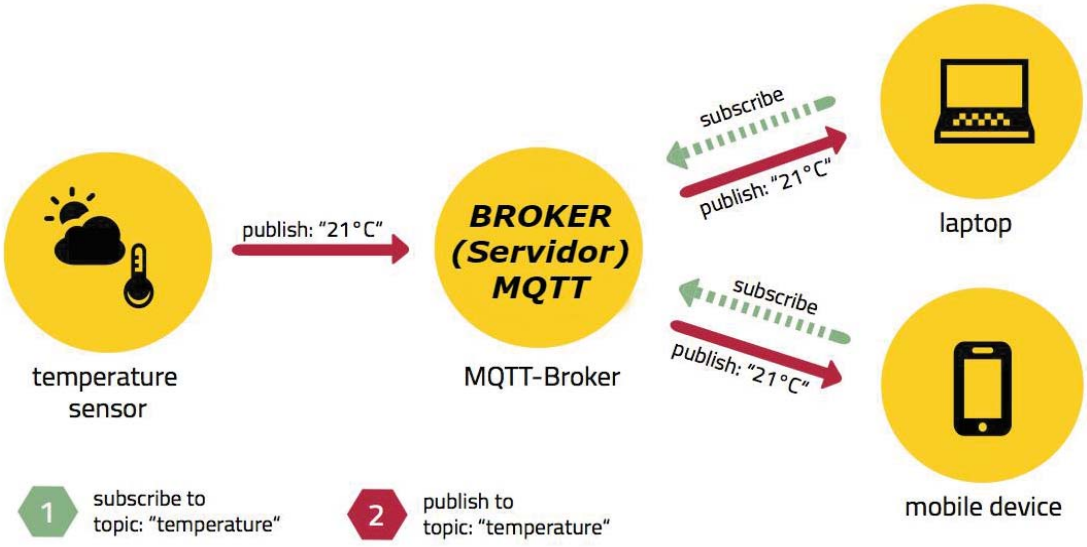

Figura 1-8 Ejemplo del funcionamiento del protocolo MQTT.

Los principales actores que participan en la red del protocolo MQTT son los clientes y servidores:

- *Los clientes MQTT*: Estos pueden abarcar un amplio rango de formatos. Pueden ser clientes MQTT que recolectan información del medio (sensores y sistemas embebidos) o aplicaciones ejecutando alguna librería MQTT y que de alguna forma interactúen con los datos. También pueden ser publicadores y subscriptores de mensajes y además, pueden controlar y configurar los sensores a su cargo mediante comandos.
- *El bróker MQTT*: Es un servicio (software) que implementa el protocolo MQTT y establece la comunicación, a nivel de aplicación, entre diferentes clientes. Hace de intermediario entre los objetos comunicantes. Además es el responsable de recibir mensajes, filtrarlos y rutarlos a los clientes subscritos según su topic. Otra tarea importante es autorizar el acceso e identificar los clientes. Pueden haber varios brokers en una misma red.

Otro aspecto importante que se debe entender del protocolo MQTT son los denominados “topics”. Estos se refieren a un tema o asusto concreto, al final, es un identificador que define el contenido del mensaje o el ámbito de influencia del mensaje (contexto). Con esto se clasifican discriminan unos mensajes de otros.

Todo aquel que quiera publicar un mensaje MQTT, es responsable de clasificarlo en un topic concreto y los clientes subscritos a ese topic, recibirán esos mensajes. La subscripción puede ser

explicita, con lo que el cliente solo recibe mensajes de un topic concreto o puede recibir mensajes de varios topics si la subscripción se hace a través de algún identificador común a varios topics, es decir, utilizando lo que se conoce como wildcards (# y +).

Los tópicos se organizan según una estructura jerárquica en árbol, utilizando el carácter "/" para formar un topic de varios niveles. Similar a la ruta de un directorio de carpetas típico en un ordenador. Entonces los topics pueden ser cadenas:

- *Multinivel*: Se utiliza el carácter "#" para admitir cualquier nivel por debajo del nivel mostrado en la cadena.
- *Simple*: Se utiliza el carácter "+" para admitir solo hasta el nivel mostrado en la cadena.

Un aspecto importante es que el publicador de los mensajes no se entera si los subscriptores están leyendo sus mensajes o no. Para solventar esto, el protocolo MQTT implementa has tres niveles de QoS (Quality of Service), garantizando la entrega de mensajes de forma correcta. Cuanto más alto el nivel, más fiable es la transmisión, pero también supone un mayor consumo de ancho de banda o una mayor latencia.

1.3.2 Plataforma Android y Smartphone

Para el desarrollo de la solución a la problemática se eligió la plataforma Android. Esta elección se fundamenta en el hecho que este sistema operático ha ido creciendo de forma vertiginosa sobre las demás plataformas desde el año 2010, en lo que respecta a dispositivos inteligentes.

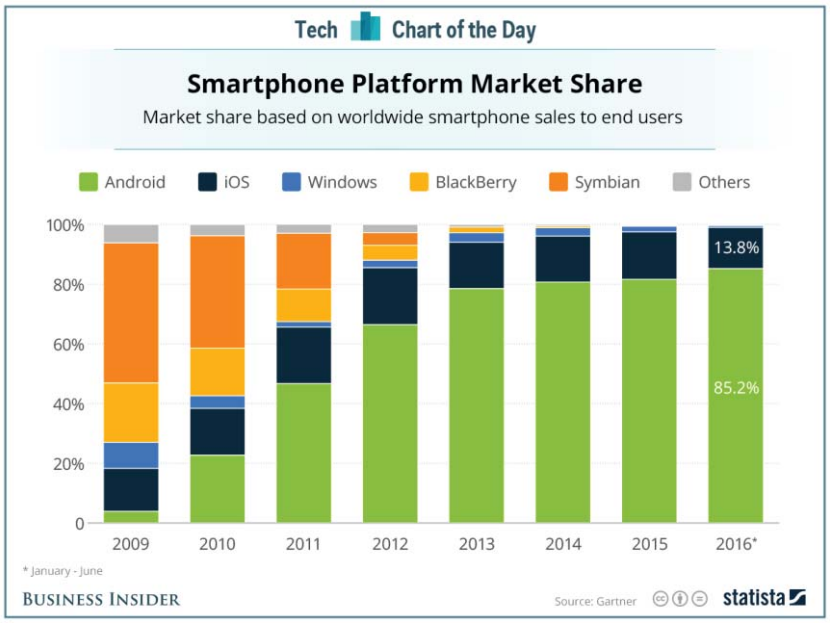

Figura 1-9 Crecimiento de la plataforma Android en el mercado de los Smartphone [9].

También la elección de trabajar con dispositivos móviles inteligentes se fundamenta principalmente en que Chile lidera el uso de estos en Latinoamérica. La cantidad de usuarios de Smartphone en nuestro país representa aproximadamente un 74.3% de la población nacional.

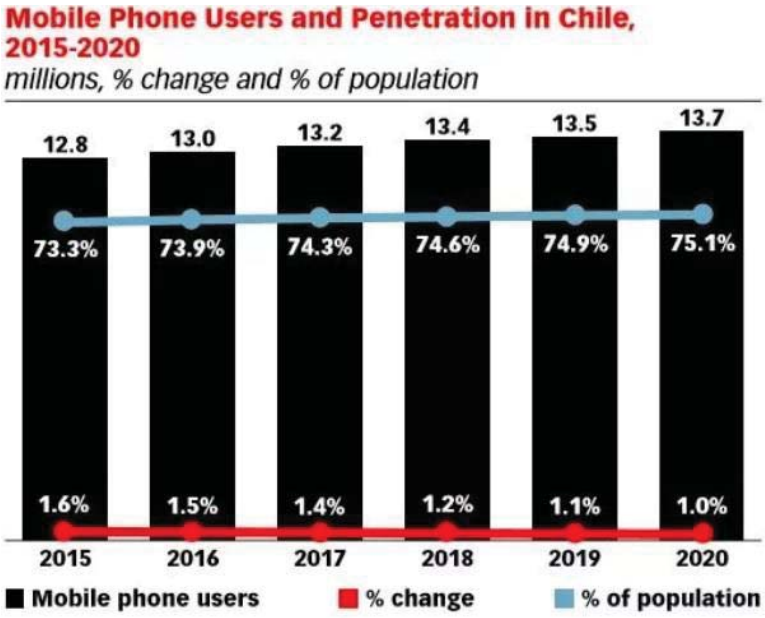

Figura 1-10 Usuarios de Smartphone y su penetración en Chile [2].

Debido a estos factores se hace clara la decisión de trabajar con dispositivos móviles inteligentes con Android. Además esta cuenta con mayor feedback, las aplicaciones de Android tienen muchas opiniones y review en Google Play, con una mayor comunidad de desarrolladores, es de sistema abierto y de fácil distribución.

1.3.3 Plataforma Raspberry Pi

Raspberry Pi es un computador de placa reducida, computador de placa única o computador de placa simple (SBC) de bajo coste. Desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

La versión a utilizar, Raspberry Pi 3 modelo B trae consigo nuevas características interesantes respecto a versiones anteriores, como por ejemplo: Procesador a 1,2 GHz de 64 bits con cuatro núcleos ARMv8, Bluetooth 4.1, Bluetooth Low Energy (BLE). Por otra parte sigue manteniendo ciertas especificaciones de versiones anteriores, como 4 puertos USB, salida HDMI, conector compuesto de audio y vídeo, espacios para insertar la microSD.

En comparación a la versión anterior cuenta con mejoras destacables, las cuales se pueden observar en la *Figura 1-11* y que sirve como resumen de sus características.

		
Raspberry Pi	RPi V2 modelo B	RPi 3 modelo B
SoC	Broadcom BCM2836	Broadcom BCM2837
CPU	900MHz Quad-core ARM Cortex-A7	1.2Ghz Quad Cortex A53
GPU	250Mhz VideoCore IV	400Mhz VideoCore IV
RAM	1Gb	1Gb
USB	4	4
Video	Jack, HDMI	Jack, HDMI
Audio	Jack, HDMI	Jack, HDMI
Boot	Memoria microSD	Memoria microSD
Wireless	No tiene	802.11n / Bluetooth 4.1
Red Ethernet	Ethernet 10/100	Ethernet 10/100
Alimentación	5V / 2Amp	5V / 2,5Amp
GPIO	40 pines GPIO	40 pines GPIO
Tamaño	85 x 56 x 17 mm	85 x 56 x 17 mm

Figura 1-11 Comparativa entre RPi V2 y RPi 3 con sus características [10].

En la *Figura 1-12* se tiene una visión superior de la placa, junto a indicadores que presentan sus principales características.

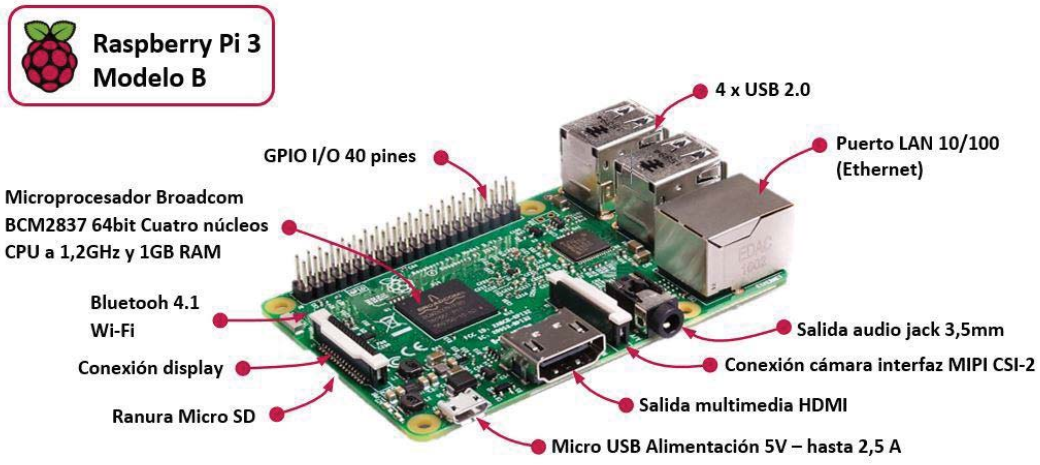

Figura 1-12 Principales características de Raspberry Pi 3 Model B [10].

Uno de los puntos más importantes a conocer de esta placa son sus Pinout GPIO (General Purpose Input Output) triestado. La conexión de este modelo cuenta con 40 pin y tiene una serie de registros que son los que controlan las entradas y salidas de propósito general (GPIO).

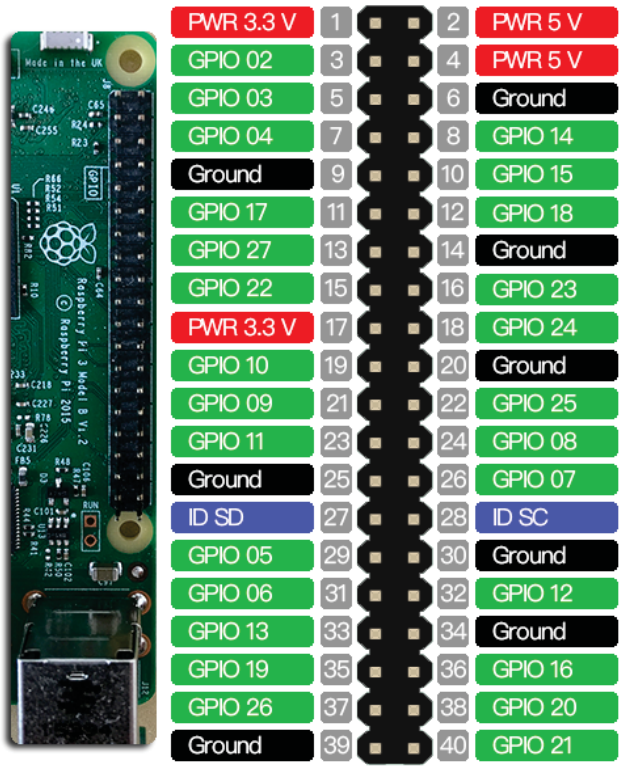

Figura 1-13 Pines de conexionado Raspberry Pi 3 [10].

Entre los 40 pines GPIO se cuenta con:

- 24 terminales de entrada/salida de propósito general, con pull up y pull down independientes programables por software.
- Un módulo UART, entrada y salida.
- Dos canales PWM con DC independiente y dos modos de operación.
- Un bus I2C.
- Dos buses SPI.
- Terminales de voltaje de 3,3 V, +5 V.
- Ocho canales de GND (Tierra).

Conocer la posición, función y nombre de cada pin GPIO es fundamental para trabajar con ellos debido, a que para su programación por software se debe precisar su GPIO o pin conectado. Esto se usara en la programación en Python para la lectura de datos en los sensores.

1.3.4 Sensores usados en domótica

En lo que a domótica respecta, los sensores son dispositivos que se encargan de darle cierto nivel de automatismo a la vivienda, de manera que se encargan de identificar una señal y enviarle la información, a una unidad de control, o directamente a los dispositivos responsables de accionar el elemento que controlan.

Dentro de la gran variedad de sensores del mercado, algunos ejemplos más comunes que se utilizan en el sector de la domótica son:

-Temperatura y humedad: Son los encargados de medir la temperatura y humedad de la vivienda o del lugar en el cual es colocado. Su función es la de enviar de forma constate la información de la temperatura y humedad para que pueda ser controlada de forma programable. El sensor que se estudia y se usara en el proyecto es el DHT11, este sensor de temperatura y humedad ya está inserto en un módulo con el circuito necesario para una conexión directa a la Raspberry Pi. Este tiene 3 pines, VCC que requiere una alimentación entre 3.3 [V] ~ 5.5 [V], DATA es la salida con los valores tomados por el sensor y GND hacia tierra del circuito montado con la Raspberry Pi.

Figura 1-14 Vista de un módulo sensor DHT11 [11].

El DHT11 es un sensor muy limitado que se usa con fines de formación, pruebas, o en proyectos. Las características que cuenta este son escazas pero cumplen con su función.

- Medición de temperatura entre 0 a 50 [ºC], con una precisión de 2 [ºC].
- Medición de humedad entre 20 a 80%, con precisión del 5%.
- Frecuencia de muestreo de 1 muestra por segundo (1 [Hz]).

-Luminosidad: Los sensores fotosensibles son ideales en domótica para el ahorro energético. Su función principal es la de detectar el brillo de la luz ambiental, enviando la información a un microcontrolador. Se utilizara un LDR (Light Dependent Resistor) con un circuito simple para la medición de la intensidad de luminosidad. Este dispositivo es una resistencia que varía su valor dependiendo de la cantidad de luz que la ilumina. El valor de la fotorresistencia no varía de forma instantánea cuando se pasa de luz a oscuridad o viceversa, el tiempo varía según el caso o intensidad lumínica. Esto hace que el LDR se pueda usar en aplicaciones que requieran mucha precisión. Este se puede utilizar para aplicaciones domóticas simples definiendo rangos de funcionamiento.

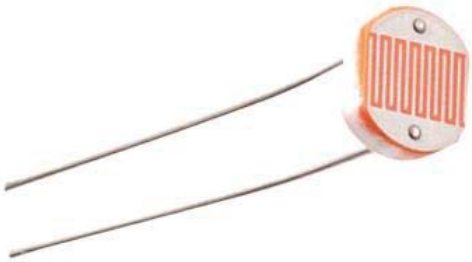

Figura 1-15 Vista de un LDR [12].

-Sensor de movimiento: Se emplean para la detección de intrusos no deseados en la vivienda, así como para automatizar ciertas funciones de la vivienda. La técnica usada por estos sensores es la detección de calor mediante radiación infrarroja que todos los seres vivos desprenden. El sensor estudiado para solventar esta tarea es el sensor detector de movimiento HC-SR501, el cual se activa al detectar movimiento de cuerpos que emiten en el espectro infrarrojo (IR) como los humanos. El módulo HC-SR501 tiene 3 pines de conexión +5 [V], OUT (3.3 [V]) y GND, y dos resistencias variables de calibración. Otras de sus características son:

- Reducido tamaño y bajo costo.
- Indiferencia a la luz natural.
- Lente fresnel de 19 zonas, ángulo < 100º
- Bajo consumo de energía y fácil manejo.
- Rango de detección: 3 a 7 [m], ajustable mediante potenciómetro.

Figura 1-16 Vista de un sensor PIR HC-SR501 [13].

-Sensor de calidad del aire: Son sensores especialmente usados para alarmas frente a la detección de fugas de gas o humo, estos se instalan en zonas del hogar que más pueden presentar este tipo de problemas, cerca de cañerías de gas, cocina, concentración de electrodomésticos, etc. El sensor estudiado es el MQ135, el cual es especialmente usado para alarmas frente a la detección de fugas de gas o humo, instalándose en zonas de riesgo en el hogar. Este se encargara de prevenir altos niveles de contaminación a nivel aeróbico y cuenta con las siguientes características a destacar:

- Señal de salida dual, de carácter analógico y digital.
- Potencia de consumo: 800 [mW].
- Concentración detectable: Amoniaco, sulfuro, benceno, humo.
- Temperatura de operación: -20~70 [ºC].

Figura 1-17 Vista de un sensor de calidad de aire MQ135 [14].

2 Estudio para el desarrollo del proyecto

En este capítulo se presentara la arquitectura del protocolo MQTT, fundamentos y arquitectura de Android, el lenguaje de programación Python, el lenguaje de programación Java, software para el desarrollo de App y la estructura detallada para desarrollo del proyecto.

2.1 Arquitectura del protocolo MQTT

La arquitectura del protocolo MQTT sigue una topología en estrella, donde existe un nodo central (Broker) con la capacidad para gestionar y ocuparse de un gran número de clientes (Publisher/Subscriber).

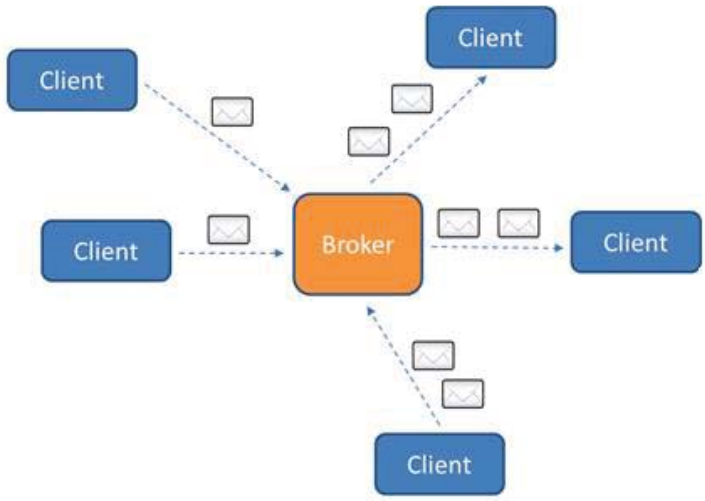

Figura 2-1 Diagrama de la arquitectura del protocolo MQTT [1].

2.1.1 Puertos utilizados

El protocolo MQTT utiliza el puerto estándar TCP/IP 1883, siendo reservado para comunicaciones sin encriptar. Los puertos 8883 y 8884 usan encriptación SSL (Secure Sockets Layer)/TLS (Transport Layer Security). El puerto 8884 requiere un certificado.

2.1.2 Seguridad

Los datos que intercambian los objetos son muy importantes, por lo que es necesario garantizar la seguridad de los intercambios. Esto lo cumple el protocolo MQTT a varios niveles:

- Transporte en SSL/TLS y autenticación mediante certificados SSL/TLS.
- Autenticación mediante usuario y contraseña.

Hay tres conceptos fundamentales en la seguridad de MQTT: identidad, autenticación y autorización, también conocido como el protocolo 3A (Authentication, Authorization and Accounting).

- *Identidad:* Puede identificar un cliente MQTT por su identificador de cliente, su ID de usuario o su certificado digital público. Uno u otro de estos atributos definen la identidad del cliente. Un servidor MQTT autentica el certificado que envía el cliente con el protocolo SSL, o la identidad del cliente con una contraseña definida por el cliente. El servidor controla los recursos que puede acceder el cliente, en base a la identidad del mismo.
- *Autenticación:* Un cliente MQTT puede autenticar el servidor MQTT al que se conecta y, a su vez, el servidor puede autenticar el cliente que se conecta a él. Un cliente autentica un servidor con el protocolo SSL. Un servidor autentica un cliente con el protocolo SSL o con la contraseña, o con ambas cosas. Si el cliente autentica el servidor pero el servidor no autentica el cliente, el cliente se conoce como 'cliente anónimo'.
- *Autorización:* La autorización no forma parte del protocolo MQTT. La proporcionan los servidores MQTT. Lo que se autoriza depende de lo que hace el servidor. Los servidores MQTT son intermediarios de publicación/suscripción, y unas útiles reglas de autorización de MQTT controlan los clientes que se pueden conectar al servidor. El número de posibles clientes es enorme, por lo que no es factible autorizar cada cliente por separado. Por lo que tendrá los medios para agrupar a los clientes por perfiles o grupos.

2.1.3 Calidad de servicios

El funcionamiento del protocolo MQTT es mediante el intercambio de una serie de paquetes de control de una manera definida. En este apartado se describe el formato de estos. El encabezado de cada paquete tiene una cabecera fija, pero no todos cuentan con una cabecera variable ni carga útil. En los paquetes de control de MQTT figuran un máximo de tres piezas.

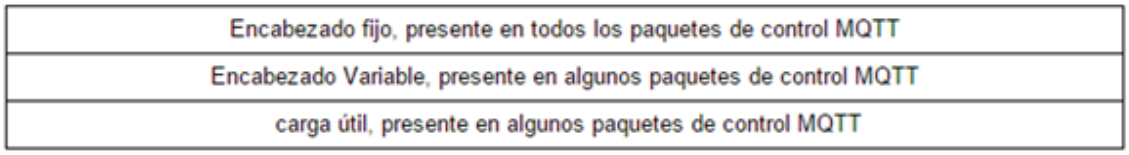

Figura 2-2 Estructura de un paquete de control MQTT [1].

2.1.4 Encabezado fijo y tipos de paquetes de control

Como se mencionó todos los paquetes de control cuentan con un encabezado fijo, el cual sigue el siguiente formato:

bit	7	6	5	4	3	2	1	0
byte 1	Tipo de mensaje				Bandera DUP	Nivel de QoS		RETENER
byte 2	Longitud restante							

Figura 2-3 Formato de una cabecera fija [1].

- *byte 1*: Contiene el tipo de paquete de control o tipo de mensaje, la bandera DUP, el nivel de QoS y la bandera retener.
- *byte 2*: Contiene el campo de longitud restante, indicando este cuantos bytes tiene en total la cabecera variable junto con la carga útil en caso de existir.

Todos los datos están ordenados desde el byte más significativo (MSB) seguido por el byte menos significativo (LSB). Es decir los bytes de orden superior preceden a los de orden inferior.

El protocolo MQTT define catorce tipos diferentes de paquetes de control, los que ocupan los bits 7, 6, 5 y 4 del byte 1 de la cabecera fija. A continuación se detalla la descripción específica de estos catorce paquetes de control:

- CONNECT: Después de establecer una conexión de red por parte de un cliente hacia un servidor, el primer paquete enviado debe ser uno del tipo *Connect*.
- CONNACK: El paquete es enviado por el servidor en respuesta a un *Connect* proveniente de un cliente. El primer paquete enviado desde el servidor hacia un cliente debe ser uno del tipo *Connack*.
- PUBLISH: Se envía desde un cliente a un servidor o de un servidor a un cliente para el transporte de un mensaje.
- PUBACK: Es la respuesta a un mensaje tipo *Publish* que ha sido enviado con un nivel de calidad de servicio (QoS) igual a 1.
- PUBREC: Es la respuesta a un paquete *Publish* que ha sido enviado con un QoS igual a 2. Es el segundo paquete de intercambio de un QoS 2.
- PUBREL: Es la respuesta a un paquete *Pubrec*. Es el tercer paquete de intercambio de un QoS 2.
- PUBCOMP: Es la respuesta a un paquete *Pubrel*. Es el cuarto y último paquete de intercambio de un QoS 2.
- SUBSCRIBE: Se envía desde cliente a servidor para crear una o más suscripciones, ya que en cada suscripción se registra el interés de un cliente en uno o más temas. El servidor envía un *Publish* al cliente con el fin de enviar mensajes que fueron publicados en temas

que responden a estas suscripciones. El paquete *Subscribe* también específica para cada suscripción las QoS máximas que el servidor puede enviar en los mensajes al cliente.

- SUBACK: Es enviado por el servidor al cliente para confirmar la recepción y el procesamiento de un paquete *Subscribe*. Un paquete *Suback* contiene una lista de códigos de retorno, que especifican el máximo nivel de calidad de servicio que se concedió en cada suscripción que se solicitó por la suscripción.
- UNSUBSCRIBE: Es enviado por el cliente al servidor y se utiliza para darse de baja a los temas suscritos previamente.
- UNSUBACK: Es enviado por el servidor al cliente para confirmar la recepción del paquete de baja *Unsubscribe*.
- PINGREQ: Se envía desde cliente a servidor. Se puede utilizar para solicitar que el servidor responda para confirmar que está vivo, o también se utiliza para indicarle al servidor que el cliente está vivo en ausencia de cualquier otro paquete de control. El cual que se envía desde el cliente al servidor.
- PINGRESP: Es enviado por el servidor al cliente en respuesta a un paquete *Pingreq*. Con esto se indica que el servidor está vivo.
- DISCONNECT: Es el paquete de control final enviado desde el cliente al servidor, con el cual se indica que el cliente está desconectado limpiamente.

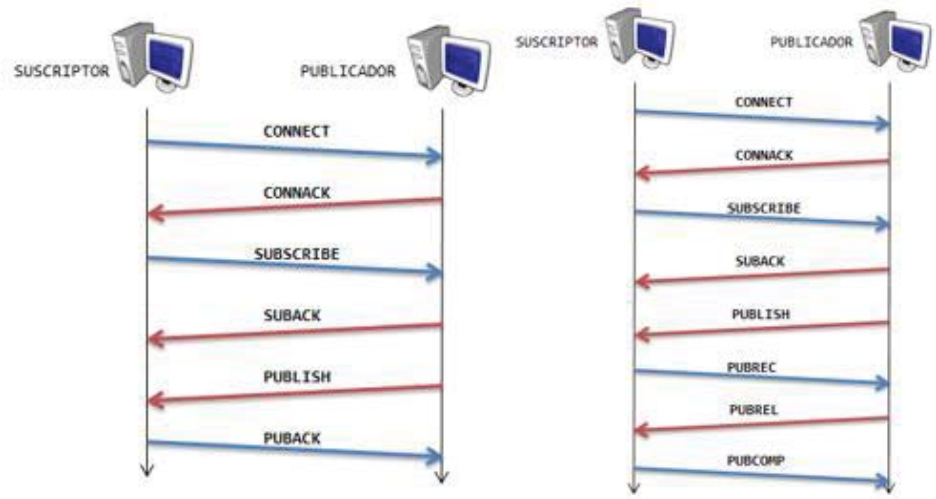

Figura 2-4 Diagrama de la secuencia de paquetes MQTT [1].

2.1.5 Encabezado variable e identificador de paquete

En la cabecera variable se encuentran algunos tipos de paquetes de control que contienen un componente de cabecera variable. Esta se ubica entre la cabecera fija y la carga útil. El contenido de este encabezado varía dependiendo del tipo de paquete de control. El campo "*Packet Identifier*" o "*Message ID*" del encabezado variable es común en varios tipos de paquetes y solo está presente en mensajes con QoS 1 y 2.

Muchos de los catorce tipos de paquetes de control incluyen 2 bytes del campo identificador de paquete (*Packet Identifier/Message ID).*

Los paquetes de control que incluyen un identificador de paquete son:

- PUBLISH (donde QoS > 0).
- PUBACK.
- PUBREC.
- PUBREL.
- PUBCOMP.
- SUBSCRIBE.
- SUBACK.
- UNSUBSCRIBE.
- UNSUBACK.

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

Figura 2-5 Estructura de Packet Identifier/Message ID [1].

Este campo sólo está presente en los mensajes donde los bits de calidad de servicio en la cabecera fija indican los niveles de calidad de servicio QoS 1 o 2. Es recomendado no utilizar un identificador de paquete para niveles QoS 0.

Cada vez que un cliente envía un paquete nuevo del tipo *SUBSCRIBE*, *UNSUBSCRIBE* y *PUBLISH* (QoS > 0) deberá asignarle un identificador de paquete que actualmente no se utilice. El cliente y servidor asignan identificadores de paquetes independientemente uno del otro. Como resultado, los pares de cliente servidor pueden participar en intercambios de mensajes simultáneos utilizando los mismos identificadores de paquetes.

2.1.6 Carga útil

Pocos paquetes de control requieren carga útil como parte final de paquete.

- *Caso PUBLISH*: Su carga útil corresponde al mensaje de aplicación en sí.
- *Caso SUBSCRIBE*: Su carga útil contiene una lista de nombres de los temas a los que el cliente se ha suscrito, y el nivel de calidad de servicio con la que el cliente requiere para recibir los mensajes.
- *Caso SUBACK*: Su carga útil contiene una lista con los niveles de QOS concedidos por el bróker al cliente para subscribirse a un determinado tópico.

2.2 Fundamentos y arquitectura de Android

Android constituye una pila de software pensada especialmente para dispositivos móviles y que incluye tanto un sistema operativo, como middleware y diversas aplicaciones de usuario. Representa la primera incursión seria de Google en el mercado móvil y nace con la pretensión de extender su filosofía a dicho sector.

La decisión de la utilización de este sistema operativo, frente a sus principales competidores, iOS y Windows Phone se presentó en el capítulo anterior pero es importante conocer las características, ventajas y desventajas de cada uno de estos. Por lo que se presentan dos tablas comparativas entre los distintos sistemas operativos.

Tabla 2-1 Comparativa de características Android, iOS y Windows Phone.

Sistema Operativo	**Android**	**iOS**	**Windows Phone**
Kernel	Linux	OS X	Windows NT
Lenguaje Programación	Java	Objective C	C#
Seguridad	Muy Buena	Susceptible a Malware	Muy Buena
Costos de Desarrollo	25 USD Solo una vez	99 USD Anuales	99 USD Anuales
Estándares soportados	GSM y CDMA	GSM y CDMA	GSM y CDMA
Hardware soportado	Amplia gama de dispositivos	iPhone, iPad y iPod	Limitada gama de dispositivos
Apps	+2.800.000	+2.200.000	+650.000

Tabla 2-2 Ventajas y desventajas de Android, iOS y Windows Phone.

	Ventajas	**Desventajas**
Android	✓ Open Source. ✓ Integrado a IoT ✓ Cuenta con más del 80% del mercado ✓ Mayor feedback y review ✓ Gran comunidad de desarrolladores ✓ Fácil distribución	× Fragmentación por las distintas versiones del S.O × Poco intuitivo
iOS	✓ Buen diseño y funcionalidad ✓ Sistema intuitivo ✓ Perfecta integración con servicios en la nube.	× Sistema cerrado × Control rígido de aplicaciones × Altos costos de uso × Solo existe un fabricante
Windows Phone	✓ Diseño moderno ✓ Practico y atractivo en su uso ✓ Características innovadoras	× S.O joven × Todavía está en crecimiento

La arquitectura de Android está formada por varias capas, como se aprecia en la *Figura 2-6*.

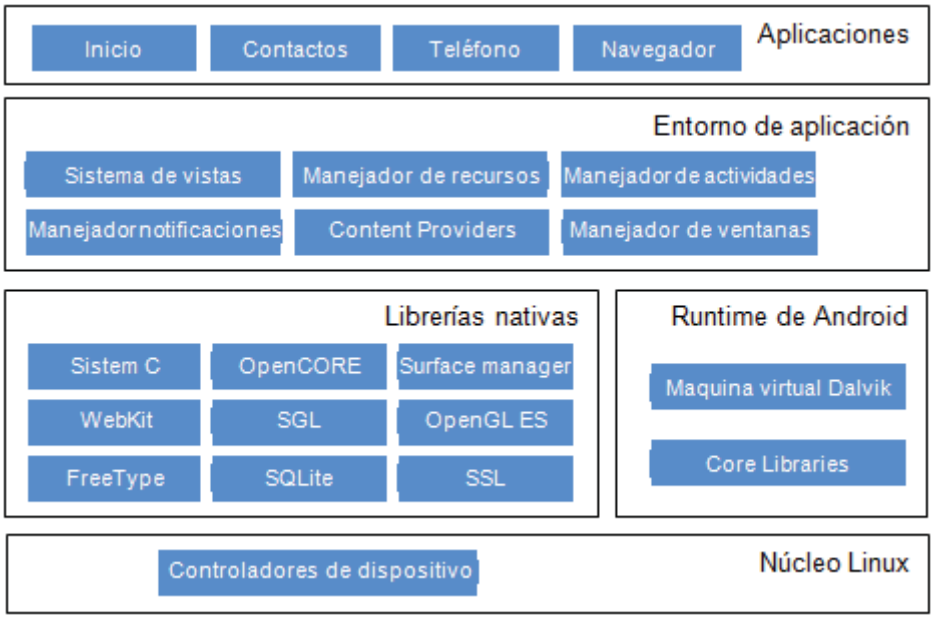

Figura 2-6 Arquitectura Android [15].

2.2.1 Nucleo Linux

Este está formado por el S.O Linux versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el de soporte de drivers para dispositivos. También actual como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del hardware.

2.2.2 Runtime de Android

Está basado en el concepto de máquina virtual utilizando Java, Google creo su propia máquina virtual estándar conocida como Dalvik. Esta máquina virtual delega al kernel de Linux algunas funciones como threading y el manejo de la memoria a bajo nivel. También incluye una serie de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas del lenguaje Java.

2.2.3 Librerías nativas

Incluye un conjunto de librerías usadas en varios componentes de Android. Están compiladas en código nativo del procesador. Muchas librerías utilizan proyectos de código abierto, algunas de estas librerías son:

- *System C library*: Una derivación de la librería BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.

- *Media Framework*: Librería basada en OpenCORE de PacketVideo. Soporta codecs de reproducción y grabación de multitud de formatos de audio y vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- *Surface Manager*: Maneja el acceso al subsistema de representación gráfica en 2D y 3D.
- *WebKit/Chromium*: Soporta un moderno navegador Web utilizado en el navegador Android y en la vista Webview.
- *SGL*: Motor de gráficos 2D.
- *Librerías 3D*: Implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador harware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- *FreeType*: Fuentes en bitmap y renderizado vectorial.
- *SQLite*: Potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- *SSL*: proporciona servicios de encriptación Secure Socket Layer (capa de conexión segura).

2.2.4 Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones. Diseñada para simplificar la reutilización de componentes. Además aprovecha el lenguaje de programación Java. Los servicios más importantes son:

- *Views*: Extenso conjunto de vistas, (parte visual de los componentes).
- *Resource Manager*: Proporciona acceso a recursos que no son en código.
- *Activity Manager*: Maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
- *Notification Manager*: Permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- *Content Providers*: Mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

2.2.5 Aplicaciones

Este nivel está formado por el conjunto de aplicaciones instaladas en un maquina Android. Todas las aplicaciones se han de ejecutar en la máquina virtual Dalvik para garantizar la seguridad del sistema. Las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

Tanto en la Raspberry Pi, como en el sistema operativo Android todas sus aplicaciones se programan en distintos lenguajes de programación, es por ello que se estudia en el siguiente punto los dos principales, Python y Java.

2.3 Lenguajesde programación Python

Python es un lenguaje de programación poderoso que fue creado a principios de los años '90. Este lenguaje cuenta con estructuras de datos eficientes, de alto nivel y enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto a su naturaleza interpretada, hacen de este un lenguaje ideal para programación y desarrollo de aplicaciones en diversas áreas y plataformas.

Python es un lenguaje extensible para nuevas funcionalidades, cuenta con una gran biblioteca de módulos y rica librería distribuida de forma libre. Además su importancia se eleva actualmente al estar entre los cuatro lenguajes más populares y ser el tercero en generar mayores ingresos.

Figura 2-7 Diagrama con los 10 lenguajes más populares y su importancia en el mercado [16].

2.3.1 Estructura y elementos del lenguaje

Como se ha mencionado Python es clasificado como un lenguaje interpretador, de alto nivel, multiplataforma, de tipado dinámica y multiparadigma. A diferencia de la mayoría de los lenguajes de programación, Python provee ciertas reglas de estilos, a fin de poder escribir código fuente más legible y de manera estandarizada.

Python se compone de una serie de elementos que alimentan su estructura. Entre ellos podemos encontrar los siguientes:

- *Variables*: Es un espacio para almacenar datos modificables en la memoria de un ordenador. En Python, una variable se define con la siguiente sintaxis:
  nombre_de_la_variable = valor_de_la_variable
  Se recomienda utilizar nombres descriptivos y en minúsculas, separando nombres compuestos con guiones bajos.
- *Tipos de datos*: Una variable puede contener valores de diversos tipos como una cadena de texto (string), número entero, octal, hexadecimal, real, booleano (True/False), entre otros más complejos.

- *Operadores aritméticos*: En estos siempre se coloca un espacio en blanco, antes y después de un operador, de los cuales en Python podemos encontrar:

Símbolo	Significado
+	Suma
-	Resta
-	Negación
*	Multiplicación
**	Exponente
/	División
//	División entera
%	Módulo

Figura 2-8 Operadores aritméticos en Python.

- *Comentarios*: Son notas que como programador se indican en el código para poder comprenderlo mejor. Estos pueden ser de una solo línea (mediante el símbolo #) separados por dos espacios en blanco del código y luego del símbolo # debe ir un solo espacio en blanco.
- *Estructura de control de flujo de condicionales*: Son aquellas que permiten evaluar si una o más condiciones se cumplen, para declarar que acción se va a ejecutar. La evaluación de condiciones solo puede arrojar un resultado, verdadero o falso. Para describir la evaluación a realizar sobre una condición, se utilizan operadores relacionales:

OPERADORES RELACIONALES (DE COMPARACIÓN)

Símbolo	Significado	Ejemplo	Resultado
==	Igual que	5 == 7	Falso
!=	Distinto que	rojo != verde	Verdadero
<	Menor que	8 < 12	Verdadero
>	Mayor que	12 > 7	Falso
<=	Menor o igual que	12 <= 12	Verdadero
>=	Mayor o igual que	4 >= 5	Falso

Figura 2-9 Operadores relacionales en Python.

Y para evaluar más de una condición simultáneamente, se utilizan operadores lógicos:

OPERADORES LÓGICOS

Operador	Ejemplo	Resultado	
and (y)	5 == 7 and 7 < 12	0 y 0	Falso
	9 < 12 and 12 > 7	1 y 1	Verdadero
	9 < 12 and 12 > 15	1 y 0	Falso
or (o)	12 == 12 or 15 < 7	1 o 0	Verdadero
	7 > 5 or 9 < 12	1 o 1	Verdadero
xor	4 == 4 xor 9 > 3	1 o 1	Falso
(o excluyente)	4 == 4 xor 9 < 3	1 o 0	Verdadero

Figura 2-10 Operadores lógicos en Python.

Estas estructuras de control de flujo condicionales, de definen mediante el uso de tres palabras claves reservadas del lenguaje: *if* (si), *elif* (sino, sí) y *else* (sino).

- *Estructuras de control iterativas*: Las estructuras de control iterativas, también llamadas cíclicas o bucles, permiten ejecutar un mismo código de manera repetida mientras se cumpla una condición. En Python se dispone de dos estructuras cíclicas, el bucle "*while*" que se encarga de ejecutar una misma acción "mientras que" una determinada condición se cumpla y el bucle "*for*" que permite iterar sobre una variable compleja.

2.3.2 Módulos, paquetes y funciones

En Python cada archivo *.py* se denomina módulos, que a su vez pueden formar paquetes. Los paquetes a su vez pueden contener sub-paquetes y los módulos, no necesariamente deben pertenecer a un paquete. El contenido de cada módulo, podrá ser utilizado a la vez por otros módulos. Para ello, es necesario importar los módulos que se quieran utilizar, para importar un módulo se utiliza la instrucción "*import"*, seguida del nombre del paquete (si aplica) más el nombre del módulo sin el punto de extensión que se desee importar. Python tiene sus propios módulos, los cuales forman parte de su librería de módulos estándar que también pueden ser importados.

La definición de funciones en Python se realiza mediante la instrucción *"def"* más un nombre de función descriptivo, seguido de paréntesis de apertura y cierre que finaliza con dos puntos (:) y el algoritmo que lo compone.

Otro aspecto básico a considerar en la programación en Python son las llamadas de retorno, que llaman a una función dentro de otra de forma fija y de la misma manera en que se llamaría desde fuera de dicha función. Esto se realiza con la instrucción *"returnt"*. Además de considerar la instrucción "*print*" la cual imprimirá en pantalla la información requerida por programa creado. Estos aspectos son fundamentales para el desarrollo de programas en Python.

2.4 Lenguaje de programación Java

Java es un lenguaje de desarrollo de propósito general, y como tal es válido para realizar todo tipo de aplicaciones móviles profesionales. El lenguaje Java es un derivado del lenguaje C, por lo que sus reglas de sintaxis se parecen mucho a C. Como cualquier lenguaje de programación, Java tiene su propia estructura de sintaxis y paradigma de programación. Este paradigma se basa en el concepto de programación orientada a objetos (OOP), que las funciones del programa soportan.

Estructuralmente, el lenguaje Java comienza con paquetes. Un paquete es el mecanismo de espacio de nombres del lenguaje Java. Dentro de los paquetes se encuentran las clases y dentro de las clases se encuentras métodos, variables, constantes, entre otros. Algunas características importantes que posee son:

- *Compilado:* Generando ficheros de clases compilados, pero estas clases compiladas son en realidad interpretadas por la máquina virtual Java.
- *Multiplataforma:* El mismo código Java que funciona en un sistema operativo funcionará en cualquier otro sistema operativo que tenga instalada la máquina virtual Java.
- *Interpretativo*: El intérprete Java (sistema run-time) puede ejecutar directamente el código objeto.
- *Portable*: Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas.
- *Seguro*: La máquina virtual al ejecutar el código java realiza comprobaciones de seguridad, además el propio lenguaje carece de características inseguras, como por ejemplo los punteros.

Por lo mencionado anteriormente Java es el lenguaje de programación más popular orientado a objetos, es decir, se utilizan imágenes, botones, cuadros de texto y tablas con las cuales se facilita el desarrollo y uso de distintos programas. Este incluye una serie de características que lo hacen único, y está siendo adoptado por una multitud de fabricantes como herramienta básica para el desarrollo de aplicaciones comerciales de gran repercusión.

2.5 Herramienta para el desarrollo de Apps

Para el desarrollo de aplicaciones en Android existen una serie de herramientas y software disponibles. Entre la gran variedad de estas, hay una que se destacan sobre todas las otras, el software Android Studio [17].

Antes de presentar este entorno de desarrollo es necesario conocer el concepto de IDE. Es un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. Consiste de un editor de código fuente, herramientas de construcción automática y un depurador, algunos contienen un compilador, un intérprete o ambos.

2.5.1 Android Studio

Es el entorno de desarrollo integrado oficial para la plataforma Android desarrollado por Google. Este entorno de trabajo salió a la luz en el año 2013, y reemplazo a Eclipse como IDE oficial para el desarrollo de aplicaciones en Android. Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Se encuentra disponible para los sistemas operativos Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android y actualmente está disponible la versión estable 3.0 lanzada en octubre de 2017.

Figura 2-11 Logo de Android Studio.

Android Studio proporciona las herramientas más rápidas para crear apps en todas las clases de dispositivos Android. La edición de códigos de primer nivel, la depuración, las herramientas de rendimiento, un sistema de compilación flexible y un sistema instantáneo de compilación e implementación permiten concentrar el esfuerzo en la creación de aplicaciones únicas y de alta calidad. Los aspectos a destacar de este entorno de desarrollo son:

- *Instan Run*: Introduce cambios en el código y los recursos para tu app en ejecución en un dispositivo o emulador, y visualiza estos cambios al instante. Instant Run acelera notablemente la edición, compilación y ejecución.

- *Editor de código inteligente*: Se logra escribir un mejor código, trabajo más rápido y más productivo gracias a un editor de código inteligente que ayuda en cada paso. Android Studio se basa en IntelliJ y permite completar, re-factorizar y analizar código de manera avanzada.
- *Emulador rápido y cargado de funciones*: Cuenta con un nuevo Android Emulator 2.0 que es más rápido de manera dinámica, permite cambiar el tamaño del emulador y acceder a un conjunto de controles de sensores. Se puede instalar y ejecutar apps más rápido con un dispositivo físico, y probarlas en casi cualquier configuración de dispositivos Android.
- *Sistema de compilación sólido y flexible*: Configura de manera sencilla proyectos de modo que incluya bibliotecas de códigos y genera diferentes variantes de compilación desde un único proyecto. Con Gradle, Android Studio ofrece automatización de compilaciones de alto rendimiento, administración sólida de dependencias y configuraciones de compilación personalizables.
- *Desarrollo para todos los dispositivos Android*: Proporciona un entorno unificado que permite desarrollar apps para teléfonos y tablets Android, y dispositivos Android Wear, Android TV y Android Auto. Además de compartir el código de manera sencilla.
- *Plantillas de código e integración con GitHub*: Inicia proyectos con plantillas de código para patrones como el panel lateral de navegación y los paginadores de vistas, o importa ejemplos de código de Google desde GitHub.

Los requisitos para ejecutar este software (versión 3.x) para cada sistema operativo compatible se presentan en la *Tabla 2-3* y la interfaz con las funciones de este en la *Figura 2-12*.

Tabla 2-3 Requisitos mínimos y recomendados de Android Studio.

	Windows	**OS X/macOS**	**Linux**
OS versión	Windows 10/8/7 (32- o 64-bit)	Mac OS X 10.10 o superior	GNOME o KDE desktop
RAM	3 GB RAM mínimo, 8 GB RAM		
Espacio en disco	2 GB de espacio en disco para Android Studio, 4GB recomendados		
Java versión	Java Development Kit (JDK) 8		
Resolución pantalla	1280x800 mínimo, 1440x900 recomendado		

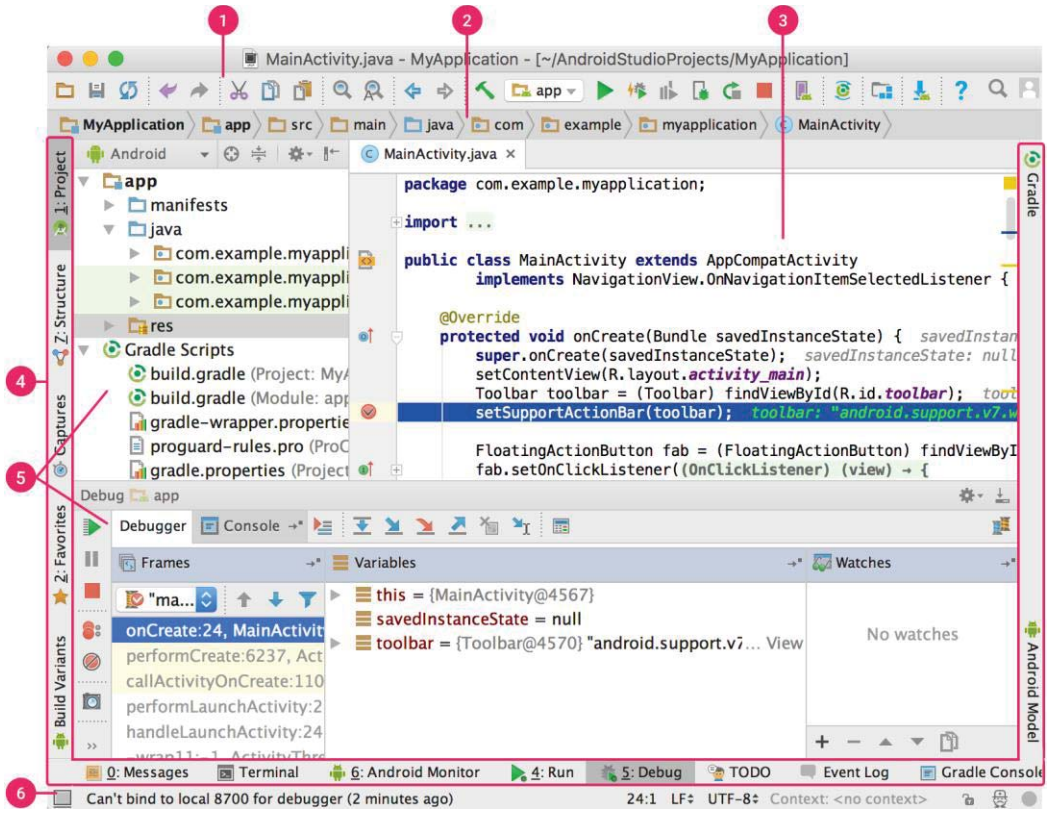

Figura 2-12 Interfaz de trabajo de Android Studio.

1) La *barra de herramientas* permite realizar una gran variedad de acciones, como la ejecución de app y el inicio de herramientas de Android.
2) La *barra de navegación* ayuda a explorar el proyecto y abrir archivos para editar. Proporciona una vista más compacta de la estructura visible en la ventana *Project*.
3) La *ventana del editor* es el área donde se crea y modifica el código. Según el tipo de archivo actual, el editor puede cambiar. Por ejemplo, cuando se visualiza un archivo de diseño, el editor muestra el editor de diseño.
4) La barra de la *ventana de herramientas* se extiende alrededor de la parte externa de la ventana del IDE y contiene los botones que permiten expandir o contraer ventanas de herramientas individuales.
5) Las *ventanas de herramientas* permiten acceder a tareas específicas, como la administración de proyectos, las búsquedas, los controles de versión, etc. Se pueden expandir y contraer.
6) En la *barra de estado*, se muestra el estado del proyecto y del IDE en sí, como también cualquier advertencia o mensaje.

2.6 Estructura detallada propuesta para el desarrollo del proyecto

En este punto se planteara la estructura funcional con los principales elementos, tecnologías, herramientas y dispositivos a usar. Se busca que mediante una App se pueda monitorear distintos sensores puestos en un lugar en específico.

Esto tanto de manera global, permitiendo leer los datos en tiempo real desde cualquier parte del mundo, siempre y cuando se cuente con un dispositivo móvil con la App y conexión a internet. Además se debe poder monitorear los sensores de manera local, sin la necesidad de tener una conexión a internet pero si contar con la red local mediante un router y un dispositivo móvil con la App.

Por lo mencionado, se hará un estudio para lograr una comunicación local, global y poder hacer un puente comunicativo entre estos. Posteriormente se presenta la estructura que se pretende lograr.

2.6.1 Servidor local Mosquitto

Mosquitto [18] es un broker Open Source ampliamente utilizado debido a su ligereza, lo que nos permite fácilmente, emplearlo en gran número de ambientes, incluso si éstos son de pocos recursos. Pero todo esto solo de manera local, no soporta

Figura 2-13 Logo del servidor local Mosquitto.

Mosquitto se puede utilizar como servicio en diversos sistemas operativos, lo más común es usarlo en sistemas basados en Linux. Este implementa la versión 3.1 y 3.1.1 (fase de pruebas) del protocolo de comunicación MQTT. Este bróker además brinda dos clientes importantes, un subscriptor y un publicador.

-Comando mosquitto sub: Este comando permite subscribirnos a un topic en específico, escuchando y mostrando en pantalla los mensajes recibidos en este topic. Este comando, al igual que el siguiente, puede tomar muchos parámetros pero solo se estudiaran los más importantes.

mosquitto_sub -h host –p puerto–u usuario –P contraseña-t tópico

-Comando mosquitto_pub: Este comando permite publicar a un topic en específico, escuchando y mostrando en pantalla los mensajes recibidos en este topic.

mosquitto_pub -h host –p puerto–u usuario –P contraseña-t tópico –m “mensaje”

Donde:

- *-h*: Indica que lo que viene después es el host. El host se refiere a la dirección IP o nombre de la maquina en la red del bróker Mosquitto. Si no se define se usa por defecto la dirección local (localhost).
- *-p*: Indica que lo que viene después es el puerto. Se pueden utilizar distintos puertos para este servidor, si no se configura se usara el puerto 1883 por defecto.

- *-u*: Indica que lo que viene después es el nombre del usuario, este por defecto no está definido pero mediante configuración se le puede agregar un usuario y contraseña específico. Esto brinda más seguridad a la red local del servidor.
- *-P*: Indica que lo que viene después es la contraseña de un usuario en específico. Si la contraseña no es correcta no se puede acceder a los servicios de Mosquitto.
- *-t*: Indica que lo que viene después es el topic al que se quiere subscribir/publicar. Se debe indicar el nombre del tópico y sus respectivos multi-niveles si se tienen.
- *-m*: Indica que lo que viene después es el mensaje que se quiere publicar. Este debe ir entre comillas dobles.

2.6.2 Servidor en la nube CloudMQTT

Como se requiere una comunicación de forma global, es necesario utilizar otro tipo de bróker, uno que esté disponible 24/7 en la nube. Para esta función existen muchas alternativas tanto gratuitas como de pago, y entre todas estas se buscó la que mejor se adapta al proyecto. Este es CloudMQTT [19] el cual brinda distintos planes, lo que hace a este bróker es escalable.

El plan que se utilizara para el proyecto es el gratuito “Cute Cat” limitado a cinco conexiones, lo que es suficiente para este proyecto. Para poder utilizar este servidor es necesario crear una cuenta, siendo posible usar las credenciales de google haciendo esto rápido y sencillo.

Creada la cuenta este nos brinda toda la información necesaria para utilizarlo. Como la dirección de servidor, un usuario, su respectiva contraseña y puerto a utilizar.

Tabla 2-4 Datos proporcionados por el servidor CloudMQTT.

Servidor	Puerto	Usuario	Password
m14.cloudmqtt.com	19165	cjkjjze	WFJKWObn6WqQ

2.6.3 Node RED

Node-RED [20] es una herramienta muy potente que sirve para comunicar hardware y servicios de una forma rápida y sencilla. Simplifica enormemente la tarea de programar del lado del servidor gracias a la programación visual. Fue creada por dos ingenieros del grupo de Servicios de Tecnologías Emergentes de IBM en el año 2013, estos mismos están trabajando también en el protocolo de comunicación MQTT de IBM, por lo que Node-RED está completamente ligado a MQTT.

La estructura mínima son los nodos, estos se arrastran a través de la interfaz gráfica y nos permiten hacer una tarea concreta. Estos nodos se organizan en flujos o flows que agrupan nodos que se conectan entre ellos. Todo de una forma visual sin apenas tener que programar. Node-RED esta creado a partir de NodeJS, que proporciona la potencia suficiente para que Node-RED sea fiable y escalable.

Su ventaja más importante es que esta optimizado para poder tratar múltiples conexiones concurrentes de una manera óptima. Es el mayor ecosistema de código abierto que existe en el mundo y está siendo utilizado por empresas punteras como Paypal y Netflix.

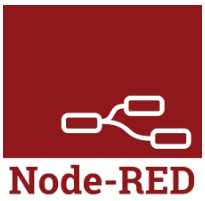

Figura 2-14 Logo de la herramienta Node-RED.

Node-RED es accesible a través de un navegador, es decir, solo se necesita acceder a una página web para poder crear nuestros propios sistemas. No hace falta instalar ningún entorno de desarrollo o IDE. En la aplicación web se pueden arrastrar y conectar los nodos con diferentes funcionalidades para crear flujos de datos que hagan una determinada tarea.

Nuevamente se debe recalcar que a mayor medida todo lo que se realiza en Node-RED, es sin tener que programar una línea de código. Esto no quiere decir que no se deba conocer sobre programación, al contrario. Se requieren unos altos conocimientos de lógica computacional para desarrollar proyectos con Node-RED.

Con esta herramienta se pretende hacer un nexo entre ambos servidores MQTT. También se debe destacar las ventajas de Node-RED y que lo hacen compatible con el proyecto.

- Es de código abierto y esta mantenido por IBM.
- Soporta y está pensado para la utilización del protocolo de comunicación MQTT.
- Se puede instalar en cualquier sistema operativo incluyendo Raspbian.
- Evita tener que profundizar en tecnologías complejas difíciles de implementar y programar.
- No requiere de un entorno de desarrollo o IDE.
- Permite crear prototipos muy rápidamente, centrándose en la tarea que se quiere lograr sin perder mucho tiempo.

2.6.4 Librería Paho

Paho [21] es un esfuerzo de la fundación Eclipse por tener librerías para MQTT en cualquier lenguaje de programación, incluyendo Java y Python. Las librerías que originaron el proyecto le pertenecían a IBM, que donó el código a la fundación Eclipse y todavía lo financia económicamente. Este proyecto proporciona implementaciones de cliente de código abierto de los protocolos de mensajería MQTT y MQTT-SN destinados a aplicaciones nuevas, existentes y emergentes para Internet de las cosas (IoT).

Existen tres aspectos a destacar de esta librería, las cuales se nombran y detallan a continuación.

- Pensado para redes restringidas: Los sistemas de IoT deben lidiar normalmente con interrupciones de red o trabajar con redes intermitentes, lentas o de baja calidad. La pérdida mínima de datos es crucial en redes con millones de dispositivos conectados.
- Dispositivos y plataformas integradas: A menudo los dispositivos y servidores frontera de la red (edge-of-network) tienen disponible recursos para el procesamiento limitados. Paho entiende los clientes pequeños y el soporte necesario requerido.
- De confianza: Paho se centra en implementaciones confiables que se integren adecuadamente con una amplia gama de middleware, programación y modelos de mensajería.

2.6.5 Estrucura a desarrollar

Como parte final de este capítulo se presenta un esquema de la estructura que seguirá el desarrollo del proyecto. Esto en base a cada uno de los puntos estudiados a lo largo del capítulo.

En este se indica la conexión entre cada uno de los elementos, donde la comunicación será mediante el protocolo MQTT. Los sensores estarán conectados a la Raspberry Pi, en la cual también contara con los programas en Python necesarios para la lectura de los datos de los sensores y que estos se conecten al servidor local Mosquitto, pudiendo realizar publicación y subscripción de tópicos. El servidor local al igual que la herramienta Node-RED se ejecutaran en la Raspberry Pi, a su vez Node-RED hace de puente entre Mosquitto y CloudMQTT, permitiendo que este último reciba todos los mensajes que publican los sensores. La App desarrollada en Java debe poder acceder a los datos tanto de forma local como global por lo que se conectara a ambos servidores.

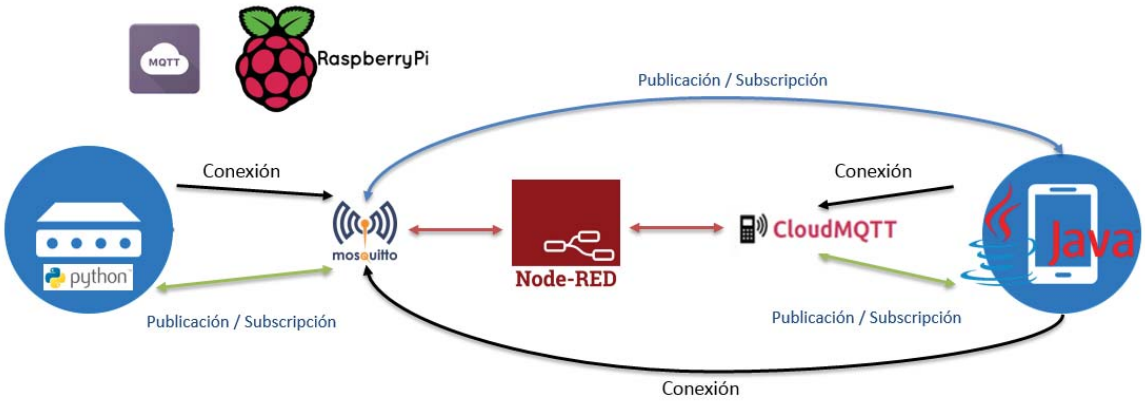

Figura 2-15 Esquema estructural del proyecto.

3 Desarrollo en la Raspberry Pi

Este capítulo contiene todo el desarrollo del proyecto que involucra a la placa Raspberry Pi. Las configuraciones necesarias, los servicios, conexiones y programas desarrollados.

3.1 Instalación y configuración de Mosquitto

En la Raspberry Pi conectada a internet, se abre un terminal donde se escriben los siguientes comandos para la instalación del bróker local Mosquitto.

Listado 3-1 Comandos para la instalación de Mosquitto.

```
1 sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa
2 sudo apt-get update
3 sudo apt-get install mosquitto
4 sudo apt-get install mosquitto-clients
```

Hecho esto se contara con el bróker instalado, este servicio se puede detener, comenzar o reiniciar usando los siguientes comandos en un terminal.

Listado 3-2 Comandos para usar el servicio de Mosquitto.

```
1 sudo service stop mosquitto
2 sudo service start mosquitto
3 sudo service mosquitto restart
```

Ahora se procede a la configuración del servicio de este bróker local. Para ello se debe modificar las configuraciones por defecto de Mosquitto por unas personalizadas. Las configuraciones por defecto se encuentran en el siguiente directorio "/etc/mosquitto/mosquitto.conf", pero este no se modificara ya que sirven como respaldo ante cualquier inconveniente. En cambio se copiara esta configuración al directorio "/etc/mosquitto/conf.d/", a este archivo se le escribirá la configuración personalizada para el bróker.

Entre las configuraciones se puede establecer un usuario con una respectiva contraseña, el puerto a utilizar, el máximo de mensajes en cola, el límite de tamaño de los mensajes, retener o no los mensajes, entrar con un usuario anónimo entre otros.

Cada uno de los parámetros que se pueden configurar se encuentra detallado en la página oficial de Mosquitto proporcionado por la fundación Eclipse [18]. La configuración que se utilizara para el proyecto es la siguiente.

Listado 3-3 Configuración mosquitto.conf.

```
user domqttica
max_queued_messages 200
message size limit 0
allow_zero_length_clientid true
allow_duplicate_messages false

listener 1883
autosave_interval 900
autosave on changes false
persistence true
persistence file mosquitto.db
allow_anonymous false
password_file /home/pi/MQTT/password.txt
```

Se debe que notar que se creó un directorio para alojar las contraseñas usadas por los usuarios, para así poder acceder de forma privada a este servidor local. En el archivo "password.txt" se escribe el nombre del usuario y contraseña usando la siguiente sintaxis: USUARIO:CONTRASEÑA separando de forma vertical los distintos usuarios que se quieran agregar. En este caso se crearon dos usuarios con sus respectivas contraseñas.

Listado 3-4: Contenido del archivo de texto con el usuario y contraseña sin encriptar.

```
domqttica:eie655
prueba:12345
```

Se hace necesario que las contraseñas de cada usuario estén encriptadas y que Mosquitto las pueda reconocer, para ello se escriben los siguientes comandos en un terminal.

Listado 3-5: Comandos para encriptar la contraseña del usuario y sea reconocido por Mosquitto.

```
cd /home/pi/MQTT/
mosquitto_passwd -U password.txt
```

Con esto Mosquitto reconocerá solo a los usuarios que estén en este archivo con su respectiva contraseña. Además encriptara estas para que no sean visibles con facilidad.

Listado 3-6: Usuario y contraseña encriptados para usar el servicio Mosquitto.

```
domqttica:$6$HBgffhG3/tLVOTJ5NO5l+/uvUpaJmcKrHsSu/nSRaoSh/PGkztH0dtt+/lVuODp2pJw==
prueba:$6$T00vZUA6W8FtVqN$7C5Fi6r8/XIO8mXQzwkXgsBSt6StouHhlLAvGVqFSVRDOPMeUuRcw==
```

Realizada la instalación y configuración de Mosquitto se procede a instalar librerías adicionales para la comunicación MQTT en Python.

3.2 Comunicación MQTT en Python

Para poder usar el protocolo MQTT en programas de Python se requiere de una librería adicional, a la cual es conocida como Paho Python. Las características con las que cuenta el cliente MQTT que proporciona Paho para Pyhton son las que se observan en la *Figura 3-1*.

MQTT 3.1	✔	Offline Buffering	✔
MQTT 3.1.1	✔	WebSocket Support	✔
LWT	✔	Standard TCP Support	✔
SSL / TLS	✔	Non-Blocking API	✔
Message Persistence	✖	Blocking API	✔
Automatic Reconnect	✔	High Availability	✖

Figura 3-1 Características del cliente Paho Python [21].

Instalar este cliente en la Raspberry Pi se requiere de la ejecución de los siguientes comandos en un terminal.

Listado 3-7 Comandos para la instalación del cliente Paho Python.

```
sudo pip install paho-mqtt
git clone https://github.com/eclipse/paho.mqtt.python.git
cd paho.mqtt.python
python setup.py install
```

Cualquier programa en Python desarrollado debe contar con las siguientes líneas en su código, y así podrá establecer la conexión del cliente Paho Python con el protocolo MQTT.

Listado 3-8 Líneas de código para establecer la conexión a un bróker MQTT en Python.

```
#Importar Libreria Paho
import paho.mqtt.client as mqttClient

#Conexion MQTT
def on_connect(client, userdata, flags, rc):
     if rc == 0:
        print("Conexion Establecida")
        global Connected
        Connected = True
     else:
         print("Conexion Fallida")
         Connected = False

broker_address= "localhost"
port = 1883
user = "domqttica"
password = "eie655"

client = mqttClient.Client("Nombre_Cliente")
client.username_pw_set(user, password=password)
client.on_connect= on_connect
client.connect(broker_address, port=port)
```

La línea de código que se debe agregar para la publicación de un mensaje a un tópico en específico es la siguiente.

Listado 3-9: Línea de código para publicar un mensaje en un tópico usando Python.

```
client.publish("Topico","Mensaje")
```

Y para la subscripción de algún tópico en específico se debe agregar las siguientes líneas de código.

Listado 3-10 Líneas de código para subscribirse a un tópico y recibir mensajes usando Python.

```
client.subscribe("Topico")
def on_message(client, userdata, msg):
 print("Has recibido un Mensaje!")
 print('%s %s' % (msg.topic, msg.payload))
```

Con estos tres aspectos básicos para la conexión, publicación y subscripción en el lenguaje de programación Python.

3.3 Creación del entorno de trabajo

Para tener un entorno adecuado de trabajo tanto virtual como físico se deben realizar unas tareas previas. Primero se procede a instalar git-core para poder cargar librerías de la comunidad de GitHub. También se creara una carpeta para almacenar los distintos programas en Python a desarrollar. Para ello se usaran las siguientes sentencias en un terminal:

Listado 3-11: Comandos para instalar git-core y crear directorio para programas Python.

```
sudo apt-get install git-core
sudo mkdir /home/pi/archivos/python
```

En el directorio se crearan todos los archivos con la extensión ".py" necesarios para programar, cada uno de estos contara con un nombre descriptivo para su fácil identificación.

También se requiere de una protoboard, cables jumper M-M, cables jumper H-H, un pulsador y la Rapberry Pi alimentada como mínimo con 1500 [mA] para su correcto funcionamiento. Con todo esto se procede a programar, conectar y probar cada uno de los sensores.

3.4 Sensor DHT11

En la utilización de este sensor de temperatura y humedad, se comenzó creando un programa en Python, en base a proyectos de la comunidad y principalmente en lo estudiado de este lenguaje de programación. Para trabajar con este sensor se requiere de una librería adicional de DHT Adafruit y un software que permita adicionar librerías a Python. Las líneas para instalarla se presentan en el siguiente listado.

Listado 3-12 Líneas de código para poder utilizar el sensor DTH.

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git
```

```
2 sudo apt-get install build-essential python-dev
3 cd Adafruit_Python_DHT
4 sudo python setup.py install
```

3.4.1 Programa Python sensor temperatura y humedad

El programa diseñado en el archivo “dht11.py” se encuentra con más detalle en el *Apéndice A.1* del informe. Lo programado para la lectura y publicación de la temperatura y humedad sigue el diagrama de flujos que se presenta en la siguiente figura.

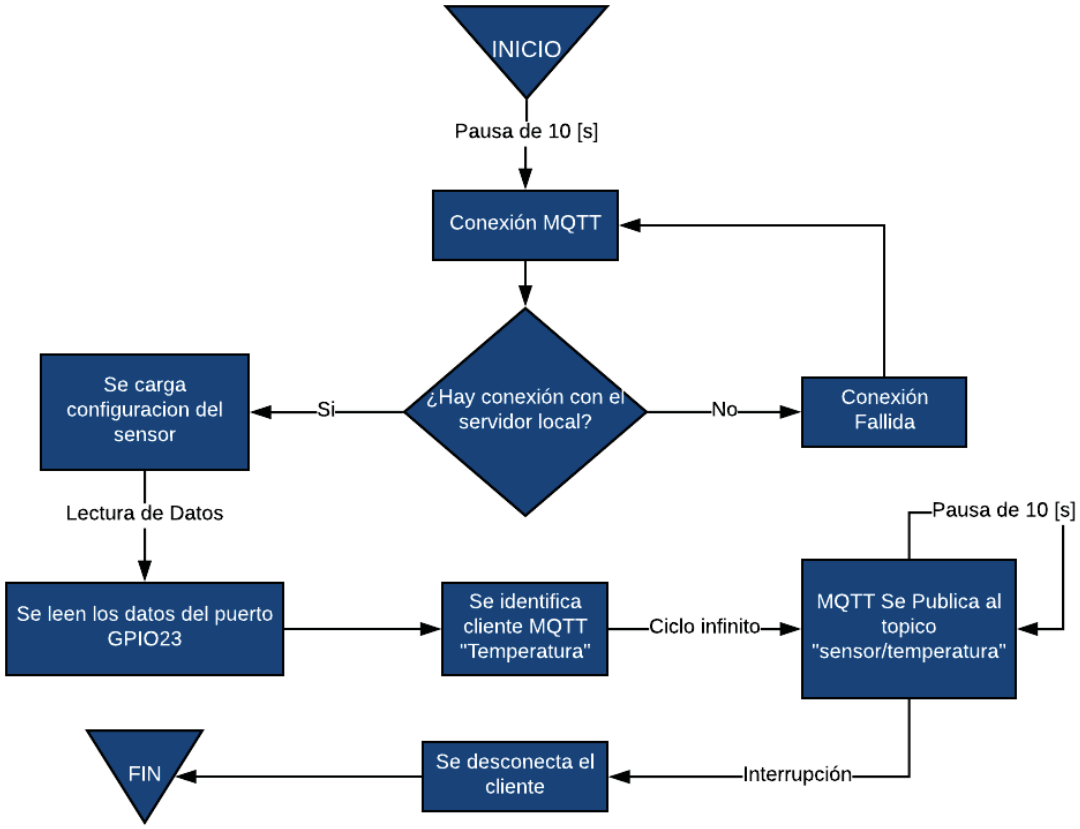

Figura 3-2 Diagrama de flujo del programa desarrollado en Python para el sensor DHT11.

3.4.2 Conexión del sensor DHT11 con la Raspberry Pi

En base al programa desarrollado y una tabla GPIO de la Raspberry Pi se construye el circuito funcional. Dado que el sensor ya se encuentra en una placa con todos los elementos necesarios, es posible realizar una conexión directa con la plataforma. El sensor se alimentara con 3.3 [V], el pin DATA ira al GPIO23 (Pin 16) y la GND a una tierra de la RPi. La conexión realizada se puede observar de manera más intuitiva en el siguiente esquema realizado con el software fritzing [22].

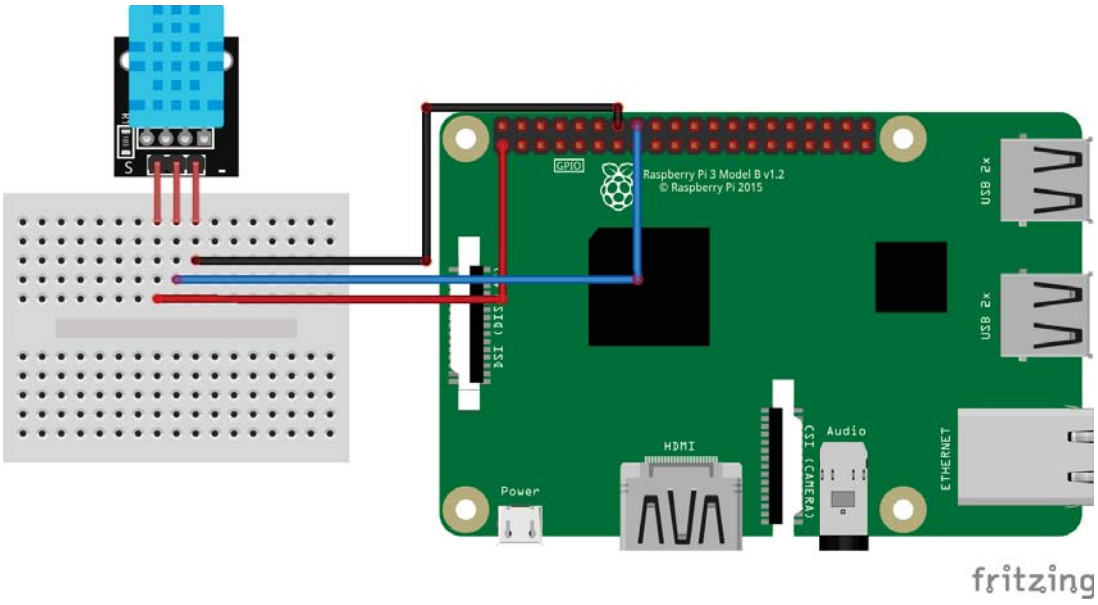

Figura 3-3 Esquema de conexión de un módulo DHT11 con la Raspberry Pi.

3.4.3 Pruebas con el sensor DHT11

Con el programa y la conexión con la Raspberry Pi realizado. Se procede a ejecutar el programa en un terminal, mediante el comando:

Listado 3-13: Línea de código para ejecutar el programa del sensor DHT11.

```
1 sudo python dht11.py
```

Con esto en el terminar empieza a mostrar cada diez segundos la temperatura y humedad del ambiente cercano en el que se encuentra.

```
Temperatura=24.0*  Humedad=61.0%
Temperatura=21.0*  Humedad=63.0%
Temperatura=21.0*  Humedad=62.0%
Temperatura=21.0*  Humedad=62.0%
Temperatura=21.0*  Humedad=62.0%
Temperatura=22.0*  Humedad=63.0%
Temperatura=22.0*  Humedad=63.0%
```

Figura 3-4 Captura de los datos obtenidos por el sensor DTH11 en un terminal.

El sensor fue probado a distintas horas del día, la humedad se mantuvo sobre el 60% lo que es normal para viviendas que están en un clima cálido cerca del mar. La temperatura se fue comparando con la medición de un termómetro de uso doméstico, encontrado diferencias de solo uno a dos grados Celsius. Lo mencionado indica el buen funcionamiento de este sensor a temperaturas normales.

3.5 Sensor LDR

El LDR requiere de un capacitor electrolítico de 1 [uF], en un principio se desarrolló un programa en Python sencillo que solo mostrara los datos obtenidos por el LDR y así poder generar un rango de distintas intensidades. A luz ambiente los valores oscilaban entre los 300 [Ω] a 600 [Ω], a alta intensidad luminosa como un rayo de luz directo o luz de una linterna el valor baja de 100 [Ω] y en un ambiente oscuro el valor se dispara sobre los 1000 [Ω]. Con estos datos se creó un programa con una estructura de flujo de control de condicionales, indicado la intensidad detectada y si esta es alta, normal u oscura.

3.5.1 Programa Python sensor de luminosidad

El programa final escrito en el archivo "ldr.py" esta de forma detallada en el *Apéndice A.2*. El programa desarrollado funciona siguiendo el diagrama de flujo que se presenta a continuación.

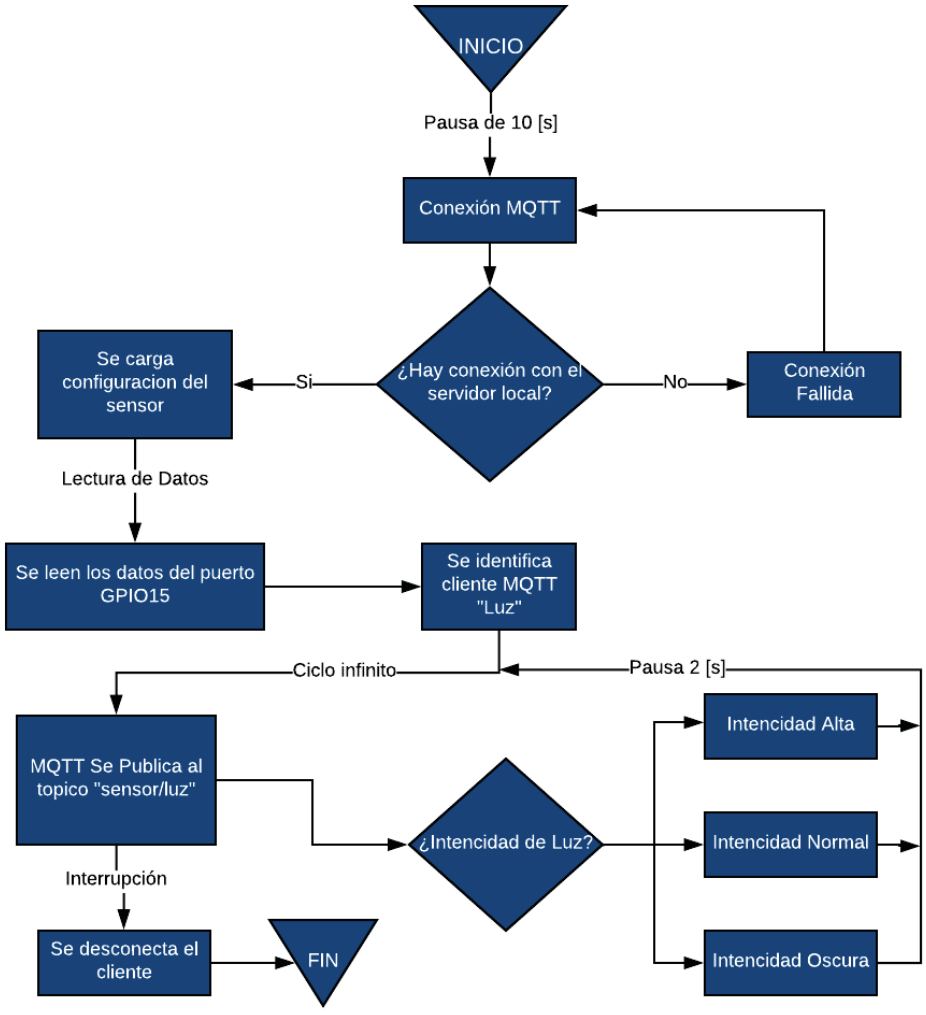

Figura 3-5 Diagrama de flujo del programa desarrollado en Python para el LDR.

3.5.2 Conexión del LDR con la Raspberry Pi

La conexión del circuito realizado para conectar el LDR a la Raspberry Pi se hace en base al programa en Python desarrollado. Alimentado el LDR con 3.3 [V], el valor de la intensidad se obtiene mediante un cable entre el LDR y el condensador electrolítico hacia el Pin 10 de la Raspberry Pi (GPIO15), y el pin negativo del condensador hacia una tierra de la placa.

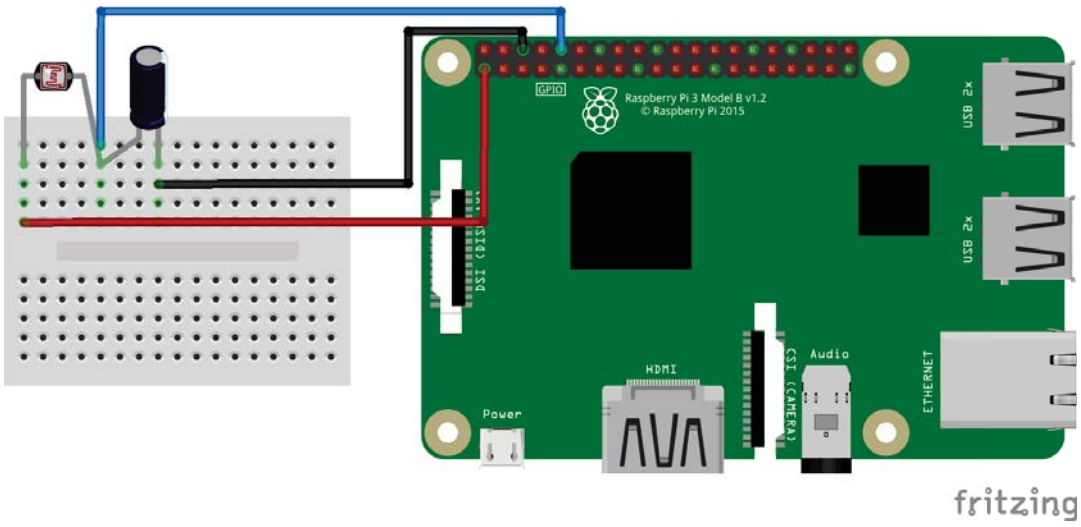

Figura 3-6 Esquema de conexión de un LDR con la Raspberry Pi.

3.5.3 Pruebas con el sensor LDR

Con el programa y la conexión con la Raspberry Pi realizado. Se procede a ejecutar el programa en un terminal, mediante el comando:

Listado 3-14: Línea de código para ejecutar el programa del sensor LDR.

```
1 sudo python ldr.py
```

Y en la pantalla del terminal se ven los datos de intensidad lumínica cada dos segundos.

```
('Intencidad Normal:', 399)
('Intencidad Normal:', 403)
('Intencidad Oscura:', 47266)
('Intencidad Oscura:', 29430)
('Intencidad Alta:', 28)
('Intencidad Alta:', 27)
```

Figura 3-7 Captura de los datos obtenidos por el LDR en un terminal

El funcionamiento de este sensor fue el esperado, detectando bastante rápido el cambio brusco de intensidad luminosa, no se encontraron mayores errores en su utilización. Al tener ya un rango conocido de intensidades se puede utilizar este sensor para diversas aplicaciones hacia la domótica, como puede ser la automatización con ahorro energético o de seguridad.

3.6 Sensor HC-SR501

Este sensor se activa al detectar el movimiento de cuerpos que emiten en el espectro infrarrojo, el programa a desarrollar publicara en un tópico específico a qué hora, minuto y segundo se detectó un movimiento dentro de su rango visible.

3.6.1 Programa Python del sensor de movimiento

El programa final escrito en el archivo "hcsr501.py" esta de forma detallada en el *Apéndice A.3*. El programa funciona siguiendo el diagrama de flujo que se presenta en la siguiente figura.

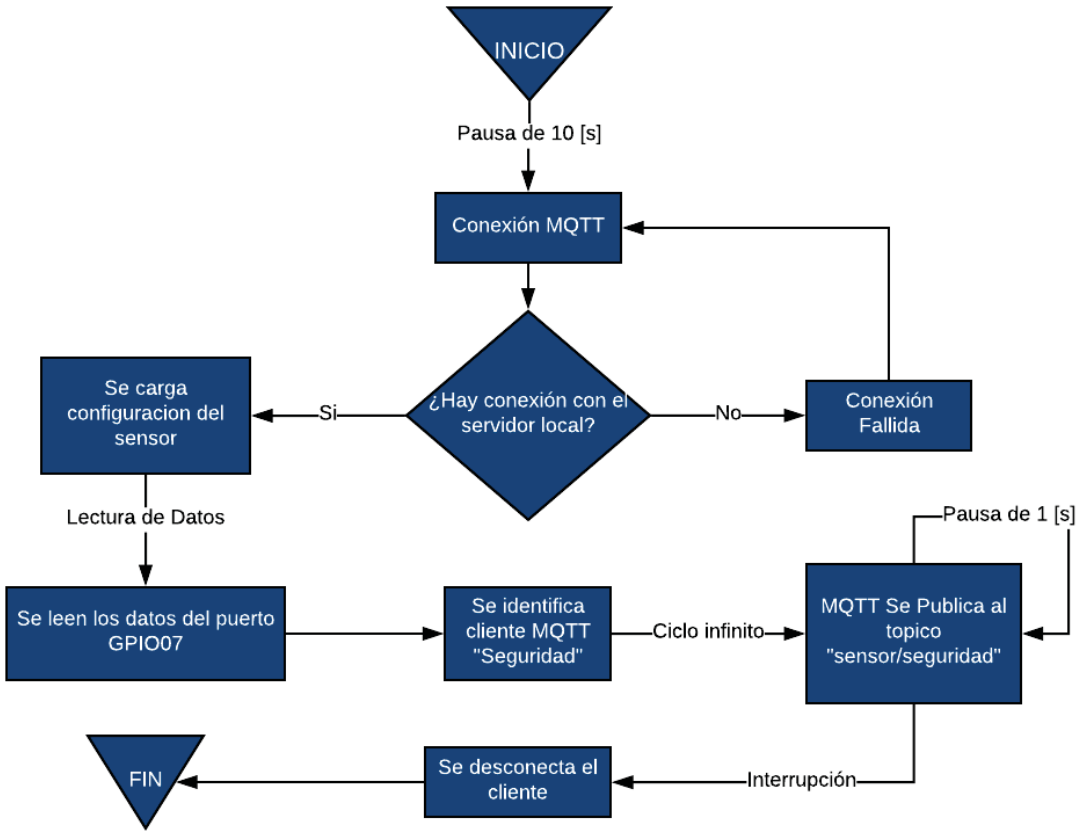

Figura 3-8 Diagrama de flujo del programa desarrollado en Python para el sensor HC-SR501.

Para el programa se importó la librería "time" para poder indicar el tiempo exacto en el cual se detectó el movimiento.

3.6.2 Conexión HC-SR501 con la Raspberry Pi

La conexión del sensor con la Raspberry Pi se hace de forma directa, ya que se encuentra en una placa para poder realizar esto. Solo de debe tener precaución en el momento de conectar a los pines correspondientes según el programa escrito y que su fuente de alimentación sea de 5 [V].

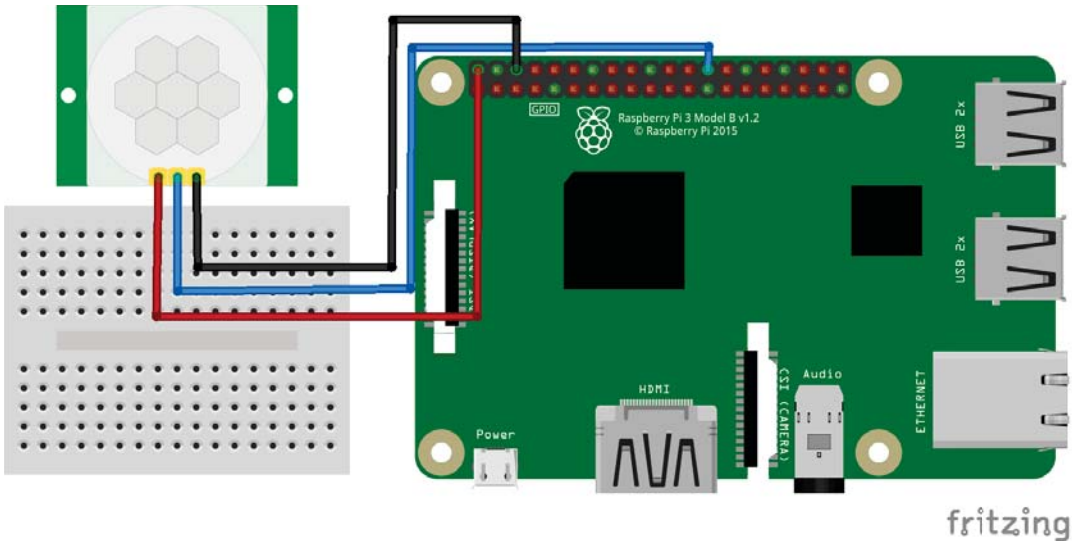

Figura 3-9 Esquema del conexionado del sensor HC-SR501 con la Raspberry Pi.

3.6.3 Pruebas con el sensor HC-SR501

Al ejecutar el archivo “hcsr501.py” en un terminal se obtuvo el siguiente resultado:

```
Movimiento Detectado a las 03:54:59
Movimiento Detectado a las 03:55:01
Movimiento Detectado a las 03:55:10
Movimiento Detectado a las 03:55:12
```

Figura 3-10 Captura de los datos obtenidos por el sensor HC-SR501 en un terminal.

El sensor detecto el movimiento de emisores de infrarrojo con mucha precisión y un buen alcance. Una vez que detecta el movimiento de un cuerpo emisor y lo registra. Este no vuelve a registrar el movimiento si el cuerpo no vuelve a moverse. Su buen funcionamiento lo hace ideal para aplicaciones con alarmas en domótica.

3.7 Sensor MQ-135

La creación del programa para la lectura de datos de este sensor que detecta la calidad del aire es más desarrollada que los anteriores. Ya que en este se busca además que se pueda notificar mediante un email la presencia de gases peligrosos.

3.7.1 Programa Python sensor calidad del aire

Para este programa fue necesario importar la librería “smtplib”, con este es posible enviar correos electrónicos junto a un mensaje predefinido. El programa final escrito en el archivo “mq135.py” esta de forma detallada en el *Apéndice A.4*. El programa funciona en base al diagrama de flujo que se presenta en la siguiente figura.

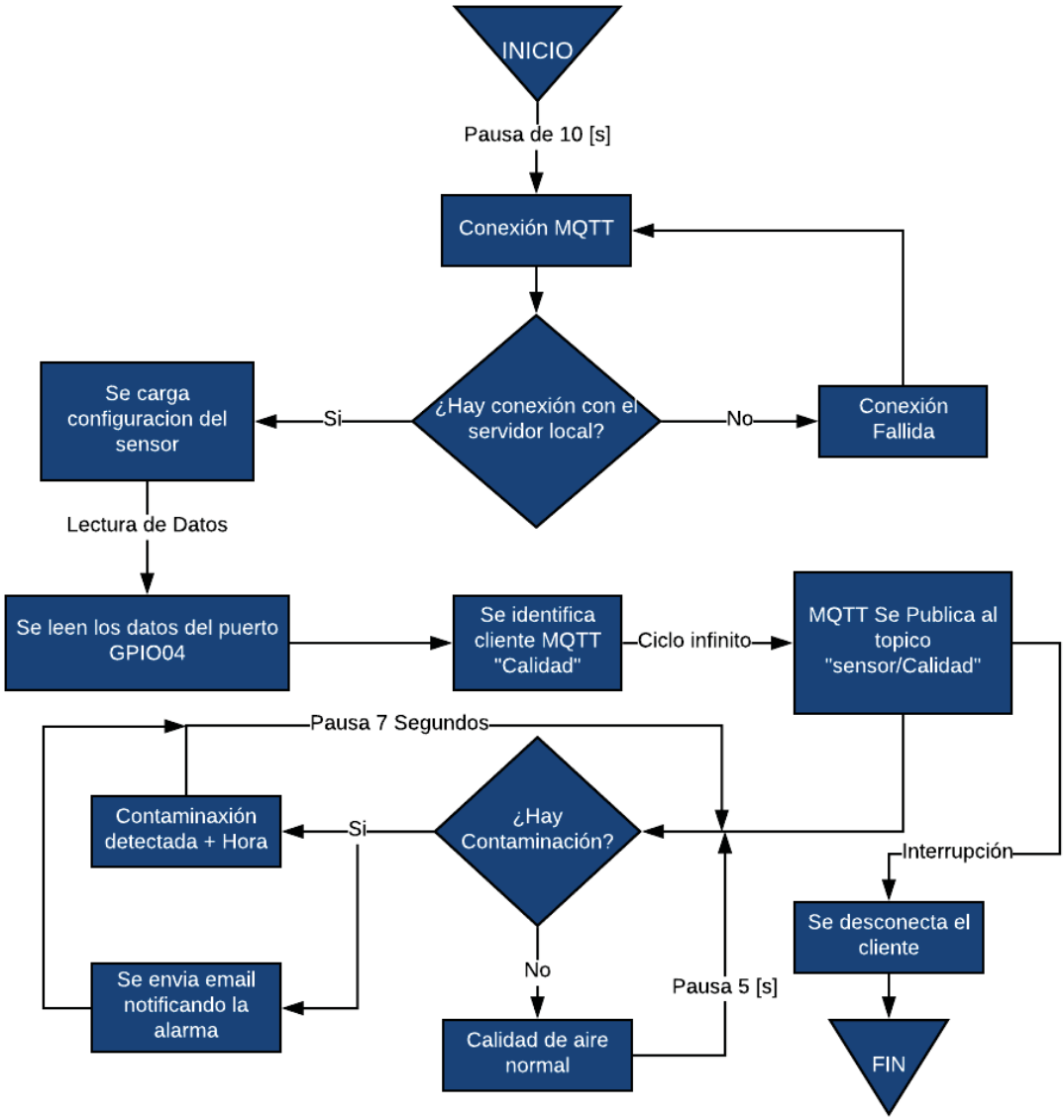

Figura 3-11 Diagrama de flujo del programa desarrollado en Python para el sensor MQ-135.

3.7.2 Conexión MQ-135 con la Raspberry Pi

El conexionado del sensor con la Raspberry Pi es de manera directa, ya que al igual que algunos de los sensores previamente trabajados este se encuentra inserto en una placa. Se deben tener las mismas precauciones que todos los sensores, verificar que se conectan los pines correctos entre el sensor y los de la Raspberry Pi. También se comprueba que el pin conectado corresponda a la GPIO especificada en el programa escrito y que la alimentación sea de 5 [V], la conexión correcta se presenta en la siguiente figura.

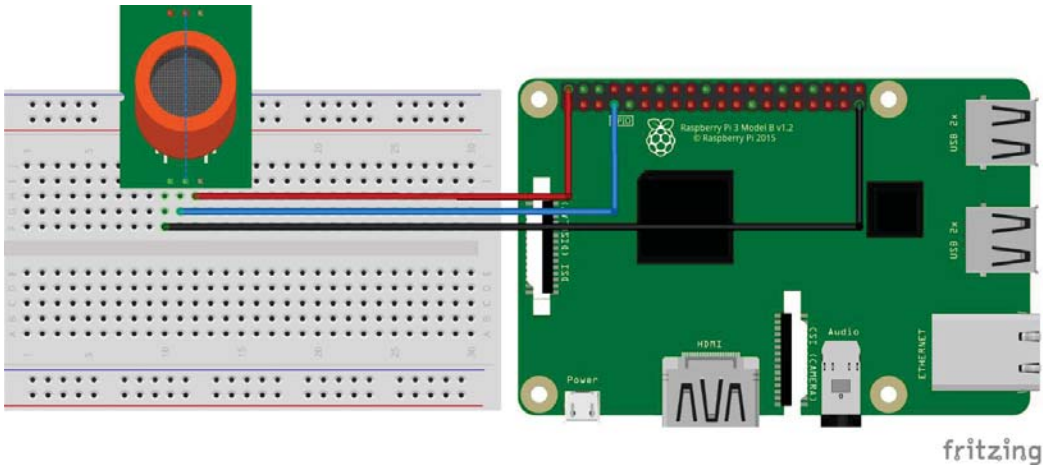

Figura 3-12 Esquema del conexionado del sensor MQ-135 con la Raspberry Pi.

3.7.3 Pruebas con el sensor MQ-135

Al ejecutar el archivo “mq135.py” en un terminal se obtuvo el siguiente resultado:

```
Calidad del Aire normal
Calidad del Aire normal
Contaminacion detectada a las 23:07:54
Calidad del Aire normal
```

Figura 3-13 Captura de los datos obtenidos por el sensor MQ-135 en un terminal.

En los mensajes se observa que se detectó contaminación en el aire, esto fue realizado intencionalmente para probar el sensor. El sensor MQ-135, es bastante versátil ya que puede detectar sulfuros, benceno, amoniaco, alcohol y gases volátiles como el butano y metano según lo investigado. Para emitir el contaminante de aire se usó un encendedor el cual al presionarlo sin activar la mecha, libera ligeramente gas butano. Cuando detecta algún contaminante este sensor enciende un LED verde y si todo está normal se mantiene apagado. También se recibe un correo electrónico cuando el sensor detecta un gas toxico.

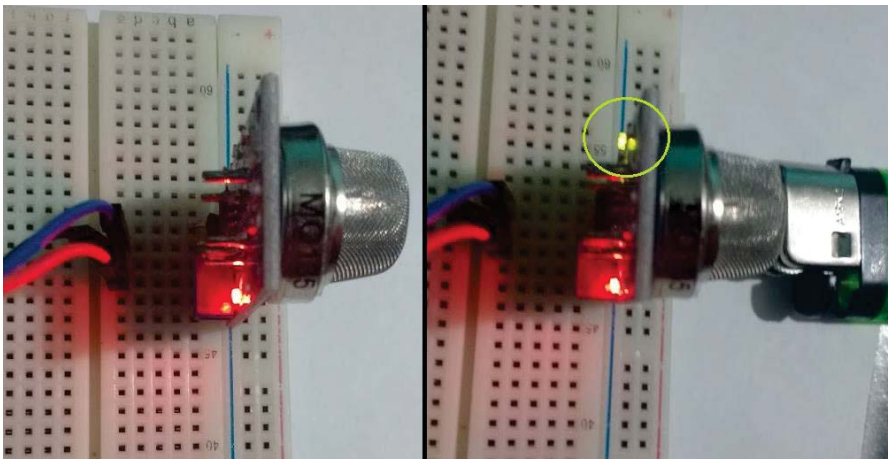

Figura 3-14 Funcionamiento del sensor MQ-135 frente a calidad de aire normal y contaminado.

3.8 Apagado físico y remoto de la Raspberry Pi

Para poder apagar la Raspberry Pi, se realizaron dos programas en el lenguaje Python, uno que permite el apagado físico mediante un pulsador y otro que permite un apagado remoto desde la App.

3.8.1 Programa para el apagado/encendido físico

Se desarrolla un programa que permita apagar/encender el dispositivo de forma directa y física por medio de un pulsador que estará conectado mediante cableado a la Raspberry Pi. Se utiliza un pulsador de dos pines, uno se conecta a la GPIO03 y otro a tierra. El programa final escrito en el archivo "power.py" esta de forma detallada en el *Apéndice A.5*. El funcionamiento del código sigue el diagrama de flujos que se presenta.

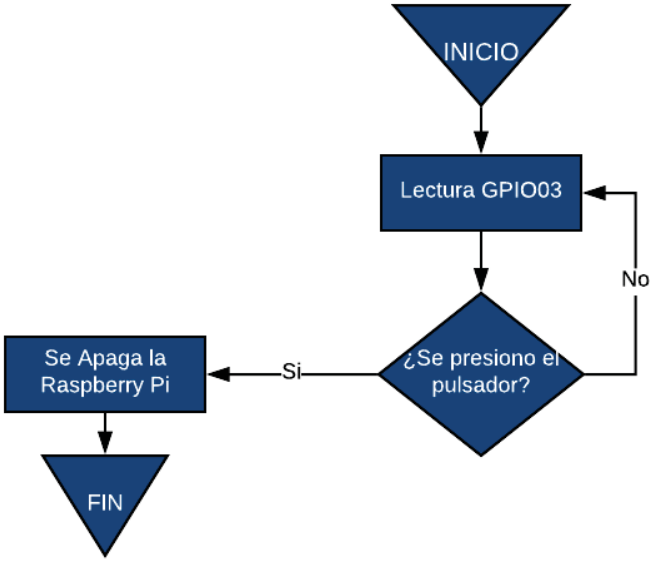

Figura 3-15 Diagrama de flujo del programa desarrollado en Python para el apagado físico.

3.8.2 Programa para el apagado remoto

Se realizó el programa que permite apagar la Raspberry Pi de forma remota desde la App que se desarrollara. Este programa se conectara al servidor local Mosquitto y se subscribirá al tópico "domqttica/power", donde el publicador será un botón en la aplicación móvil. Cuando se envié la publicación desde la App esta será recibida por el servidor Mosquitto. En el código se define un parámetro como "apagar" y un variable vacía, la cual tendrá el valor del mensaje recibido en el tópico, dado que el mensaje que se envía desde la aplicación móvil será "apagar", la variable tendrá ese valor cuando lo reciba y cuando el parámetro definido sea igual a la variable recibida se apagara la Raspberry Pi. El programa escrito en el archivo "powermqtt.py" esta de forma detallada en el *Apéndice A.6*. El funcionamiento del código sigue el diagrama de flujos que se presenta.

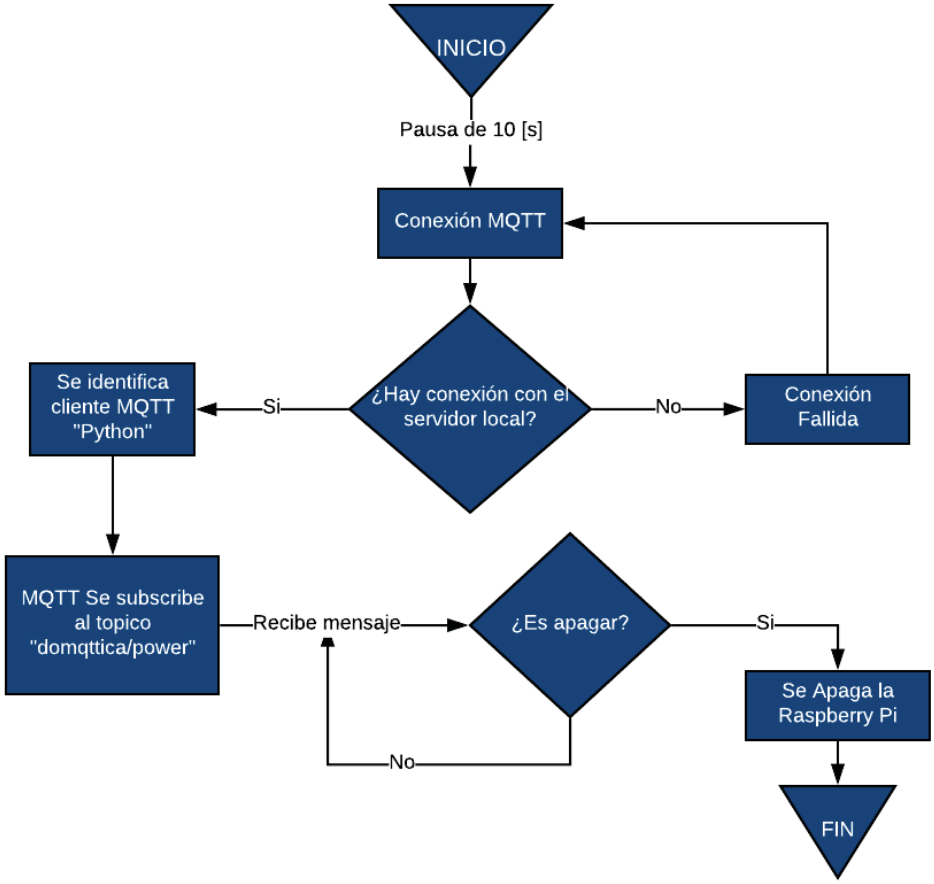

Figura 3-16 Diagrama de flujo del programa desarrollado en Python para el apagado remoto.

3.8.3 Led indicador del estado de la Raspberry Pi

Se le agregara un LED de color azul, el cual indicara el estado de encendido de la Raspberry Pi 3. Para ello se debe habilitar el puerto serial GPIO14, modificando las configuraciones de boot del dispositivo. Esto se realiza con el siguiente comando en un terminal.

Listado 3-15: Comando para modificar las configuraciones de boot de la Raspberry Pi.

```
1 sudo nano /boot/config.txt
```

Al final del archivo se le debe agregar "*enable_uart=1*" y luego guardar este mismo. El ánodo del LED azul se conectara a la GPIO14, el cátodo a una resistencia de 330 [Ω] y de este a un Pin a tierra, en este caso el PIN34. Esta conexión se muestra en la siguiente figura.

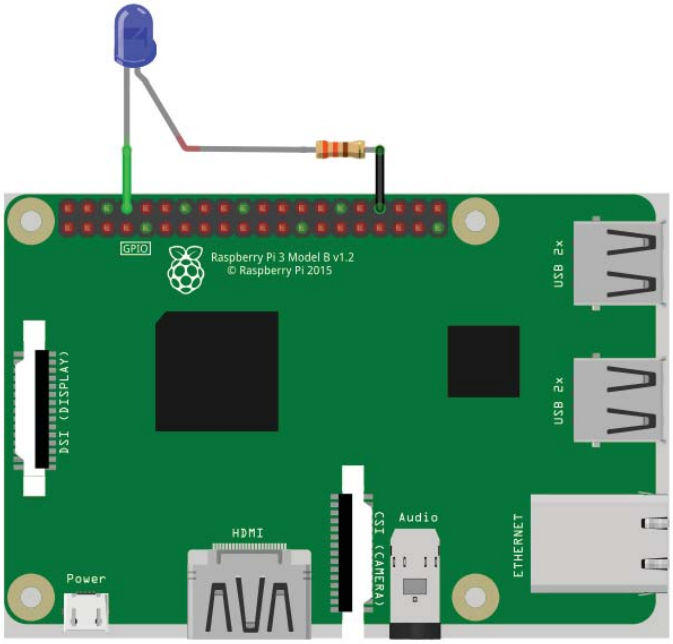

Figura 3-17 Esquema del conexionado del LED indicador de estado con la Raspberry Pi.

3.9 Instalación y configuración de Node-RED en la Raspberry Pi

Instalar Node-RED en una Raspberry Pi es extremadamente sencillo. Esto es debido a que por defecto, ya viene preinstalado en el sistema operativo Raspbian. Solo se debe abrir un terminal e introducir el siguiente comando:

Listado 3-16: Comando para actualizar Node-RED desde un terminal.

```
1 bash <(curl -sL https://raw.githubusercontent.com/node-red/raspbian-
  deb-package/master/resources/update-nodejs-and-nodered)
```

Básicamente lo que realiza este comando es actualizar NodeJS y Node-RED. Es importante siempre trabajar con las últimas versiones. Para iniciar el servicio que proporciona esta herramienta, en un terminal se debe introducir el siguiente comando:

Listado 3-17: Comando para iniciar Node-Red desde un terminal.

```
1 node-red-start
```

Iniciado Node-RED se puede acceder a trabajar con él a través de una página web. La dirección de la página es indicada una vez que se inicia Node-RED, está por defecto es la IP local de la Raspberry Pi o se puede acceder al entorno visual introduciendo la dirección “http://127.0.0.1:1880/#”.

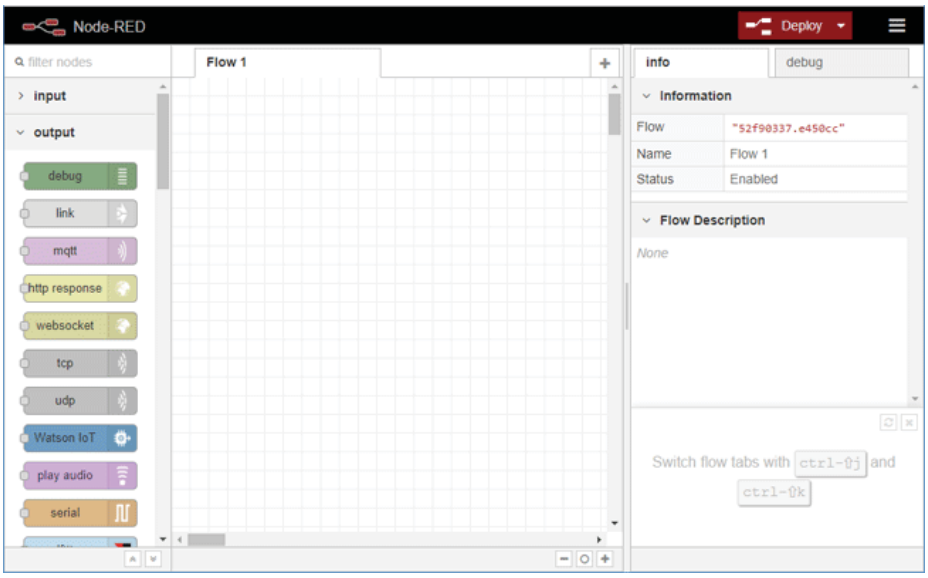

Figura 3-18 Entorno visual de trabajo de la herramienta Node-RED.

Lo que se busca realizar con esta herramienta es un puente entre el bróker local Mosquitto con el bróker en la nube CloudMQTT. Para esto se crearon dos enlaces entre cuatro nodos, dos nodos MQTT de entrada y dos nodos MQTT de salida.

El primero de estos enlaces enviara todos los mensajes que reciba del tópico “sensor/#” desde el broker local Mosquitto hasta el servidor en la nube CloudMQTT. Cada uno de estos nodos se configura introduciendo la dirección del host, el puerto utilizado, el tópico, la calidad de servicio y dar un nombre si se requiere.

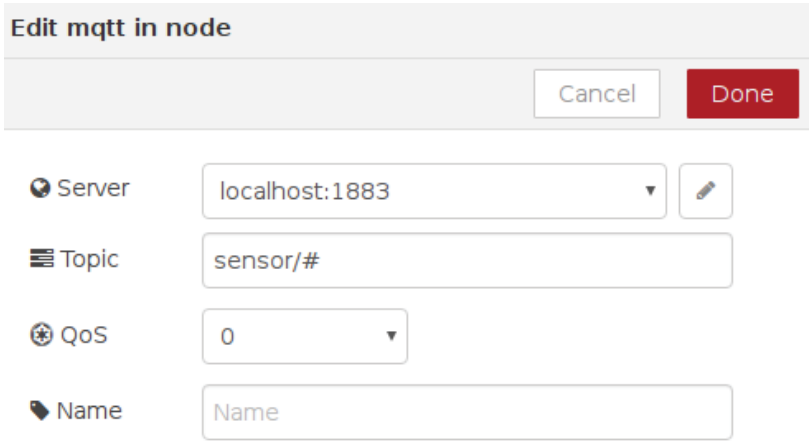

Figura 3-19 Configuración de conexión en un nodo MQTT.

A estos también se debe indicar los parámetros de seguridad como usuario y contraseña para establecer una conexión de forma segura y exitosa.

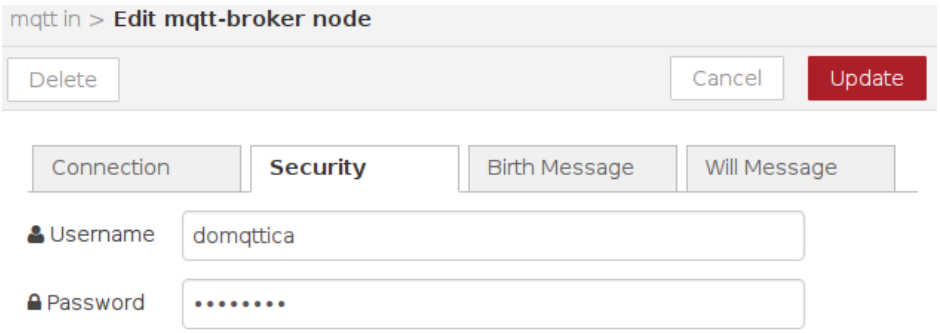

Figura 2-8 Configuración de seguridad un nodo MQTT.

Se debe realizar la misma configuración para el nodo MQTT de salida, colocando los parámetros correspondientes para conectarse al bróker CloudMQTT. El segundo enlace se encargara de enviar el mensaje "apagar" al tópico "domqttica/power" desde el servidor CloudMQTT al servidor local Mosquitto. Esto para poder enviar la instrucción a un programa en Python y apagar de forma remota la Raspberry Pi desde el móvil.

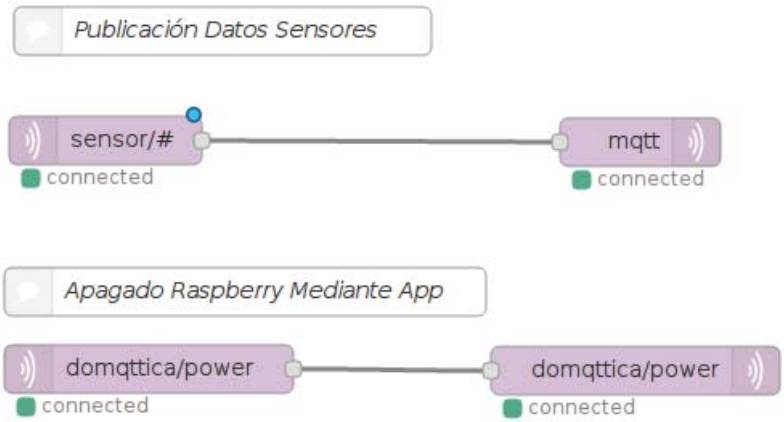

Figura 3-20 Estructura de nodos realizada en Node-RED.

Como se puede apreciar en la estructura final desarrollada de la *Figura 3-20*, las conexiones entre el servidor local y el servidor en la nube son correctas. Además Node-RED permite exportar y guardar el flujo de nodos desarrollado en líneas de código, estas se encuentran a detalle en el *Apéndice A.7*. Con esto ya se puede realizar una comunicación por medio del protocolo MQTT de manera global.

Ya instalado y configurado Node-RED, se introduce el siguiente comando en el terminal para que este inicie de forma automática al encender la Raspberry Pi.

Listado 3-18: Comando para iniciar Node-Red de forma automática desde un terminal.

```
1 sudo systemctl enable nodered.service
```

3.10 Automatización de la Raspberry Pi

Dado que se cuenta con los programas de Python necesarios desarrollados y se tiene configurado los servicios de Mosquitto y Node-RED. Se hace necesario el inicio y ejecución de cada uno de estos programas y servicios de forma automática. Esto ya se realizó con el servicio de Node-RED, para el servicio del bróker local se debe hacer lo mismo mediante el siguiente comando en un terminal.

Listado 3-19: Comando para iniciar Mosquitto de forma automática desde un terminal.

```
1 sudo systemctl enable mosquitto.service
```

Con esto inicia el servicio de forma automática pero no carga las configuraciones personalizadas. Para poder cagar las configuraciones establecidas una vez iniciado el servicio se crea un launcher script, el cual se ejecutara cada vez que inicie el sistema operativo. Lo primero a realizar es crear el script.

Listado 3-20: Comando crear un script desde un terminal.

```
1 sudo nano launcher.sh
```

Luego a este archivo se le escribe el comando "*sudo mosquitto -d*" y se guarda el archivo. Después en el mismo terminal se debe convertir el script en un ejecutable con el siguiente comando.

Listado 3-21: Comando convertir un script en un ejecutable desde el terminal.

```
1 sudo chmod 755 launcher.sh
```

Ahora se debe crear un crontab, que es un proceso maestro que ejecuta otros en segundo plano.

Listado 3-22: Comando para crear un crontab desde un terminal.

```
1 sudo crontab -e
```

Se escogerá la segunda opción que da como respuesta al escribir el comando previo, esta es la de crear un nuevo archivo crontab y editarlo con mediante "*sudo nano*". En la parte final del archivo se debe colocar la siguiente línea para que se inicie el script de forma automática cuando se encienda la Raspberry Pi.

Listado 3-23: Línea que se debe agregar al archivo crontab desde un terminal.

```
1 @reboot sh /home/pi/launcher.sh
```

Se debe realizar lo mismo para cada uno de los programas en Python. Con esto se logra que se ejecuten y funcionen de manera automática la publicación continua de datos cada uno de los sensores conectados, una vez encendida la Raspberry Pi. Solo que en cada script de los programas se les debe agregar el directorio donde están alojados, como por ejemplo el del sensor de temperatura y humedad.

Listado 3-24: Ejemplo de líneas que se debe agregar al ejecutable de cada programa desarrollado.

```
cd /home/pi/archivos/python
sudo python dht11.py
```

Los seis programas desarrollados con el lenguaje de programación Python se encuentran alojados en el directorio "/home/pi/archivos/python".

Figura 3-21 Programas desarrollados en Python.

Una vez creado todos los ejecutables para cada programa, el archivo crontab queda con las siguientes líneas de código, ejecutándose siete scripts de forma automática.

Listado 3-25 Líneas de código para establecer la conexión a un bróker MQTT en Python.

```
@reboot sh /home/pi/launcher.sh
@reboot sh /home/pi/power.sh
@reboot sh /home/pi/powermqtt.sh
@reboot sh /home/pi/temperatura.sh
@reboot sh /home/pi/seguridad.sh
@reboot sh /home/pi/luminosidad.sh
@reboot sh /home/pi/calidad.sh
```

Los siete ejecutables creados para la automatización de los programas Python, se encuentran alojados en el directorio "/home/pi/".

Figura 3-22 Scripts desarrollados para la ejecución automática de programas en Python.

4 Desarrollo de la aplicación móvil

En el presente capitulo se expone todo el desarrollo que involucra la aplicación móvil para Android. Se indicaran las características que este debe tener así como los elementos más importantes en la App.

4.1 Requisitos y diseño de la App

En este punto como parte inicial para el desarrollo de la App, se detallaran los requisitos funcionales, no funcionales y diseño que deberá cumplir la aplicación móvil.

4.1.1 Requisitos funcionales

Los requisitos funcionales describen todas las interacciones que tendrán los usuarios con la aplicación. Estos requisitos son:

- *Pantalla principal*: Esta debe constar con diversos botones para el control y gestión de la aplicación.
- *Iconos intuitivos*: El usuario podrá identificar a simple vista las distintas funciones de la aplicación por medio de iconos.
- *Información estática*: La aplicación debe proporcionar información estática de diversos elementos conociendo su estado actual.
- *Información dinámica*: La aplicación debe proporcionar en tiempo real información de los distintos sensores sobre su estado real.
- *Control*: Mediante la aplicación se podrá apagar el dispositivo con los sensores por medio de un botón.
- *Notificaciones*: Notificar al usuario distintos eventos de interés, como activación de alarmar o posibles errores.

4.1.2 Requisitos no funcionales

Los requisitos no funcionales para la aplicación, es decir los que no especifican el comportamiento del sistema son:

- *Rendimiento*: La aplicación debe desempeñar su función de una manera fluida. Se debe buscar la experiencia de uso más agradable para el usuario.

- *Interfaz y usabilidad*: La aplicación debe constar con una interfaz sencilla, atractiva e intuitiva. De tal forma que su uso no suponga un impedimento o esfuerzo al usuario a la hora de hacer uso de la aplicación.
- *Autenticación*: Para utilizar la aplicación se debe identificar el usuario para darle autorización de uso.
- *Seguridad*: La aplicación debe proporcionar seguridad al usuario, es decir no debe permitir que se instale ningún elemento dañino o malicioso. Los datos del usuario deben ser cifrados.
- *Documentación*: La aplicación debe contar con un manual de usuario, la documentación debe ser clara y concisa.
- *Almacenamiento*: La aplicación requiere un determinado espacio disponible para poder ser instalado y poder funcionar.
- *Disponibilidad*: La aplicación debe estar disponible para poder ser instalada.

4.1.3 Diseño

Toda aplicación o producto a desarrollar requiere de un nombre, icono y slogan. Para la App a desarrollar en este proyecto se le dio el nombre de: "DoMQTTica", destacando que su función es para la domótica y que utiliza el protocolo de comunicación MQTT. Se diseñó un icono y slogan, el slogan es: "*Monitoreo desde tu Smartphone*". El icono diseñado junto al slogan se aprecia en la *Figura 4-1*.

Figura 4-1 Diseño del icono de la aplicación móvil DoMQTTica.

Un aspecto fundamenta en el diseño de la interfaz, para ello se creó un mockups a través del servicio online de NinjaMock [23]. El esquema del diseño del menú principal con sus respectivas funciones se presenta a continuación.

Figura 4-2 Esquema del diseño interfaz principal de la App DoMQTTica.

1) *Interfaz Principal*: Pantalla principal de la aplicación, en la que se podrá visualizar las principales funciones que realiza la App.
2) *Temperatura Y Humedad*: Contara con un icono para tener acceso a una nueva pantalla y poder monitorear la temperatura y humedad actual en diversos sitios de la vivienda.
3) *Luz*: Contara con un icono para tener acceso a una nueva pantalla para el control y gestión de las luces de la vivienda.
4) *Seguridad*: Contara con un icono para acceder a una nueva pantalla, en la que se podrá tener conocimiento de los sensores de movimiento y seguridad activos.
5) *Calidad del aire*: Contara con un icono para acceder a una nueva pantalla, conociendo la calidad del aire en la vivienda.
6) *Imagen de la App*: Se tendrá el icono de la aplicación en el centro junto con su nombre.
7) *Mensajes Recibidos*: Es un cuadro de texto, en el cual se podrán visualizar todos mensajes enviados por los sensores.

8) *C. Local*: Contara con un icono para acceder a una pantalla a la conexión local, donde se podrá monitorear de forma local los distintos sensores.
9) *Ayuda*: En ayuda se podrá acceder a comentarios del desarrollador junto la información de contacto de la persona.
10) *Botón de Apagado*: Al presionar este botón rectangular se podrá apagar de forma remota la Raspberry Pi.

Cabe mencionar que la interfaz presentada es la versión esquemática final que paso por múltiples revisiones. También antes de acceder a esta pantalla principal se presentara un splash al iniciar la App.

4.2 Comunicación MQTT en Java

La utilización del protocolo MQTT en el lenguaje Java requiere de una librería adicional, tal y como se menciona en el capítulo previo esta es Paho. En este apartado se indicara las características del cliente que proporciona y como utilizarlo en el software Android Studio.

4.2.1 Cliente Paho Java

Es una biblioteca de cliente MQTT escrita en Java para desarrollar aplicaciones que se ejecuten en plataformas compatibles con Java, como por ejemplo Android.

MQTT 3.1	✔	Offline Buffering	✔
MQTT 3.1.1	✔	WebSocket Support	✔
LWT	✔	Standard TCP Support	✔
SSL / TLS	✔	Non-Blocking API	✔
Message Persistence	✔	Blocking API	✔
Automatic Reconnect	✔	High Availability	✔

Figura 4-3 Características del cliente Paho Java.

Este cliente soporta:

- Las especificaciones de la versión 3.1 y 3.1.1 del protocolo MQTT.
- LWT (Last Will and Testament) o última voluntad y testamento para notificar la desconexión de un cliente de manera abrupta, especificando si se requiere un último mensaje.
- Transporte en SSL/TLS y autenticación mediante certificados SSL/TLS.
- Persistencia de mensajes en caso de fallo de la aplicación y reconexión automática si la conexión con el servidor se pierde.
- Memorización de mensajes cuando esta desconectado y soporte a Websocket.
- Standard TCP, el bloqueo o no de API. Además es de alta disponibilidad, buscando alternativas si no se puede conectar al servidor.

4.2.2 Paho Java en Android Studio

Para utilizar esta librería en el software de programación Android Studio, es necesario agregar unas líneas de código en nuestra aplicación. Estas líneas de código son en base al tutorial de proporcionado por HiveMQ [24]. Primero en el *build.gradle* de la App, se agrega el repositorio y las dependencias de Paho con las siguientes líneas de código:

Listado 4-1 Líneas de código que se agregan al build gradle de la App.

```
repositories {
    maven {url "https://repo.eclipse.org/content/repositories/paho-
releases/"}}

dependencies {
 compile('org.eclipse.paho:org.eclipse.paho.android.service:1.0.2'){
    exclude module: 'support-v4'}}
```

Realizado esto se sincroniza el software, descargando las librerías necesarias para utilizar Paho en Android Studio. Para poder crear un enlace al servicio Android Paho, el servicio debe declararse en archivo "AndroidManifest.xml" con la siguiente línea dentro de la etiqueta <application>:

Listado 4-2 Líneas de código que se agregan al AndroidManifest.xml de la App.

```
<service android:name="org.eclipse.paho.android.service.MqttService" >
</service>
```

Además para el funcionamiento de este servicio, en el mismo archivo "AndroidManifest.xml" se deben agregar los siguientes permisos:

Listado 4-3 Permisos que se agregan al AndroidManifest.xml de la App.

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Para realizar la conexión se declaran cadenas de texto estáticas con la dirección del servidor MQTT, el puerto, usuario y contraseña a utilizar.

Listado 4-4 Líneas de código para declarar los datos del servidor MQTT.

```
static String MQTTHOST = "tcp://SERVIDOR:PUERTO";
static String USERNAME = "USUARIO";
static String PASSWORD = "CONTRASEÑA";
```

Luego se debe crear el cliente MQTT, cargando los datos previamente establecidos.

Listado 4-5 Líneas de código para declarar los datos del servidor MQTT.

```
String clientId = MqttClient.generateClientId();
client = new MqttAndroidClient(this.getApplicationContext(),MQTTHOST, clientId);
MqttConnectOptions options = new MqttConnectOptions();
options.setUserName(USERNAME);
options.setPassword(PASSWORD.toCharArray());
```

La conexión con el servidor MQTT ser realiza por medio de las siguientes líneas de código:

Listado 4-6 Líneas de código para conectarse con el servidor MQTT.

```
try {
    IMqttToken token = client.connect(options);
    token.setActionCallback(new IMqttActionListener() {
    @Override
    public void onSuccess(IMqttToken asyncActionToken) {
    Toast.makeText(menu.this, "Conexión Establecida!", Toast.LENGTH_LONG).show();
    setSubscription();
    }
    @Override
    public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
    Toast.makeText(menu.this, "Conexión Fallida!", Toast.LENGTH_LONG).show();
    }});
    } catch (MqttException e) {e.printStackTrace();}
```

Esto también enviara un mensaje durante unos segundos en la pantalla del Activity en ejecución, indicando si la conexión fue establecida o fallida.

Para realizar una publicación de algún mensaje a un tópico en específico se utiliza las siguientes líneas de código, donde se declara el tópico a publicar, el mensaje y la calidad de servicio entre otras opciones:

Listado 4-7 Líneas de código para publicar un mensaje a un tópico.

```
String topicStrpub = "NOMBRE_DEL_TOPICO";
    public void pub(View v){
    String topic = topicStrpub;
    String message = "MENSAJE";
    try {
        client.publish(topic, message.getBytes(), 0, false);
        } catch (MqttException e) {
            e.printStackTrace();
        }}
```

En tanto para la subscripción, se deben indicar los parámetros como nombre del tópico y calidad del servicio, tal como se indica en el siguiente listado de código:

Listado 4-8 Líneas de código para subscribirse a un tópico.

```
String topicStrsub = "NOMBRE_DEL_TOPICO";
    private void setSubscription(){
        try {
            client.subscribe(topicStrsub, 0);
        }catch (MqttException e){
            e.printStackTrace();
        }}
```

Con las líneas básicas para poder realizar la comunicación con MQTT en Java, se procede a desarrollar la aplicación móvil en el software Android Studio.

4.3 App DoMQTTica

En este punto se presentara de manera descriptiva el trabajo que se realizó para el desarrollo de al App DoMQTTica. Todo el código implicado en el desarrollo de la App se encontrara de forma detallada en el *Apéndice A.8* del presente informe.

4.3.1 Entorno de trabajo

El entorno de trabajo cuenta con ocho archivos java y ocho layout en .xml, de los cuales siete son ventanas interactivas y una es el splash de carga al iniciar la aplicación móvil.

Figura 4-4 Entorno de trabajo para el desarrollo de la App.

4.3.2 Splash de la App

Se programó un splash para cuando se ejecute la aplicación muestre el logo, nombre y eslogan de la app durante 1.5 [s]. Este será el Activity por defecto de la App, ya que será la primera en ejecutarse de toda la aplicación.

4.3.3 Menú y submenús de la App

Para el menú de la App se utilizaron iconos intuitivos de cada función que realizan. Desde este menú se puede visualizar todos los submenús y los mensajes entrantes, se podrá acceder a los submenús y monitorear a los datos de temperatura y humedad, iluminación, seguridad, calidad de aire, conexión local y ayuda para usar la aplicación.

Cada uno de estos elementos está posicionado según márgenes y alineaciones definidas en su código. Además cada uno de ellos estará identificado, con lo cual conocerá su función y que debe estar a la derecha, izquierda, abajo o arriba de otro elemento identificado.

También en cada submenú se encontraran dos botones, uno para regresar al menú principal y otro cerrando la aplicación. El menú que se carga una vez finalizado el splash es que se ve en la App es el de la siguiente figura.

Figura 4-5 Menú principal de la App DoMQTTica.

Tanto en el menú principal como los submenús se establece la conexión con el servidor en la nube CloudMQTT, a excepción de submenú de la conexión local y el de ayuda. Para esto se utiliza la dirección del host, puerto y usuario que proporciona CloudMQTT, También se establecen los nombres de cada tópico que tiene la aplicación, los cuales son:

- Menú Principal: "sensor/#" y "domqttica/power"
- Sub-Menú Temperatura y Humedad: "sensor/temperatura"
- Sub-Menú Luminosidad: "sensor/luz"
- Sub-Menú Seguridad: "sensor/seguridad"
- Sub-Menú Calidad del Aire: "sensor/calidad"

En la aplicación solo se dejaron subscripciones a tópicos donde ser reciben datos para su monitoreo y un solo publicador para el apagado remoto el cual tiene el mensaje "apagar". Además en cada uno de esos se agregó un cuadro de texto que permite visualizar los mensajes recibidos por cada tópico subscrito. Estos se posicionaron según márgenes y alineaciones definidas en el código para una correcta visualización de los mensajes.

La excepción más destacable es el submenú de la conexión local, en este en solo una pantalla se pude monitorear de forma independiente cada sensor. Este cuenta con los datos del servidor local para establecer una conexión con Mosquitto y poder recibir los mensajes de cada tópico.

5 Carcasa para el hardware

En este capítulo se describe el diseño, realización y montaje de una carcasa para poder todo el hardware involucrado en el proyecto. Tanto para la Raspberry Pi como los diversos sensores utilizados.

5.1 Diseño 3D

Antes de diseñar se tomaron las medidas de todo el hardware involucrado.

- *Raspberry Pi 3*: La placa cuenta con 85 [mm] de largo con 56 [mm] de ancho y su punto más alto esta en los conectores USB siendo esta de aproximadamente 20 [mm] de altura.
- *DHT11*: Su placa cuenta con 36 [mm] de largo versus 14 [mm] de ancho, el sensor tiene 15 [mm] de largo y 13 [mm] de ancho.
- *LDR*: Este componente se colocara de forma externa por lo que no se hace necesario sus medidas.
- *HC-SR501*: La placa mide 32 [mm] de largo versus 29 [mm] de ancho, el foco del sensor es un cuadrado de 23x23 [mm].
- *MQ-135*: Su placa cuenta con 33 [mm] de largo versus 21 [mm de ancho], el sensor en si forma una circunferencia con un radio de 10 [mm].

Con las medidas hechas se propone crear una caja con forma de casa, donde su tejado hará la función de tapa. Según las medidas tomadas se prende construir un modelo en 3D. Su exterior será de 95 [mm] de largo, 66 [mm] de ancho y 84 [mm] de alto. Su interior será de 87 [mm] de largo, 58 [mm] de ancho y 80 [mm] de alto. El tejado se diseñara a la medida y contara con cuatro soportes para un encaje firme y seguro con la caja.

Para el diseño 3D se utilizó el programa SketchUp 2018 [25], en este primero se diseñó en su totalidad la caja base según las medidas tomadas. En la base se dejaron ventanas para los sensores, en el frente una para el sensor de movimiento y para los costados el sensor de temperatura y humedad, de lado opuesto el sensor de calidad de aire. Además de los espacios necesarios para conectar dispositivos por USB, Ethernet, HDMI y principalmente la alimentación por micro-USB. También en el frente se dejó un texto hundido indicando el nombre de la aplicación móvil y debajo de este las siglas de la Escuela de Ingeniería Eléctrica, EIE.

En el mismo programa se modelo la tapa de la caja, esta se realizó con forma de tejado y que contara con cuatro pilares para generar soporte y encaje con la base.

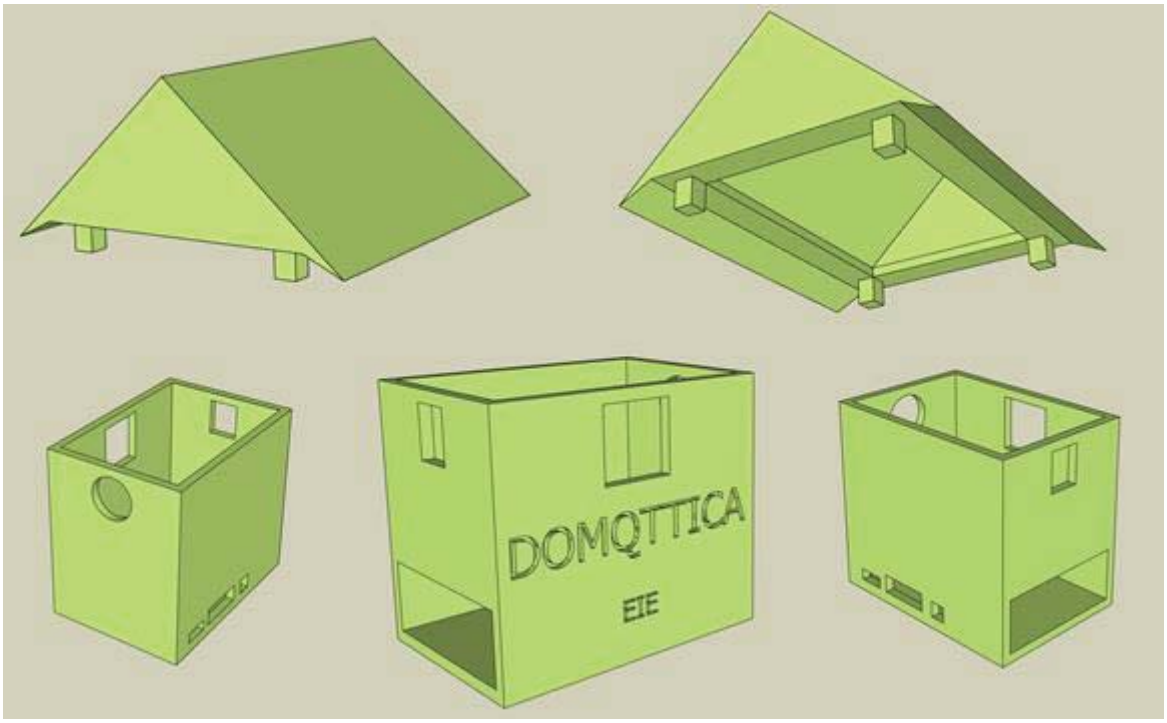

Figura 5-1 Modelo en 3D realizado para la base y tapa de la carcasa en SketchUp.

Con el diseño en 3D correctamente realizado, se deben exportar cada modelo en archivos ".stl" para poder ser imprimidos con una impresora 3D. Se debe mencionar que es importante comprobar con algún software lector de este tipo de archivo, que el modelo se encuentre con todas sus capas conectadas y bien enlazadas para evitar inconvenientes.

5.2 Resultado del diseño carcasa

Una vez impresas la base como la tapa, se obtuvo dos piezas de muy buena calidad y alta resistencia. En un principio ambas piezas no encajaban una con la otra, debido a que comúnmente se tiene un porcentaje de error en el grosor de algunas piezas durante su impresión.

Otro aspecto a mencionar es que la placa Raspberry Pi no encajaba con facilidad, ya que en el diseño no se consideraron los milímetros extras que presentaban la entrada HDMI, Micro-USB y la de alimentación. Esto se pudo solucionar mediante el lijado del material, logrando un encaje más fácil de la placa con la base de la carcasa.

Otra modificación que se le realizo a la carcasa de forma posterior, fueron dos orificios en el tejado en los cuales se colocara el LDR para monitorear la intensidad de luz superior. También se realizaron cuatro orificios en la parte trasera de la carcasa donde estará situado un pulsado para poder encender o apagar la Raspberry Pi y dos en la parte lateral izquierda para el LED indicador de estado del dispositivo.

5.3 Montaje carcasa con el harware

Contando con toda la programación requerida automatizada, la conexión de cada cable de los sensores con los pines GPIO de la Raspberry Pi, se procede con el montaje de la carcasa con todo el hardware involucrado.

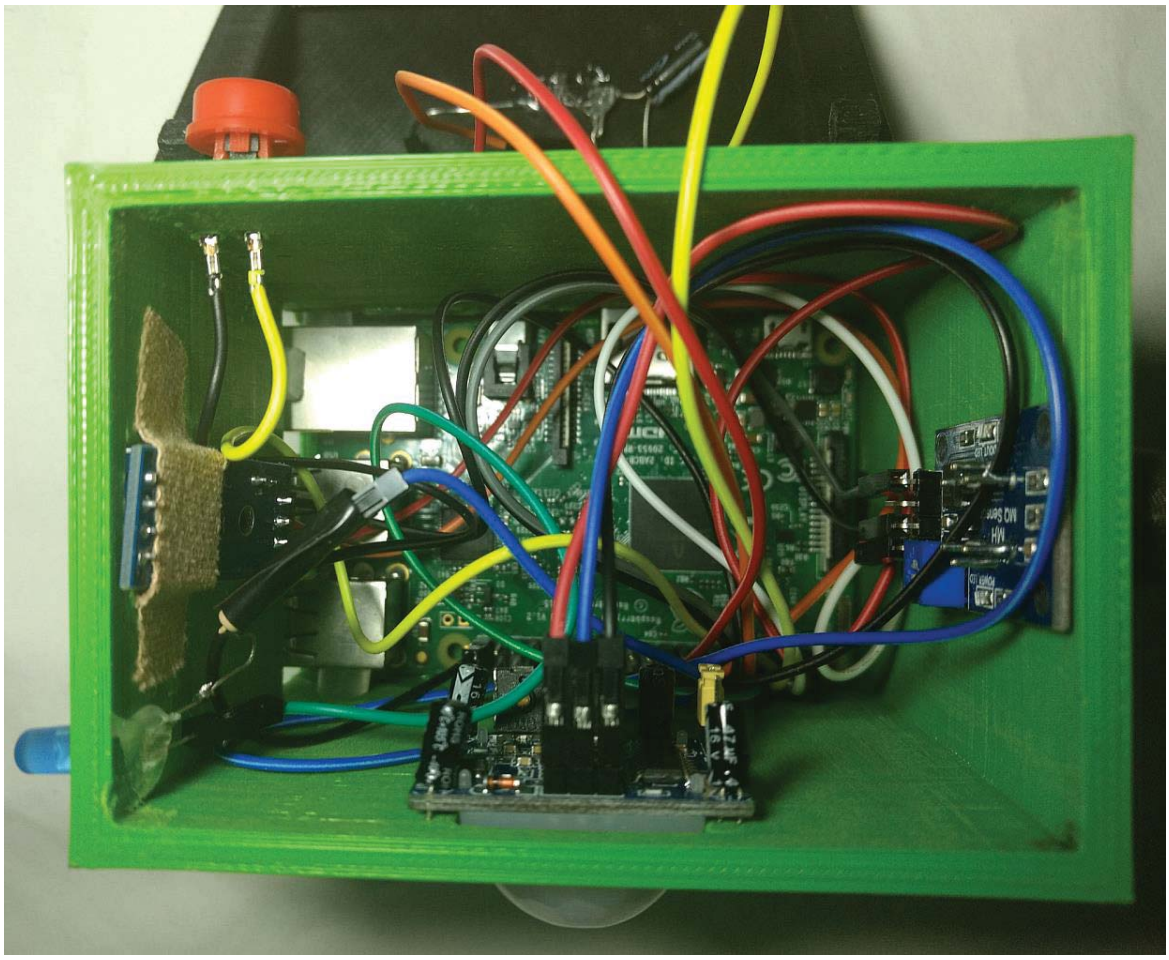

Figura 5-2 Vista superior de la carcasa con todos los dispositivos montados.

El sensor de calidad de aire MQ-135 encaja perfectamente, quedando de forma segura e inmóvil en su lugar. El sensor de temperatura y humedad DHT11 encaja muy bien en la carcasa, pero su placa queda un poco suelta en su parte inferior, para lograr una mayor firmeza esta se puede presionar usando cinta adhesiva. El sensor detector de movimiento HC-SR501 es el que queda con menos firmeza, el utilizar cinta adhesiva no se hace suficiente por lo que la mejor opción es utilizar un poco de silicona en sus costados o la utilización de un elemento externo para realizar presión donde se encuentra el sensor.

En las siguientes figuras se presentaran la visualización del frente, ambos costados y parte posterior de la carcasa con todos los sensores conectados.

Figura 5-3 Vista frontal y lateral derecha de la carcasa montada con los sensores.

Figura 5-4 Vista lateral izquierda y posterior de la carcasa montada con los sensores.

6 Aplicación terminada, resultados

En este capítulo final se presentan los resultados obtenidos con el dispositivo portátil para la toma de datos, los de la aplicación móvil desarrollada y con el protocolo MQTT.

6.1 Resultados del dispositivo portátil

Con todos los dispositivos montados en la carcasa, se procede a comprobar que todo esté funcionando correctamente. Se alimenta la Raspberry Pi con un cargador micro-usb, encendiéndose el LED azul, indicando que está funcionando. Después de treinta segundos aproximadamente el sistema operativo se ejecuta y luego de otros diez segundos los programas en Python comienza a enviar datos. Con todos los servicios y programas activos solo se utiliza un 22 [%] de la CPU de la Raspberry Pi. También se comprueba el correcto funcionamiento del botón de apagado/encendido físico, para esto se presiona el pulsador en la parte trasera del dispositivo, resultando completamente funcional, apangando o encendiendo la Raspberry Pi.

Esto demuestra que el montaje y la conexión entre los distintos dispositivos quedo realizado de forma correcta, al igual que toda la programación involucrada.

6.2 Resultados de la App DoMQTTica

La aplicación móvil en Android Studio desarrollada se construye el proyecto en un archivo .apk dando los siguientes resultados.

Nombre:	DoMQTTica.apk
Tamaño:	2.11 MB
Nombre Paquete:	com.proto.domqttica.domqtticaver1

Figura 6-1 Datos de la App DoMQTTica.

Una vez que se instala la aplicación en un Smartphone está ocupa 17.04 [MB] de almacenamiento, en tres horas continuas de actividad utiliza 67 [MB] de memoria Ram del Smatphone. Si se está utilizando una red móvil para monitorear los datos, está en un primer plano esta gasta en datos móviles 0.11 [KB/s] y ejecutándose en un segundo plano 0.028 [KB/s]. Si se usa la aplicación en un primer plano para monitorear los datos durante cinco minutos se gastan 33 [KB] en datos móviles. Todo lo anterior mencionado demuestra el bajo consumo de batería y datos que tiene la aplicación móvil desarrollada. Además queda en evidencia la ligereza de la utilización del protocolo de comunicación MQTT.

Una vez que se inicia la aplicación, se visualiza su splash para luego pasar al menú principal donde se indica que se realizó una conexión con el servidor CloudMQTT de manera exitosa o con el servidor local si se requiere. Se empiezan a recibir los datos publicados por cada sensor, al igual que en cada submenú. En la siguiente figura se presentan capturas del funcionamiento de la aplicación. De esto queda corroborado el buen resultado de la aplicación desarrollada para monitorear distintos sensores usados en domótica por medio del protocolo MQTT.

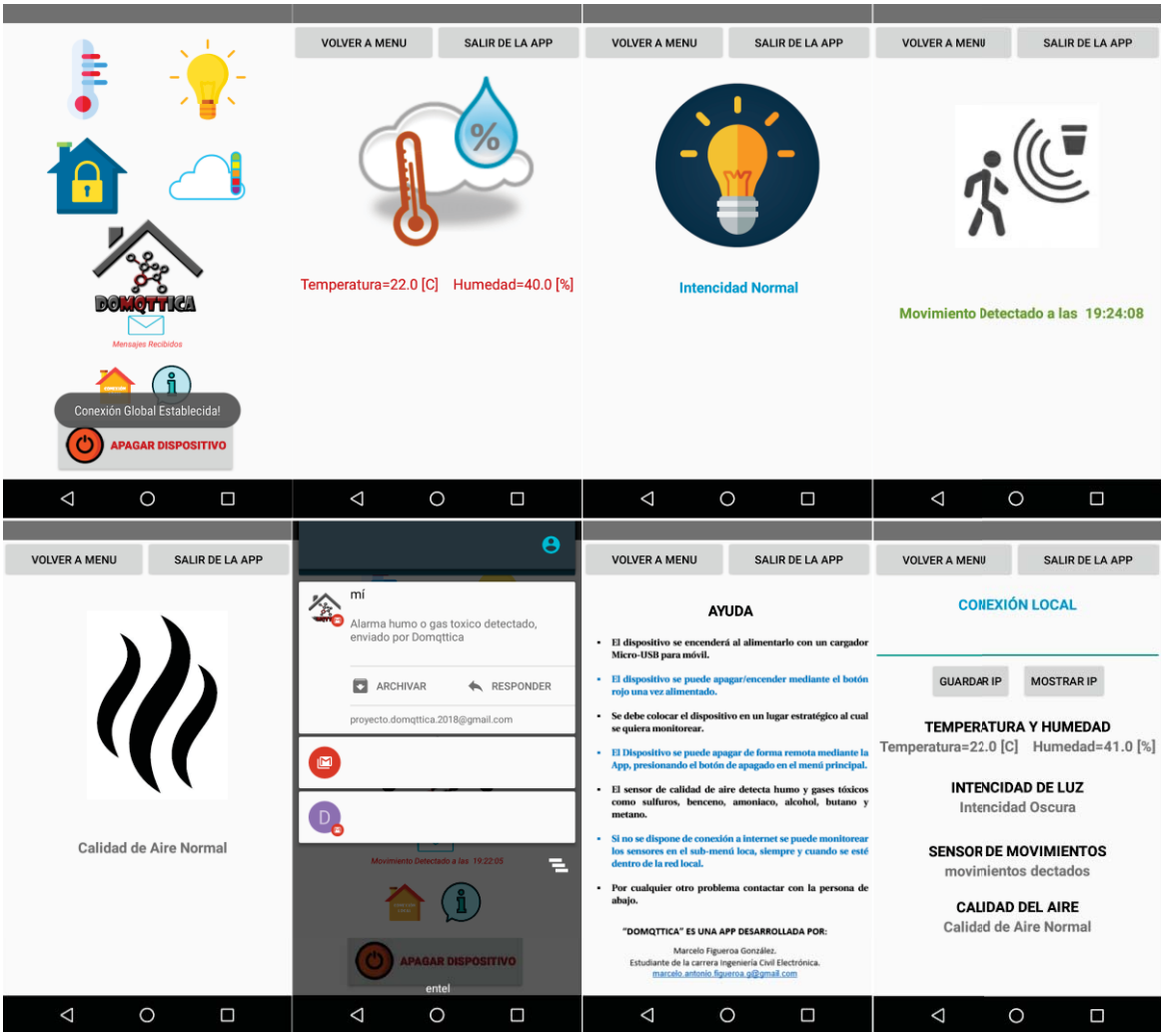

Figura 6-2 Capturas del funcionamiento de la aplicación móvil DoMQTTica.

6.3 Resultados utilización del protocolo MQTT

La utilización del protocolo MQTT a lo largo del proyecto es de gran importancia para la comunicación. Los resultados del uso de este protocolo quedaron demostrados en el correcto funcionamiento de la aplicación móvil DoMQTTica. Es por esto que se presentara un diagrama secuencial del funcionamiento del protocolo para lograr la comunicación.

En el diagrama se muestra como se realiza la comunicación desde el sensor de temperatura y humedad DHT11, hasta la APP. El diagrama es válido para cada sensor usado, solo cambia la identidad del cliente, tópico a publicar y subscribirse.

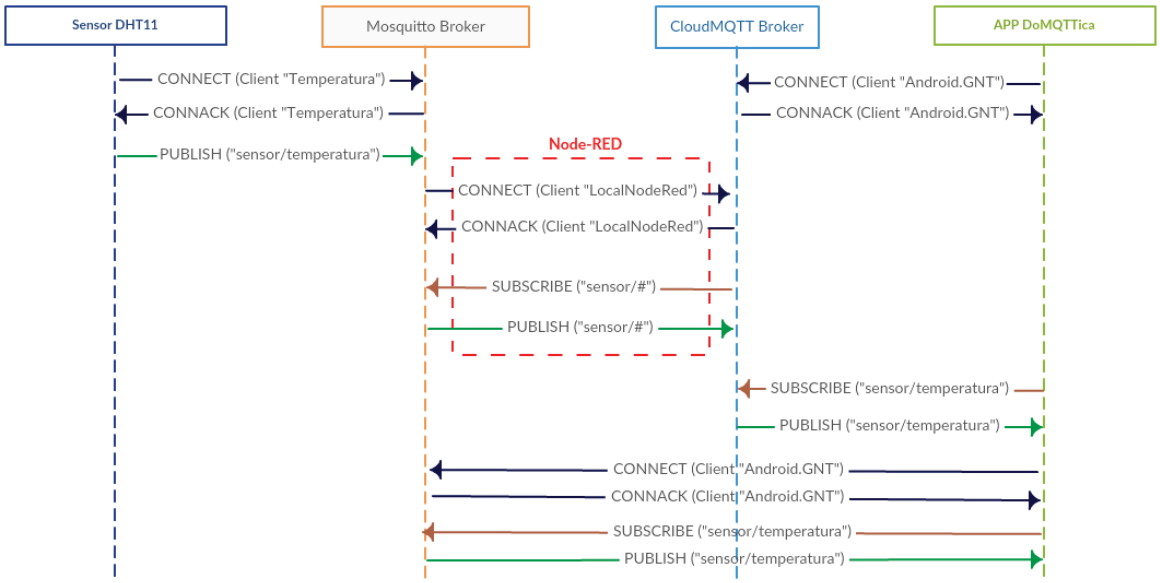

Figura 6-3 Diagrama secuencial de la comunicación desde el sensor a la App por medio de MQTT.

Lo que acontece es que en un principio el sensor se conecta con el servidor local Mosquitto, el que lo identifica y empieza a recibir las publicaciones de los datos del sensor. A su vez el servidor local se conecta al servidor en la nube CloudMQTT, el cual se subscribe a todas las publicaciones que envíen los distintos sensores al servidor local Mosquitto. Todo esto gracias al puente realizado por medio de la herramienta Node-RED. Simultáneamente el cliente generado por Paho en Android se conectara al servidor en la nube y se subscribirá al tópico correspondiente recibiendo los datos que publica el sensor. Además se cuenta con una conexión directa desde la App al servidor local en caso de que se esté usando la red local sin acceso a internet.

Discusión y conclusiones

La domótica se ha convertido en una de las prioridades constructivas y tecnológicas de los últimos años. La demanda de accesorios que generan un mayor bienestar, seguridad y comodidad en los hogares e industria, son cada vez más demandados. En la actualidad a nivel mundial, encontramos múltiples servicios, aplicaciones y funcionalidades que permiten centralizar toda la actividad del hogar o empresa en uno o varios dispositivos, contribuyendo así, al ahorro energético y económico. Se espera que en un futuro cercano a escala global, más de 30 millones de hogares y empresas estén domotizadas parcial o totalmente.

El aporte del Internet de las Cosas (IoT) en el modo de vivir y trabajar de las personas se está posicionando firmemente en nuestro país. No obstante el uso de tecnologías relacionadas al funcionamiento inteligente y conectado, con sistemas de monitoreo y control a distancia, tienen grandes desafíos por resolver. Un plus extra que cuenta Chile es la gran base de teléfonos inteligentes, junto con la adaptación a la revolución digital que se está dando estos últimos años. Es por esto que el proyecto busca usar como base estas ventajas a nivel país, para dar solución a las problemáticas planteadas en un comienzo. Tomando como eje la utilización del protocolo de comunicación MQTT debido al estudio y fundamentos del proyecto “Estudio del protocolo MQTT y sus potenciales aplicaciones en el ambiente comercial/industrial”.

Este proyecto se presenta como un verdadero desafío, debido a la cantidad de diversas tecnologías a estudiar y usar. Desde conocer a fondo el funcionamiento del protocolo MQTT, como utilizarlo en distintos lenguajes de programación, trabajar con dispositivos IoT, desarrollo de aplicaciones móviles, hasta el modelado y diseño en 3D. Pero el desarrollo de este proyecto será de bastante utilidad para seguir avanzando en la innovación tecnología en Chile.

Respecto a lo trabajado y estudiado del protocolo de comunicación MQTT, se dio a conocer la librería de la fundación Eclipse Paho la cual tiene como objetivo que este protocolo pueda ser usado en diversos lenguajes de programación. Esto demuestra la importancia que está tomando MQTT actualmente en el mundo para aplicaciones IoT. Con los resultados obtenidos del uso del protocolo MQTT se verifico que este resulta ideal para aplicaciones móviles, dado su bajo consumo de energía y distribución eficiente para el intercambio de información de forma fiable, en una red con conectividad no confiable. Haciéndolo ideal para dispositivos móviles como para sensores, siendo un protocolo altamente escalable.

El uso de hardware centrado en IoT, como los sensores y la Raspberry Pi, permiten generar proyectos que den solución a problemas de diversas índoles a un bajo costo. Otra ventaja de estos dispositivos es su reducido tamaño y altas capacidades. Por estas cualidades se pudo desarrollar un dispositivo portátil de tamaño reducido que permite realizar un monitoreo del lugar que se encuentre. La placa Raspberry Pi como cerebro para la conexión de sensores y el funcionamiento de servicios y programas, usando muy pocos recursos de la misma.

Desarrollar programas en el lenguaje de programación Python para la toma de datos y poder publicar estos usando el protocolo MQTT, no resulto una tarea compleja. Lo mencionado se debe a que se contaba con conocimientos de lenguajes de programación similares a Python, adquiridos a lo largo de la carrera en cursos de programación. Esto no le resta mérito al trabajo realizado, ya que de igual forma se debe manejar de forma correcta este lenguaje. Dado que la adecuada ejecución de los programas desarrollados significan el principio para realizar una correcta lectura de los datos obtenidos por los sensores conectados a la Raspberry Pi.

Uno de los mayores desafíos para realizar el proyecto, fue el desarrollo de la aplicación móvil para Android. Ya que no se contaba con la experiencia necesaria en el lenguaje de programación Java y el software Android Studio. Esto se solvento en base a un arduo trabajo de investigación y desarrollo de aplicaciones móviles sencillas, significando la etapa en la cual se dedicó más tiempo para la realización de proyecto. Con esto se adquirió la experiencia necesaria para poder crear la aplicación objetivo del proyecto, la cual paso por diversas modificaciones para su correcto funcionamiento y optimización. Dando como resultado la aplicación que se esperaba para poder monitorear diversos sensores desde el teléfono móvil.

Como conclusión final del proyecto, a lo largo del desarrollo del mismo se logró cumplir con el objetivo general y los específicos planteados desde un comienzo. Significando esto como resultado una aplicación móvil para el monitoreo de sensores aplicados en domótica en base al protocolo MQTT. El dispositivo portátil desarrollado es estéticamente correcto y visualmente llamativo, al igual que la aplicación móvil creada que cuenta con un nombre, slogan e icono muy intuitivo en su funcionamiento y uso. El trabajo realizado en este proyecto tiene dos puntos muy importantes, el primero es que gracias a este se adquirió la experiencia y conocimiento en el desarrollo de aplicaciones móviles para Android y el trabajo con dispositivos IoT, aumentando las habilidades que se tienen como futuro Ingeniero Civil Electrónico de la EIE. El segundo punto importante, es la proyección que tiene la idea del proyecto realizado para nuevas iteraciones del mismo y poder ser un producto comercial en un futuro cercano.

Trabajo Futuro

Respecto al trabajo realizado para este proyecto que involucra domótica, IoT, aplicaciones móviles y el protocolo MQTT. Es un piso bastante sólido que se puede tomar como base para múltiples mejoras e ideas que surgieron a lo largo del desarrollo del proyecto y que no se pudieron implementar.

La primera de estas es poder colocar distintos sensores en diversos lugares del hogar sin la necesidad de que estos estén conectados de forma directa con la placa Raspberry Pi. Esto se puede lograr mediante la utilización del módulo WIFI ESP8266, que permite conectar sensores usados en domótica, enviado los datos adquiridos al dispositivo principal. Con esto se lograría tener módulos portátiles y más pequeños en lugares claves según la función de cada uno de estos. Esta mejora le dará más versatilidad y utilidad al proyecto.

Un punto ausente a desarrollar es el otro factor importante en la domótica, el control. En base a los datos que obtienen los sensores se puede implementar actuadores con funciones definidas, estos pueden ser la activación de la ventilación por altas temperaturas, el apagado remoto de luces para el ahorro energético, la activación de alarmas, el control de cerradoras, etc. Diversas aplicaciones que darán un alto valor comercial al proyecto.

También se puede mejorar y pulir la aplicación móvil para Android. Estas mejoras pueden ser la de agregar una base de datos para que solo puedan monitorear desde la App usuarios que previamente se identificaron. El agregar más opciones si se desarrolla el control como previamente se mencionó o si se aumenta la cantidad de sensores conectados. La aplicación móvil al utilizar el protocolo de comunicación MQTT puede servir para crear otras App que también utilicen este protocolo en base a lo estudiado y desarrollado para establecer la comunicación con el intercambio de mensajes de forma fiable.

Para el trabajo a futuro de este proyecto pueden surgir otras mejoras o ideas que no se han considerado ya que este proyecto se presenta de manera muy escalable. Cada una de las posibles iteraciones futuras le dará un mayor valor, versatilidad, utilidad y aplicación al trabajo que se realizó.

Bibliografía

[1] J. Morales Ruiz, «ESTUDIO DEL PROTOCOLO MQTT Y SUS POTENCIALES APLICACIONES EN AMBIENTE COMERCIAL/INDUSTRIAL,» PUCV, Valparaíso, 2016.

[2] J. P. Majluf, «iab.trends,» 2016. [En línea]. Available: http://iabtrends.cl/2016/08/09/chile-lidera-el-uso-de-smartphones-en-latinomaerica-con-7-9-millones-de-usuarios/.

[3] F. Águila, «Emol: Accidentes en el hogar,» 22 Junio 2013. [En línea]. Available: http://www.emol.com/noticias/nacional/2013/06/21/604965/accidentes-intradomiciliarios-incendios-son-la-principal-causa-de-muerte-en-ninos--fin-de-semana.html.

[4] J. Constine, «Techcrunch,» 12 Abril 2017. [En línea]. Available: https://techcrunch.com/2017/04/12/messenger/.

[5] Intel, «Aplicaciones de la Internet de las cosas en las industrias,» [En línea]. Available: https://www.intel.la/.

[6] A. Johari, «Android Tutorial – Learn Android From Scratch!,» edureka!, 21 Junio 2017. [En línea]. Available: https://www.edureka.co/.

[7] S. Electric, «See-Home: Visualización y control en su Smartphone,» [En línea]. Available: http://www.coevagi.com/Docs/Sch_See-Home.pdf.

[8] Microsoft, «Definición de las siete capas del modelo OSI,» [En línea]. Available: https://support.microsoft.com/es-cl/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained.

[9] J. Dunn, «Busuness Insider,» [En línea]. Available: http://www.businessinsider.com/smartphone-market-share-android-ios-windows-blackberry-2016-8.

[10] R. C. Cruceira, «Raspberry Pi: características y aplicaciones,» [En línea]. Available: https://ingenierate.com.

[11] «DHT11 Humidity and Temperature,» [En línea]. Available: https://www.microbot.it/documents/mr003-005_datasheet.pdf.

[12] «Light Dependent Resistor - Datasheet,» [En línea]. Available: http://kennarar.vma.is/thor/v2011/vgr402/ldr.pdf.

[13] «Manual PIR HC-SR501,» [En línea]. Available: https://puntoflotante.net/MANUAL-DEL-USUARIO-SENSOR-DE-MOVIMIENTO-PIR-HC-SR501.pdf.

[14] «MQ-135 GAS SENSOR,» [En línea]. Available: https://www.olimex.com/Products/Components/Sensors/SNS-MQ135/resources/SNS-MQ135.pdf.

[15] U. P. d. Valencia, «Máster en Desarrollo de Aplicaciones Android,» [En línea]. Available: http://www.androidcurso.com/.

[16] TIOBE, «The Python Programming Language,» [En línea]. Available: https://www.tiobe.com/tiobe-index/python/.

[17] Google, «Android Studio,» [En línea]. Available: https://developer.android.com/.

[18] E. Mosquitto. [En línea]. Available: https://mosquitto.org/man/mosquitto-conf-5.html.

[19] CloudMQTT, «CloudMQTT,» [En línea]. Available: https://www.cloudmqtt.com/ .

[20] IBM, «Node-RED,» [En línea]. Available: https://nodered.org/.

[21] Eclipse, «Paho,» [En línea]. Available: https://www.eclipse.org/paho/.

[22] «fritzing,» [En línea]. Available: http://fritzing.org.

[23] NinjaMock, «NinjaMock,» [En línea]. Available: https://ninjamock.com/.

[24] HiveMQ, «Paho Android Service,» [En línea]. Available: https://www.hivemq.com/.

[25] «SketchUp: 3D modeling for everyone,» [En línea]. Available: https://www.sketchup.com/.

A Codigos desarrollados Python/Java

En este apéndice se presenta de forma detallada las líneas de código de los distintos programas en Python desarrollados, así como las de la aplicación móvil para Android en Java.

A.1 Código: dht11.py

```
# Importar las librerias necesarias
import sys
import time
import Adafruit_DHT
import paho.mqtt.client as mqttClient

time.sleep(10)

#Conexion MQTT
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conexion Establecida")
        global Connected
        Connected = True
    else:
        print("Conexion Fallida!")

Connected = False

broker_address= "localhost"
port = 1883
user = "domqttica"
password = "eie655"

client = mqttClient.Client("Temperatura")
client.username_pw_set(user, password=password)
client.on_connect= on_connect
client.connect(broker_address, port=port)

# Configuracion del tipo de sensor DHT
sensor = Adafruit_DHT.DHT11

# Configuracion del puerto GPIO al cual esta conectado  (GPIO 23)
pin = 23

# Intenta ejecutar las siguientes instrucciones, si falla va a la instruccion except
try:
        # Ciclo principal infinito
        while True:
# Obtiene la humedad y la temperatura desde el sensor
        humedad, temperatura = Adafruit_DHT.read_retry(sensor, pin)
# Imprime en la consola las variables temperatura y humedad con un decimal
                print('Temperatura={0:0.1f}*  Humedad={1:0.1f}%'.format(temperatura,
humedad))
```

```
                client.publish("sensor/temperatura",'Temperatura={0:0.1f} [C]
Humedad={1:0.1f} [%]'.format(temperatura, humedad))

                # Duerme 10 segundos
                time.sleep(10)

except KeyboardInterrupt: # Si se pulsa Ctrl+C
    client.disconnect()
    GPIO.cleanup()
```

A.2 Código: ldr.py

```
# Importar las librerias necesarias
import RPi.GPIO as GPIO, time
import sys
import paho.mqtt.client as mqttClient

time.sleep(10)

# Se configura la GPIO en el modo BOARD lo que significa que sigue la numeracion fisica
de los pines
GPIO.setmode(GPIO.BOARD)
aux = False  # Se define una variable auxiliar para el programa

#Conexion MQTT
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conexion Establecida")
        global Connected
        Connected = True
    else:
        print("Conexion Fallida!")

Connected = False

broker_address= "localhost"
port = 1883
user = "domqttica"
password = "eie655"

client = mqttClient.Client("Luz")
client.username_pw_set(user, password=password)
client.on_connect= on_connect
client.connect(broker_address, port=port)

# Se define la funcion pines y tiempo de lectura
def timer (pin):
        reading = 0
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, GPIO.LOW)
        time.sleep(2)
        GPIO.setup(pin, GPIO.IN)
        while (GPIO.input(pin) == GPIO.LOW):
                reading += 1
        return reading

# Estrucura de control del programa mediante condicionales
while not aux:
        luz = timer(10)  # Lectura del pin fisico 10 de la RPi
        client.publish("sensor/varluz",luz)

        if luz>100 and luz<1000:
                print ("Intencidad Normal:", (luz))
                client.publish("sensor/luz","Intencidad Normal")

        if luz>1000:
                print ("Intencidad Oscura:", (luz))
```

```
                client.publish("sensor/luz","Intencidad Oscura")

        if luz<100:
                print ("Intencidad Alta:", (luz))
                client.publish("sensor/luz","Intencidad Alta")

except KeyboardInterrupt: # Si se pulsa Ctrl+C
    print "Finaliza deteccion"
    client.disconnect()
    GPIO.cleanup()
```

A.3 Código: hcsr501.py

```
# Importar las librerias necesarias
import RPi.GPIO as GPIO
import time
from time import gmtime, strftime
import paho.mqtt.client as mqttClient

time.sleep(10)

#Conexion MQTT
def on_connect(client, userdata, flags, rc):
    if rc == 0:

        print("Conexion Establecida")
        global Connected
        Connected = True
    else:
        print("Conexion Fallida!")

Connected = False

broker_address= "localhost"
port = 1883
user = "domqttica"
password = "eie655"

client = mqttClient.Client("Seguridad")
client.username_pw_set(user, password=password)
client.on_connect= on_connect
client.connect(broker_address, port=port)
client.publish("sensor/seguridad","Conexion Lista")

# Configura los pines GPIo como BCM
GPIO.setmode(GPIO.BCM)
PIR_PIN = 7
mess = "Movimiento Detectado a las "
GPIO.setup(PIR_PIN, GPIO.IN)

try:
    while True:  # Se inicia un bucle infinito
        if GPIO.input(PIR_PIN):
                time.sleep(1)  # Pausa de 1 segundo
                timex = time.strftime("%X")
                print "Movimiento Detectado a las " + timex
                client.publish("sensor/seguridad",'{} {}'.format(mess, timex))
                time.sleep(1)  # Pausa 1 segundo

except KeyboardInterrupt: # Si se pulsa Ctrl+C
    print "Finaliza deteccion"
    client.disconnect()
    GPIO.cleanup()
```

A.4 Código: mq135.py

```
# Importar las librerias necesarias
import RPi.GPIO as GPIO
import time
from time import gmtime, strftime
import paho.mqtt.client as mqttClient
import smtplib

time.sleep(10)

#Conexion MQTT
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conexion Establecida")
        global Connected
        Connected = True
    else:
        print("Conexion Fallida!")
Connected = False

broker_address= "localhost"
port = 1883
user = "domqttica"
password = "eie655"

client = mqttClient.Client("Calidad")
client.username_pw_set(user, password=password)
client.on_connect= on_connect
client.connect(broker_address, port=port)

# Configura los pines GPIo como BCM y se definen los pines
GPIO.setmode(GPIO.BCM)
GPIO.setmode(GPIO.BCM)
PIR_PIN = 4
GPIO.setup(PIR_PIN, GPIO.IN)
normal = "Calidad de Aire Normal"
contaminado = "Contaminado detectado a las "
aux = False  # Se define una variable auxiliar para el programa

try:
 while not aux:
        if GPIO.input(PIR_PIN) == 1:
            time.sleep(1)  # Pausa de 1 segundo
            timex = time.strftime("%X")
            print "Contaminacion detectada a las " + timex
           client.publish("sensor/calidad",'{} {}'.format(contaminado, timex))
           fromaddr = 'proyecto.domqttica.2018@gmail.com'
           toaddrs  = 'proyecto.domqttica.2018@gmail.com'
           msg = 'Alarma humo o gas toxico detectado, enviado por Domqttica'
            username = 'proyecto.domqttica.2018@gmail.com'
            password = 'domqttica2018'
           server = smtplib.SMTP('smtp.gmail.com:587')
           server.starttls()
           server.login(username,password)
           server.sendmail(fromaddr, toaddrs, msg)
           server.quit()
           time.sleep(7)  # Pausa 7 segundo

       else:
            print("Calidad del Aire normal")
            client.publish("sensor/calidad",normal)
            time.sleep(5)

 except KeyboardInterrupt:
    print "Finaliza deteccion"
    client.disconnect()
    GPIO.cleanup()
```

A.5 Código: power.py

```
import RPi.GPIO as GPIO
import time
import os

GPIO.setmode(GPIO.BCM)
GPIO.setup(03, GPIO.IN, pull_up_down = GPIO.PUD_UP)
while True:
    print GPIO.input(03)
    if(GPIO.input(03) == False):
        os.system("sudo shutdown -h now")
        break
    time.sleep(1)
```

A.6 Código: powermqtt.py

```
import paho.mqtt.client as mqttClient
import time
import os
from collections import Counter

time.sleep(10)

lista = "apagar"
mensaje = []

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conexion Establecida")
        global Connected
        Connected = True
    else:
        print("Conexion Fallida")

def on_message(client, userdata, message):
        print "Mensaje: "  + message.payload
        mensaje = message.payload
        if Counter(lista) == Counter(mensaje):
                os.system("sudo shutdown -h now")

Connected = False

broker_address= "localhost"
port = 1883
user = "domqttica"
password = "eie655"

client = mqttClient.Client("Python")
client.username_pw_set(user, password=password)
client.on_connect= on_connect
client.on_message= on_message
client.connect(broker_address, port=port)
client.loop_start()

while Connected != True
    time.sleep(0.1)

client.subscribe("domqttica/power")

try:
    while True:
        if lista == 1:
                print("Intencidad Normal")
        time.sleep(1)
```

```
except KeyboardInterrupt:
    client.disconnect()
    client.loop_stop()
```

A.7 Codigo Node-RED

```
[{"id":"b652da11.ec3788","type":"mqtt
in","z":"68f6c3f2.7aa78c","name":"","topic":"sensor/#","qos":"0","broker":"8dc029f2.bfa0
88","x":210,"y":198,"wires":[["4632997a.402eb8"]]},{"id":"4632997a.402eb8","type":"mqtt
out","z":"68f6c3f2.7aa78c","name":"","topic":"","qos":"","retain":"","broker":"c8db803f.
1035e","x":534,"y":199,"wires":[]},{"id":"e77bfe53.fde1f","type":"mqtt
out","z":"68f6c3f2.7aa78c","name":"","topic":"domqttica/power","qos":"0","retain":"","br
oker":"b16e6f60.3fad7","x":531,"y":341,"wires":[]},{"id":"6146967.4f52e68","type":"mqtt
in","z":"68f6c3f2.7aa78c","name":"","topic":"domqttica/power","qos":"2","broker":"c8db80
3f.1035e","x":228,"y":340,"wires":[["e77bfe53.fde1f"]]},{"id":"889c3223.cba36","type":"c
omment","z":"68f6c3f2.7aa78c","name":"Publicación Datos
Sensores","info":"","x":274,"y":130,"wires":[]},{"id":"93be0fc5.2c4cb","type":"comment",
"z":"68f6c3f2.7aa78c","name":"Apagado Raspberry Mediante
App","info":"","x":288,"y":292,"wires":[]},{"id":"8dc029f2.bfa088","type":"mqtt-
broker","z":"","broker":"localhost","port":"1883","clientid":"","usetls":false,"compatmo
de":false,"keepalive":"60","cleansession":true,"willTopic":"","willQos":"0","willPayload
":"","birthTopic":"","birthQos":"0","birthPayload":""},{"id":"c8db803f.1035e","type":"mq
tt-broker","z":"","broker":"m14.cloudmqtt.com","port":"18051","clientid":"CloudMQTT-
NodeRed","usetls":false,"compatmode":true,"keepalive":"60","cleansession":false,"willTop
ic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","birthPayload":""},
{"id":"b16e6f60.3fad7","type":"mqtt-
broker","z":"","broker":"Localhost","port":"1883","clientid":"LocalNodeRed","usetls":fal
se,"compatmode":true,"keepalive":"15","cleansession":true,"willTopic":"","willQos":"0","
willPayload":"","birthTopic":"","birthQos":"0","birthPayload":""}]
```

A.8 Código aplicación DoMQTTica

En este apartado del apéndice se presentan todas las líneas de código involucrados en los archivos importantes programadas para el desarrollo de la aplicación móvil en Android Studio.

A.8.1 Código: menu.java

```
package com.proto.domqttica.domqtticaver1;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
```

```
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import java.io.UnsupportedEncodingException;

public class menu extends AppCompatActivity {

    ImageView temp;
    ImageView luz;
    ImageView segur;
    ImageView calid;
    ImageView configuracion;
    ImageView ayuda;
    static String MQTTHOST = "tcp://m14.cloudmqtt.com:19165";
    static String USERNAME = "cjkjjjze";
    static String PASSWORD = "WFJKWObn6WqQ";
    String topicStrtodosub = "sensor/#";
    MqttAndroidClient client;
    TextView subtodo;
    Button info;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu);
        subtodo = (TextView)findViewById(R.id.button);
        temp = (ImageView) findViewById(R.id.btntemphum);
        luz = (ImageView)findViewById(R.id.btnluz);
        segur = (ImageView)findViewById(R.id.btnseguridad);
        calid = (ImageView)findViewById(R.id.btncalidadair);
        configuracion = (ImageView)findViewById(R.id.btnconfig);
        ayuda = (ImageView)findViewById(R.id.btnayuda);

        temp.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent menu2 = new Intent(menu.this, temperatura.class);
                startActivity(menu2);
            }
        });

        luz.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent menu3 = new Intent(menu.this, luminosidad.class);
                startActivity(menu3);
            }
        });

        segur.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent menu4 = new Intent(menu.this, seguridad.class);
                startActivity(menu4);
            }
        });

        calid.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent menu5 = new Intent(menu.this, calidad.class);
                startActivity(menu5);
            }
        });

        configuracion.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent menu8 = new Intent(menu.this, config.class);
                startActivity(menu8);
            }
```

```
        });

        ayuda.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent menu9 = new Intent(menu.this, help.class);
                startActivity(menu9);
            }
        });

        String clientId = MqttClient.generateClientId();
        client = new MqttAndroidClient(this.getApplicationContext(),MQTTHOST, clientId);

        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(USERNAME);
        options.setPassword(PASSWORD.toCharArray());

        try {
            IMqttToken token = client.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Toast.makeText(menu.this, "Conexión Global Establecida!",
Toast.LENGTH_LONG).show();
                    setSubscription();
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken, Throwable exception)
{
                    Toast.makeText(menu.this, "Conexión Global Fallida!",
Toast.LENGTH_LONG).show();
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
        client.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {
            }

            @Override
            public void messageArrived(String topic, MqttMessage message) throws
Exception {
                subtodo.setText(new String(message.getPayload()));
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
            }
        });
    }

    private void setSubscription(){
        try {
            client.subscribe(topicStrtodosub, 0);
        }catch (MqttException e){
            e.printStackTrace();
        }
    }
    public void pub(View v){
        String topic = "domqttica/power";
        String message = "apagar";
        try {
            client.publish(topic, message.getBytes(), 0, false);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
}
```

A.8.2 Código: activity_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.proto.domqttica.domqtticaver1.menu">
    <Button
        android:id="@+id/btnpower"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginStart="65dp"
        android:layout_marginTop="1050dp"
        android:drawableLeft="@drawable/btnpower"
        android:onClick="pub"
        android:text="  Apagar Dispositivo"
        android:textAlignment="viewStart"
        android:textColor="@android:color/holo_red_dark"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/relativeLayout"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.493" />
    <RelativeLayout
        android:id="@+id/relativeLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center_horizontal"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">
        <ImageView
            android:id="@+id/btntemphum"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_marginRight="50dp"
            android:layout_marginTop="20dp"
            android:src="@drawable/temp2" />
        <ImageView
            android:id="@+id/btnluz"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_marginTop="20dp"
            android:layout_toRightOf="@+id/btntemphum"
            android:src="@drawable/luz" />
        <ImageView
            android:id="@+id/btnseguridad"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_below="@+id/btntemphum"
            android:layout_marginRight="50dp"
            android:layout_marginTop="20dp"
            android:src="@drawable/seguridad" />
        <ImageView
            android:id="@+id/btncalidadair"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_below="@+id/btnluz"
            android:layout_marginTop="20dp"
            android:layout_toRightOf="@+id/btnseguridad"
            android:src="@drawable/aire" />
        <ImageView
            android:id="@+id/btnmap"
            android:layout_width="250dp"
            android:layout_height="150dp"
```

```
            android:layout_below="@+id/btnseguridad"
            android:layout_marginTop="2dp"
            android:src="@drawable/mapa" />
        <TextView
            android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignEnd="@+id/btnmap"
            android:layout_alignLeft="@+id/btnmap"
            android:layout_alignRight="@+id/btnmap"
            android:layout_alignStart="@+id/btnmap"
            android:layout_below="@+id/btnmap"
            android:text="Mensajes Recibidos"
            android:textAlignment="center"
            android:textColor="@android:color/holo_red_light"
            android:textSize="10sp"
            android:textStyle="italic"
            app:layout_constraintStart_toStartOf="parent" />
        <ImageView
            android:id="@+id/btnconfig"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_below="@+id/button"
            android:layout_marginLeft="60dp"
            android:layout_marginRight="20dp"
            android:layout_marginTop="20dp"
            android:src="@drawable/config" />
        <ImageView
            android:id="@+id/btnayuda"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_below="@+id/button"
            android:layout_marginTop="20dp"
            android:layout_toRightOf="@+id/btnconfig"
            android:src="@drawable/info" />
    </RelativeLayout>
</android.support.constraint.ConstraintLayout>
```

A.8.3 Código: temperatura.java

```
package com.proto.domqttica.domqtticaver1;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

public class temperatura extends AppCompatActivity {

    Button volver1;
    Button salir1;
    static String MQTTHOST = "tcp://m14.cloudmqtt.com:19165";
    static String USERNAME = "cjkjjjze";
    static String PASSWORD = "WFJKWObn6WqQ";
    String topicStrtempsub = "sensor/temperatura";
    MqttAndroidClient client;
```

```
    TextView subtemp;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_temperatura);
        subtemp = (TextView)findViewById(R.id.subtemp);
        volver1 = (Button)findViewById(R.id.btnvolver1);
        salir1 = (Button)findViewById(R.id.btnsalir1);
        volver1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                onBackPressed();
            }
        });

        salir1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });

    String clientId = MqttClient.generateClientId();
    client = new MqttAndroidClient(this.getApplicationContext(),MQTTHOST, clientId);
    MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(USERNAME);
        options.setPassword(PASSWORD.toCharArray());

        try {
        IMqttToken token = client.connect(options);
        token.setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                Toast.makeText(temperatura.this, "Conexión Global Establecida!",
Toast.LENGTH_LONG).show();
                setSubscription();
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                Toast.makeText(temperatura.this, "Conexión Global Fallida!",
Toast.LENGTH_LONG).show();
            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }
        client.setCallback(new MqttCallback() {
        @Override
        public void connectionLost(Throwable cause) {
        }

        @Override
        public void messageArrived(String topic, MqttMessage message) throws Exception {
            subtemp.setText(new String(message.getPayload()));
        }

        @Override
        public void deliveryComplete(IMqttDeliveryToken token) {

        }
    });
}
    private void setSubscription(){
        try {
            client.subscribe(topicStrtempsub, 0);
        }catch (MqttException e){
```

```
            e.printStackTrace();
        }
    }
}
```

A.8.4 Código: activity_temperatura.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.proto.domqttica.domqtticaver1.temperatura">
    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/btnvolver1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Volver a Menu" />
        <Button
            android:id="@+id/btnsalir1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Salir de la App" />
    </LinearLayout>
    <TextView
        android:id="@+id/subtemp"
        android:layout_width="348dp"
        android:layout_height="24dp"
        android:layout_marginStart="10dp"
        android:layout_marginTop="26dp"
        android:textAlignment="viewStart"
        android:textColor="@android:color/holo_red_dark"
        android:textSize="18sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView" />
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="277dp"
        android:layout_height="222dp"
        android:layout_marginStart="52dp"
        android:layout_marginTop="16dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/linearLayout"
        app:srcCompat="@drawable/tempyhume" />
</android.support.constraint.ConstraintLayout>
```

A.8.5 Código: luminosidad.java

```
package com.proto.domqttica.domqtticaver1;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
```

```
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

public class luminosidad extends AppCompatActivity {
    Button volver2;
    Button salir2;
    static String MQTTHOST = "tcp://m14.cloudmqtt.com:19165";
    static String USERNAME = "cjkjjjze";
    static String PASSWORD = "WFJKWObn6WqQ";
    String topicStrluzpub = "sensor/luz";
    String topicStrluzsub = "sensor/luz";
    MqttAndroidClient client;
    TextView subluz;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_luminosidad);
        subluz = (TextView)findViewById(R.id.subluz);
        volver2 = (Button)findViewById(R.id.btnvolver2);
        salir2 = (Button)findViewById(R.id.btnsalir2);
        volver2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                onBackPressed();
            }
        });

        salir2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });

        String clientId = MqttClient.generateClientId();
        client = new MqttAndroidClient(this.getApplicationContext(),MQTTHOST, clientId);
        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(USERNAME);
        options.setPassword(PASSWORD.toCharArray());

        try {
            IMqttToken token = client.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Toast.makeText(luminosidad.this, "Conexión Global Establecida!",
Toast.LENGTH_LONG).show();
                    setSubscription();
                }
                @Override
                public void onFailure(IMqttToken asyncActionToken, Throwable exception)
{
                    Toast.makeText(luminosidad.this, "Conexión Global Fallida!",
Toast.LENGTH_LONG).show();
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
        client.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {
```

```
            }
            @Override
            public void messageArrived(String topic, MqttMessage message) throws
Exception {
                subluz.setText(new String(message.getPayload()));
            }
            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
            }
        });
    }
    public void pub(View v){
        String topic = topicStrluzpub;
        String message = "Luz";
        byte[] encodedPayload = new byte[0];
        try {
            client.publish(topic, message.getBytes(), 0, false);
        } catch (MqttException e) {
            e.printStackTrace();
        }}
    private void setSubscription(){
        try {
            client.subscribe(topicStrluzsub, 0);
        }catch (MqttException e){
            e.printStackTrace();
        }
    }
}
```

A.8.6 Código: activity_luminosidad.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.proto.domqttica.domqtticaver1.luminosidad">
    <LinearLayout
        android:id="@+id/linearLayout3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <Button
            android:id="@+id/btnvolver2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Volver a Menu" />
        <Button
            android:id="@+id/btnsalir2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Salir de la App" />
    </LinearLayout>
    <TextView
        android:id="@+id/subluz"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="120dp"
        android:layout_marginTop="37dp"
        android:text="Intensidad de Luz"
        android:textAlignment="center"
        android:textColor="@android:color/holo_blue_dark"
        android:textSize="18sp"
        android:textStyle="bold"
```

```
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView2" />
    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="233dp"
        android:layout_height="204dp"
        android:layout_marginStart="76dp"
        android:layout_marginTop="27dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/linearLayout3"
        app:srcCompat="@drawable/intenluz" />
</android.support.constraint.ConstraintLayout>
```

A.8.7 Código: seguridad.java

```
package com.proto.domqttica.domqtticaver1;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

public class seguridad extends AppCompatActivity {

    Button volver3;
    Button salir3;
    static String MQTTHOST = "tcp://m14.cloudmqtt.com:19165";
    static String USERNAME = "cjkjjjze";
    static String PASSWORD = "WFJKWObn6WqQ";
    String topicStrsub = "sensor/seguridad";
    MqttAndroidClient client;
    TextView subseguridad;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_seguridad);
        subseguridad = (TextView)findViewById(R.id.subseguridad);
        volver3 = (Button)findViewById(R.id.btnvolver3);
        salir3 = (Button)findViewById(R.id.btnsalir3);
        volver3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                onBackPressed();
            }
        });
        salir3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });
        String clientId = MqttClient.generateClientId();
```

```
        client = new MqttAndroidClient(this.getApplicationContext(),MQTTHOST, clientId);
        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(USERNAME);
        options.setPassword(PASSWORD.toCharArray());
        try {
            IMqttToken token = client.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Toast.makeText(seguridad.this, "Conexión Global Establecida!",
Toast.LENGTH_LONG).show();
                    setSubscription();
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken, Throwable exception)
{
                    Toast.makeText(seguridad.this, "Conexión Global Fallida!",
Toast.LENGTH_LONG).show();
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
        client.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {
            }

            @Override
            public void messageArrived(String topic, MqttMessage message) throws
Exception {
                subseguridad.setText(new String(message.getPayload()));
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
            }
        });
    }
    private void setSubscription(){
        try {
            client.subscribe(topicStrsub, 0);
        }catch (MqttException e){
            e.printStackTrace();
        }
    }
}
```

A.8.8 Código: activity_seguridad.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.proto.domqttica.domqtticaver1.seguridad">
    <LinearLayout
        android:id="@+id/linearLayout5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <Button
            android:id="@+id/btnvolver3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
```

```
            android:text="Volver a Menu" />
        <Button
            android:id="@+id/btnsalir3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Salir de la App" />
    </LinearLayout>
    <TextView
        android:id="@+id/subseguridad"
        android:layout_width="355dp"
        android:layout_height="27dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="68dp"
        android:text="Alarmas"
        android:textAlignment="center"
        android:textColor="@android:color/holo_green_dark"
        android:textSize="18sp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView3" />
    <ImageView
        android:id="@+id/imageView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="102dp"
        android:layout_marginTop="54dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/linearLayout5"
        app:srcCompat="@drawable/movientoico" />
</android.support.constraint.ConstraintLayout>
```

A.8.9 Código: calidad.java

```
package com.proto.domqttica.domqtticaver1;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

public class calidad extends AppCompatActivity {

    Button volver4;
    Button salir4;
    static String MQTTHOST = "tcp://m14.cloudmqtt.com:19165";
    static String USERNAME = "cjkjjjze";
    static String PASSWORD = "WFJKWObn6WqQ";
    String topicStrcalidadsub = "sensor/calidad";
    MqttAndroidClient client;
    TextView subcalidad;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_calidad);
```

```
        subcalidad = (TextView)findViewById(R.id.subcalidad);
        volver4 = (Button)findViewById(R.id.btnvolver4);
        salir4 = (Button)findViewById(R.id.btnsalir4);
        volver4.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                onBackPressed();
            }
        });
        salir4.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });
        String clientId = MqttClient.generateClientId();
        client = new MqttAndroidClient(this.getApplicationContext(),MQTTHOST, clientId);
        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(USERNAME);
        options.setPassword(PASSWORD.toCharArray());
        try {
            IMqttToken token = client.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Toast.makeText(calidad.this, "Conexión Global Establecida!",
Toast.LENGTH_LONG).show();
                    setSubscription();
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken, Throwable exception)
{
                    Toast.makeText(calidad.this, "Conexión Global Fallida!",
Toast.LENGTH_LONG).show();
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
        client.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {
            }

            @Override
            public void messageArrived(String topic, MqttMessage message) throws
Exception {
                subcalidad.setText(new String(message.getPayload()));
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
            }
        });
    }
    private void setSubscription(){
        try {
            client.subscribe(topicStrcalidadsub, 0);
        }catch (MqttException e){
            e.printStackTrace();
        }
    }
}
```

A.8.10 Código: activity_calidad.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.proto.domqttica.domqtticaver1.calidad">
    <LinearLayout
        android:id="@+id/linearLayout4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <Button
            android:id="@+id/btnvolver4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Volver a Menu" />
        <Button
            android:id="@+id/btnsalir4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Salir de la App" />
    </LinearLayout>
    <TextView
        android:id="@+id/subcalidad"
        android:layout_width="370dp"
        android:layout_height="30dp"
        android:layout_marginStart="2dp"
        android:layout_marginTop="32dp"
        android:text="Alarmas"
        android:textAlignment="center"
        android:textSize="18sp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView4" />
    <ImageView
        android:id="@+id/imageView4"
        android:layout_width="175dp"
        android:layout_height="265dp"
        android:layout_marginStart="105dp"
        android:layout_marginTop="25dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/linearLayout4"
        app:srcCompat="@drawable/calidadairer" />
</android.support.constraint.ConstraintLayout>
```

A.8.11 Código: config.java (conexión local)

```
package com.proto.domqttica.domqtticaver1;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
```

```
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

public class config extends AppCompatActivity {

    Button volver8;
    Button salir8;
    EditText localhostip;
    Button btnguardar, btnmostrar;
    static String host = "192.168.0.23"; // Cambiar Valor Por tu IP Local de la
Raspberry Pi
    static String MQTTHOST = "tcp://" + host + ":1883";
    static String USERNAME = "domqttica";
    static String PASSWORD = "eie655";
    String topicStrtempsublocal = "sensor/temperatura";
    String topicStrluzsublocal = "sensor/luz";
    String topicStrsublocal = "sensor/seguridad";
    String topicStrcalidadsublocal = "sensor/calidad";
    MqttAndroidClient clientlocal;
    TextView subtemperaturalocal;
    TextView subluzlocal;
    TextView subseguridadlocal;
    TextView subcalidadlocal;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_config);
        final   Context context = this;
        final   SharedPreferences sharprefs = getSharedPreferences("ArchivoSP",
context.MODE_PRIVATE);
        localhostip = (EditText)findViewById(R.id.localhostip);
        btnguardar = (Button)findViewById(R.id.btnguardar);
        btnmostrar = (Button)findViewById(R.id.btnmostrar);
        btnmostrar.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                SharedPreferences sharpref = getPreferences(context.MODE_PRIVATE);
                String valor = sharpref.getString("MiIPlocal","No hay dato");
                Toast.makeText(getApplicationContext(), "IP guardada :
"+valor,Toast.LENGTH_LONG).show();
            }
        });
        btnguardar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences sharpref = getPreferences(context.MODE_PRIVATE);
                SharedPreferences.Editor editor = sharpref.edit();
                editor.putString("MiIPlocal", localhostip.getText().toString());
                editor.commit();
            }
        });
        subtemperaturalocal = (TextView)findViewById(R.id.subtemperalocal);
        subluzlocal = (TextView)findViewById(R.id.subluzlocal);
        subseguridadlocal = (TextView)findViewById(R.id.subseguridadlocal);
        subcalidadlocal = (TextView)findViewById(R.id.subcalidadlocal);
        volver8 = (Button)findViewById(R.id.btnvolver8);
        salir8 = (Button)findViewById(R.id.btnsalir8);
        volver8.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                onBackPressed();
            }
        });
        salir8.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View view) {
                Intent intent = new Intent(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });
        String clientId = MqttClient.generateClientId();
        clientlocal = new MqttAndroidClient(this.getApplicationContext(),MQTTHOST,
clientId);
        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(USERNAME);
        options.setPassword(PASSWORD.toCharArray());
        try {
            IMqttToken token = clientlocal.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Toast.makeText(config.this, "Conexión Local Establecida!",
Toast.LENGTH_LONG).show();
                    setSubscription();
                }
                @Override
                public void onFailure(IMqttToken asyncActionToken, Throwable exception)
{
                    Toast.makeText(config.this, "Conexión Local Fallida!",
Toast.LENGTH_LONG).show();
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
        clientlocal.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {
            }

            @Override
            public void messageArrived(String topic, MqttMessage message) throws
Exception {
                if (topic.equals(topicStrtempsublocal) ) {
                    subtemperaturalocal.setText(new String(message.getPayload()));
                }
                if (topic.equals(topicStrluzsublocal))
                {
                    subluzlocal.setText(new String(message.getPayload()));
                }
                if (topic.equals(topicStrsublocal))
                {
                    subseguridadlocal.setText(new String(message.getPayload()));
                }
                else if (topic.equals(topicStrcalidadsublocal))
                {
                    subcalidadlocal.setText(new String(message.getPayload()));
                }
            }
            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
            }
        });
    }
    private void setSubscription(){
        try {
            clientlocal.subscribe(topicStrtempsublocal, 0);
            clientlocal.subscribe(topicStrluzsublocal, 0);
            clientlocal.subscribe(topicStrsublocal, 0);
            clientlocal.subscribe(topicStrcalidadsublocal, 0);
        }catch (MqttException e){
            e.printStackTrace();
        }
    }
```

```
}
```

A.8.12 Código: activity_config.xml (conexión local)

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.proto.domqttica.domqtticaver1.config">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/btnvolver8"
            android:text="Volver a Menu"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/btnsalir8"
            android:text="Salir de la App"
            android:layout_weight="1"/>
    </LinearLayout>
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:layout_editor_absoluteX="0dp"
        tools:layout_editor_absoluteY="39dp">
        <TextView
            android:id="@+id/textView0"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginTop="67dp"
            android:text="Conexión Local"
            android:textAlignment="center"
            android:textAllCaps="true"
            android:textColor="@android:color/holo_blue_dark"
            android:textSize="18sp"
            android:textStyle="bold" />
        <EditText
            android:id="@+id/localhostip"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_below="@+id/textView0"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="14dp"
            android:textAlignment="center"
            android:textColor="@android:color/darker_gray"
            tools:text="IP local de DoMQTTica" />
        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginTop="219dp"
            android:text="Temperatura y Humedad"
            android:textAlignment="center"
```

```
            android:textAllCaps="true"
            android:textColor="@android:color/background_dark"
            android:textSize="18sp"
            android:textStyle="bold" />
        <TextView
            android:id="@+id/subtemperalocal"
            android:layout_width="370dp"
            android:layout_height="30dp"
            android:layout_below="@+id/textView"
            android:layout_centerHorizontal="true"
            android:text="temperatura actual"
            android:textAlignment="center"
            android:textSize="18sp"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="parent" />
        <TextView
            android:id="@+id/textView2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignLeft="@+id/subluzlocal"
            android:layout_alignStart="@+id/subluzlocal"
            android:layout_below="@+id/subtemperalocal"
            android:layout_marginTop="21dp"
            android:text="Intencidad de Luz"
            android:textAlignment="center"
            android:textAllCaps="true"
            android:textColor="@android:color/background_dark"
            android:textSize="18sp"
            android:textStyle="bold" />
        <TextView
            android:id="@+id/subluzlocal"
            android:layout_width="370dp"
            android:layout_height="30dp"
            android:layout_alignLeft="@+id/subtemperalocal"
            android:layout_alignStart="@+id/subtemperalocal"
            android:layout_below="@+id/textView2"
            android:text="intencidad actual"
            android:textAlignment="center"
            android:textSize="18sp"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="parent" />
        <TextView
            android:id="@+id/textView3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_below="@+id/subluzlocal"
            android:layout_marginTop="25dp"
            android:text="Sensor de Movimientos"
            android:textAlignment="center"
            android:textAllCaps="true"
            android:textColor="@android:color/background_dark"
            android:textSize="18sp"
            android:textStyle="bold" />
        <TextView
            android:id="@+id/subseguridadlocal"
            android:layout_width="370dp"
            android:layout_height="30dp"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_below="@+id/textView3"
            android:text="movimientos dectados"
            android:textAlignment="center"
            android:textSize="18sp"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="parent" />
        <TextView
            android:id="@+id/textView4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
            android:layout_alignLeft="@+id/subluzlocal"
            android:layout_alignStart="@+id/subluzlocal"
            android:layout_below="@+id/subseguridadlocal"
            android:layout_marginTop="15dp"
            android:text="Calidad del Aire"
            android:textAlignment="center"
            android:textAllCaps="true"
            android:textColor="@android:color/background_dark"
            android:textSize="18sp"
            android:textStyle="bold" />
        <TextView
            android:id="@+id/subcalidadlocal"
            android:layout_width="370dp"
            android:layout_height="30dp"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_below="@+id/textView4"
            android:text="calidad del aire actual"
            android:textAlignment="center"
            android:textSize="18sp"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="parent" />
        <Button
            android:id="@+id/btnguardar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/localhostip"
            android:layout_marginEnd="12dp"
            android:layout_marginRight="12dp"
            android:layout_toLeftOf="@+id/btnmostrar"
            android:layout_toStartOf="@+id/btnmostrar"
            android:text="Guardar IP" />
        <Button
            android:id="@+id/btnmostrar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignBaseline="@+id/btnguardar"
            android:layout_alignBottom="@+id/btnguardar"
            android:layout_alignEnd="@+id/subseguridadlocal"
            android:layout_alignRight="@+id/subseguridadlocal"
            android:layout_marginEnd="68dp"
            android:layout_marginRight="68dp"
            android:text="Mostrar IP" />
    </RelativeLayout>
</android.support.constraint.ConstraintLayout>
```

A.8.13 Código: help.java

```
package com.proto.domqttica.domqtticaver1;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class help extends AppCompatActivity {

    Button volver9;
    Button salir9;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_help);
        volver9 = (Button)findViewById(R.id.btnvolver9);
        salir9 = (Button)findViewById(R.id.btnsalir9);
        volver9.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                onBackPressed();
            }
        });
        salir9.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });
    }
}
```

A.8.14 Código: activity_help.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.proto.domqttica.domqtticaver1.help">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/btnvolver9"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Volver a Menu" />
        <Button
            android:id="@+id/btnsalir9"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Salir de la App" />
    </LinearLayout>
    <RelativeLayout
        android:layout_width="368dp"
        android:layout_height="551dp"
        tools:layout_editor_absoluteX="8dp"
        tools:layout_editor_absoluteY="8dp">
        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginTop="75dp"
            android:text="Ayuda"
            android:textAlignment="center"
            android:textAllCaps="true"
            android:textColor="@android:color/background_dark"
            android:textSize="18sp"
            android:textStyle="bold" />
        <ImageView
            android:id="@+id/imageView5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentEnd="true"
            android:layout_alignParentRight="true"
```

```
            android:layout_below="@+id/textView"
            android:layout_marginTop="14dp"
            app:srcCompat="@drawable/infoayuda" />
    </RelativeLayout>
</android.support.constraint.ConstraintLayout>
```

A.8.15 Código: SplashActivity.java

```
package com.proto.domqttica.domqtticaver1;

import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.WindowManager;

public class SplashActivity extends AppCompatActivity {
    private final int DURACION_SPLASH = 1500;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.activity_splash);
        new Handler().postDelayed(new Runnable(){
            public void run(){
                Intent intent = new Intent(SplashActivity.this, menu.class);
                startActivity(intent);
                finish();
            };
        }, DURACION_SPLASH);
    }
}
```

A.8.16 Código: activity_splash.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FFFFFF">
    <ImageView
        android:paddingLeft="5dp"
        android:paddingRight="5dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/imagensplash"
        android:id="@+id/ivSplash"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"/>
</RelativeLayout>
```