# ESCUELA POLITÉCNICA NACIONAL

## FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

## DESARROLLO DE UN PROTOTIPO PARA LA AUTOMATIZACIÓN DE UN SISTEMA DE RIEGO DE AGUA Y CONTROL REMOTO MEDIANTE LA PLATAFORMA ZOLERTIA REMOTE

**TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN**

**SIAMARA ADRIANA ÑAMO MARTÍNEZ**

**DIRECTOR: MSc. PABLO WILIAN HIDALGO LASCANO**

**Quito, julio 2019**

# AVAL

Certifico que el presente trabajo fue desarrollado por Siamara Adriana Ñamo Martínez, bajo mi supervisión.

**MSc. Pablo Wilian Hidalgo Lascano**
**DIRECTOR DEL TRABAJO DE TITULACIÓN**

# DECLARACIÓN DE AUTORÍA

Yo, Siamara Adriana Ñamo Martínez, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

Siamara Adriana Ñamo Martínez

# DEDICATORIA

Kay llankaytaka ñukapak aylluman, mashikunaman, ashtawanka ñukapak mamakumanmi kuni, paymi ñuka kawsaypika tukuymanta yalli may mutsurik kashka, shinallatak ñukapak muskuykuna paktachunpash mana shaykushpami yanapashka, paypak kawsayta, llankayta kuyaytapash rikushpami llakikunawan kashpapash kay yachaykunata tukuchina pachaman chayamushkani.

Dedico este trabajo a mi familia y amigos, en especial a mi madre, quien es la persona más especial en mi vida, porque siempre me ha apoyado en el cumplimiento de mis sueños y por ser el ejemplo de fuerza, valentía y amor que he necesitado para luchar a pesar de las dificultades.

Siamara Adriana Ñamo Martínez

# AGRADECIMIENTO

Kallaripika Apunchikta yupaychani kay llankay ruraypi allí ñanta pushashpa, paypak yachaytapash kushpa ñuka mayman chayana kashkaman chayankakama yanapakushkamanta. Shinallatak ñukapak mamakupash, llaki pachapi, kushi pachapipash kuyaywan yanapashkamanta.

Shinallatak kay llaykayta yanapak MSc. Pablo Hidalgo mashitapash yupaychani, paypak sumak yachaykunawan yuyaykunawanpash kay yachaykuna taripaypi yanapashkamanta. Shinallatak yupaychani MSc. Carlos Hernández mashita, paypak yachaywan llankaywanpash kay llankaypak paktaykunata hapinapak yanapashkamanta.

Tukuripika tukuy yachachikkunata, ñukapak mashikunata, shuktakkunatapash yupaychani, paykunapak sumak rimashkakunawan kay Escuela Politécnica Nacional sumak yachana wasipi alli yachak warmi kachun yanapashkamanta.

Agradezco a Dios por ser mi guía y compañero en el transcurso de mi vida, brindándome paciencia y sabiduría para culminar con éxito mis metas propuestas. A mi madre por ser mi pilar fundamental y haberme apoyado incondicionalmente, a pesar de las adversidades que se han presentado.

Agradezco a mi director del Trabajo de Titulación, MSc. Pablo Hidalgo, quien, con su experiencia, conocimiento y motivación me ha orientado hasta la culminación de este proyecto. También agradezco al MSc. Carlos Hernández, quien con su apoyo en conocimiento y experiencia me ha permitido lograr los objetivos en este proyecto.

Finalmente, agradezco a todos los docentes y amigos por su apoyo durante mis estudios en la Escuela Politécnica Nacional y por motivarme a siempre trabajar para ser una mujer de bien.

Siamara Adriana Ñamo Martínez

# ÍNDICE DE CONTENIDO

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# ÍNDICE DE CÓDIGOS

# RESUMEN

El Internet de las Cosas (IoT) es una tendencia actual que posibilita que los objetos del mundo real se conecten a Internet, para interactuar entre ellos y con las personas. Existen diversas aplicaciones de esta tecnología que permiten mejorar los procesos en distintos campos, en las empresas, industrias, ciudades, hogares, etc. Su crecimiento va en aumento gracias a los grandes beneficios que brinda esta tecnología a la sociedad entera.

Una de las aplicaciones que se puede generar con el Internet de las Cosas es la optimización de procesos en el sector agrícola, ayudando al agricultor a tener eficiencia en la ejecución de estos procesos. Es por eso que, el presente proyecto se orienta al desarrollo de un sistema de riego automatizado y control remoto usando el enfoque del Internet de las Cosas.

El presente trabajo consta de cuatro capítulos, de los cuales, el primero describe varios conceptos y protocolos que se usan para el desarrollo de aplicaciones IoT; también se presentan las características de los dispositivos que se utilizan para la implementación del proyecto. Además, se detallan los programas que se usan y los fundamentos de riego en la agricultura, así como una breve presentación del área de cultivo en el que se desarrolla la implementación. El segundo capítulo presenta el diseño de software del proyecto que consta de la aplicación móvil y la comunicación entre la red de nodos y la aplicación móvil. El tercer capítulo describe la implementación tanto del hardware como del software que forman parte del prototipo, y las pruebas de funcionamiento que son parte de un análisis posterior. El último capítulo presenta las conclusiones y recomendaciones resultantes del desarrollo completo del proyecto. Finalmente, en la parte de Anexos se presenta el código y el modelo de entrevista desarrollado para el proyecto.

**PALABRAS CLAVE:** IoT, IEEE 802.15.4, MQTT, Paho, Zolertia RE-Mote, *broker*.

# ABSTRACT

The Internet of Things is the current trend that allows real-world objects to connect to the Internet, to interact with each other and with people. They are several applications of this technology that allow to improve the processes in different fields, in companies, industries, cities, homes, etc. Its growth is increasing thanks to the great benefits that it offers to the whole society.

One of the applications that can be generated with the Internet of Things is the optimization of processes in the agricultural sector, helping to the farmer to have efficiency in the execution of this processes. For that reason, the present project is oriented to the development of an automated irrigation system and remote control using the Internet of Things approach.

The present work consists of four chapters, of which, the first one describes several concepts and protocols that are used for the development of IoT applications; besides the characteristics of the devices that are used for the implementation of the project. In addition, the programs that are used and the basics of irrigation in agriculture are detailed, as well as a brief presentation of the cultivation area in which the implementation is carried out. The second chapter presents the software design of the project that consisting of the mobile application and the communication between the network of nodes and the mobile application. The third chapter consists of the implementation of both the hardware and software that are part of the prototype, and the performance test that are part of a later analysis. The last chapter presents the conclusions and recommendations resulting from the complete development of the project. Finally, in the Annexes part present the code and the model interview development for the project.

**KEYWORDS:** IoT, IEEE 802.15.4, MQTT, Paho, Zolertia RE-Mote, *broker.*

# 1. INTRODUCCIÓN

En el Ecuador existe una baja innovación tecnológica aplicada a los procesos del sector agrícola, por tal razón el MAGAP (Ministerio de Agricultura, Ganadería, Acuacultura y Pesca) a través de la Política Agropecuaria Ecuatoriana [1] enfocada al desarrollo territorial rural sostenible 2015-2025, promueve la implementación de la innovación tecnológica en este sector.

El proceso de riego en la agricultura en la provincia de Chimborazo, de forma general, se hace manualmente, lo cual conlleva a la generación de algunos problemas, como la dependencia física del agricultor durante este proceso, la deficiencia en el aprovechamiento del agua de riego, entre otros.

Por estas razones es que, el presente Trabajo de Titulación pretende desarrollar un prototipo de automatización de un sistema de riego y control remoto mediante la plataforma Zolertia RE-Mote. El proyecto se basa en el Internet de las Cosas y utiliza el protocolo MQTT (*Message Queue Telemetry Transport*) de capa aplicación para su ejecución.

La implementación de este proyecto se realiza en un área de cultivo de la comunidad Gatazo Chico, parroquia Cajabamba, cantón Colta, provincia de Chimborazo. Su ejecución permitirá el mejoramiento en la eficiencia del proceso de riego de agua en los cultivos, así como también la eficiencia de tiempo del agricultor y por último se promoverá la implementación de innovación tecnológica en los distintos procesos del sector agrícola.

## 1.1. OBJETIVOS

El objetivo general de este Proyecto Técnico es:

- Desarrollar un prototipo para la automatización de un sistema de riego de agua y control remoto mediante la plataforma Zolertia RE-Mote.

Los objetivos específicos de este Proyecto Técnico son:

- Analizar la teoría concerniente al Internet de las Cosas, a la plataforma Zolertia RE-Mote y a los fundamentos del riego en la agricultura.
- Diseñar los tres bloques del prototipo
- Implementar los bloques del prototipo
- Analizar los resultados de la funcionalidad del prototipo en base a pruebas.

## 1.2. ALCANCE

El proyecto se enfoca en la implementación de una aplicación del Internet de las Cosas (IoT) desarrollada para el sector agrícola, específicamente para el proceso de riego de agua en los cultivos, para lo cual se recogen datos de temperatura y humedad del suelo mediante sensores, datos que son enviados al usuario final mediante una aplicación móvil, para que el usuario tenga la posibilidad de accionar una bomba de regadío de forma remota con la información recibida.

El alcance del proyecto se define a través de un conjunto de tres bloques, los mismos que se presentan de forma gráfica en la Figura 1.1. El primer bloque abarca la red de nodos sensores y el nodo actuador, el segundo bloque comprende el *router* de borde y el tercero consta de la aplicación móvil. Todo el desarrollo de estos bloques partirá de la ejecución de la metodología ágil Kanban, para un cumplimiento eficaz de los objetivos del proyecto.

En el primer bloque, se despliegan las motas en la superficie que comprende el cultivo, de forma que abarquen la totalidad de la misma; cada una de las motas tiene conectado un sensor de humedad y temperatura del suelo. En el pozo de agua se colocan sensores de nivel de agua, éstos a su vez se conectan a una mota directamente, la misma mota se conecta a la bomba de regadío mediante un dispositivo relé, esta mota se denomina nodo actuador; las conexiones en el nodo actuador permitirán accionar la bomba de agua conociendo el nivel de agua en el pozo, mediante los valores que miden los sensores. La comunicación entre las motas se logra mediante la implementación del protocolo MQTT, donde cada una de las motas actúa como un cliente MQTT que envía y recibe información.

El segundo bloque consta de la implementación del *router* de borde, para lo cual una mota se conecta de forma serial a una Raspberry Pi, formando una interfaz virtual que permite la comunicación de una red de nodos IPv6 a una red IPv4, con el fin de salir posteriormente hacia la red externa mediante una conexión inalámbrica a un módem. La red de nodos a través del *router* de borde se comunica mediante el protocolo MQTT con un *broker* privado Mosquitto instalado en la Raspberry Pi. En la Raspberry Pi se ejecuta un cliente MQTT para recibir la información de la red de nodos y reenviarla hacia el *broker* público gracias a la conexión inalámbrica; también se ejecuta otro cliente MQTT con la finalidad de realizar la comunicación en el otro sentido, es decir, desde la red externa hacia el nodo actuador. La ejecución de ambos clientes MQTT se realiza en dos *script*s que usarán las herramientas Paho de Eclipse, proyecto desarrollado para facilitar la creación de clientes MQTT, proveyendo código fuente en distintos lenguajes de programación.

El tercer bloque se basa en el diseño e implementación de una aplicación móvil en el IDE Android Studio tomando como base el servicio Android Paho, que provee una fuente de código para la creación de clientes MQTT. En la aplicación se crea un cliente MQTT, el mismo que por medio de su conexión con un *broker* público Mosquitto recibe la información de la red de nodos y también envía información hacia el nodo actuador. Esta aplicación móvil permite al usuario final varias opciones de interacción, conectarse con la red de nodos, recibir información en tiempo real del mismo, solicitar información de nivel de agua y comandar el encendido o apagado de la bomba de agua de forma remota; todas estas opciones están disponibles cuando el teléfono móvil tiene acceso a Internet.

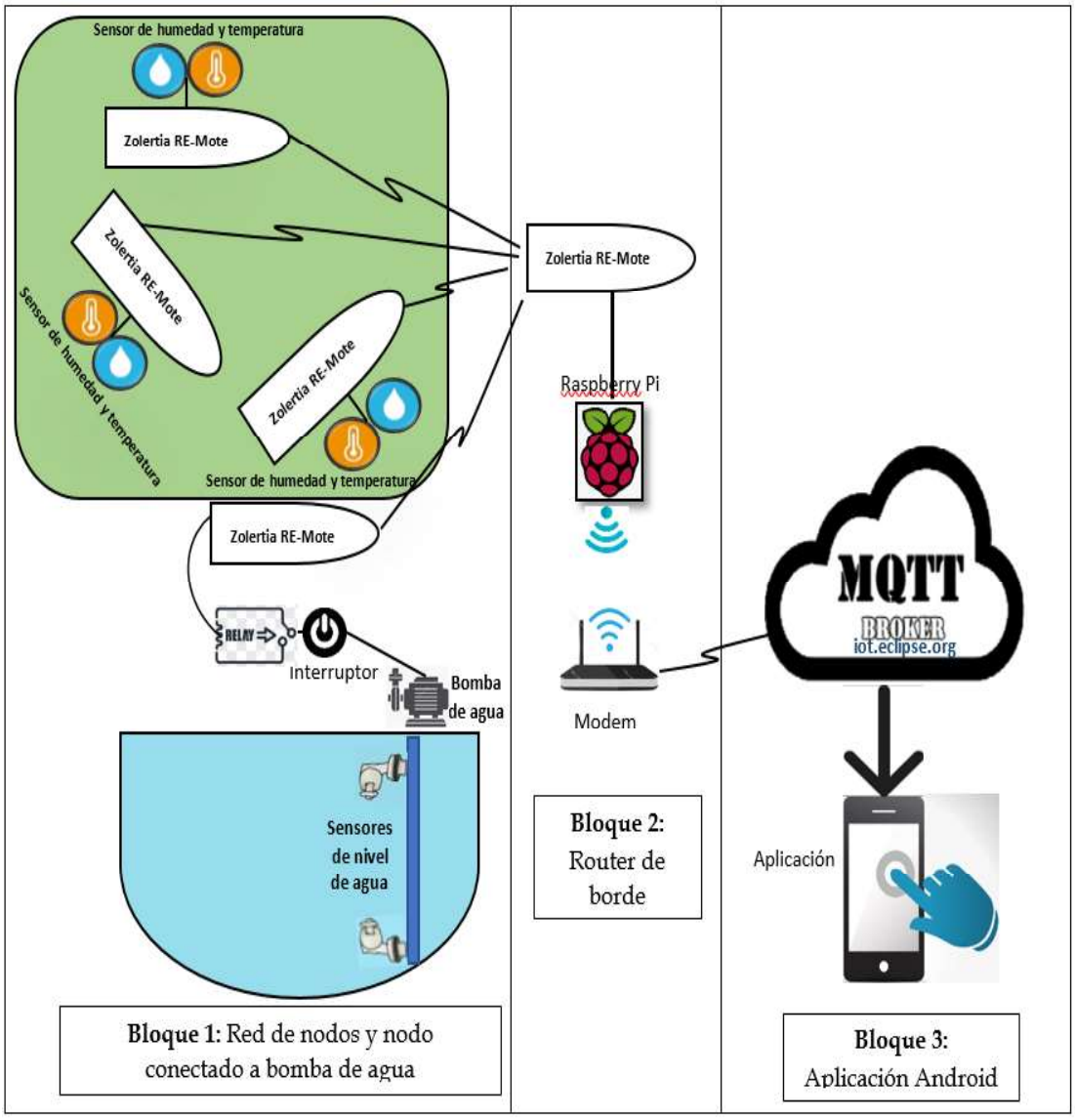



**Figura 1.1.** Bloques del prototipo

## 1.3. MARCO TEÓRICO

### 1.3.1. INTERNET DE LAS COSAS

A través de los años, el Internet ha venido tomando lugar en varios ámbitos como en la multimedia, la industria, el comercio, etc.; de la misma manera el Internet de las Cosas ha empezado la transformación de los procesos en varios sectores de la industria y sociedad. El IoT en esencia, es la ejecución de operaciones inteligentes a través de objetos del mundo real; dichas operaciones están basadas en software que permite tomar las propiedades físicas de las cosas para automatizar procesos que dan a luz distintos resultados en el campo aplicado [2].

El Foro Económico mundial ha tratado el tema de IoT en variadas ocasiones e indica que en el año 2011 existían cerca de 7 billones de dispositivos conectados a Internet y se espera alcanzar a los 50 billones de dispositivos para el año 2020, como se muestra en la Figura 1.2. Además se dice que probablemente IoT se convierta en la próxima mejor oportunidad de valor en la industria, ya que se pronostica que el IoT industrial permitirá aumentar 14 trillones de dólares a la economía global en el 2030, lo que traerá consigo nuevas oportunidades de negocio así como lo hace Internet [3].

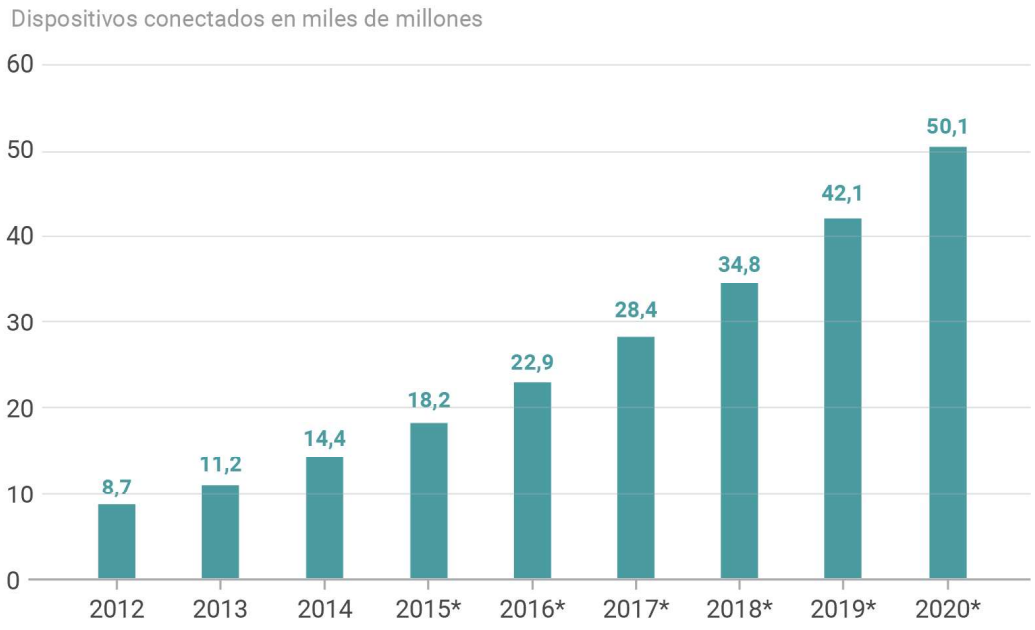



**Figura 1.2.** Predicción de conexión de dispositivos a Internet para el 2020 [4]

El IoT es aplicable en varios campos como en el comercio, las oficinas, los hogares, las ciudades, los vehículos, las fábricas, e incluso la humanidad, éste también llega al sector de la agricultura, donde varios procesos pueden ser controlados y automatizados por esta tecnología.

El IoT se relaciona con diferentes tecnologías existentes, funciona en base a una red inalámbrica de sensores (WSN), para interconectar nodos de sensores y recopilar información del entorno; a fin de posibilitar acciones en el entorno se usan sistemas de control que entrelazan el software con los objetos del mundo real. De forma general IoT centraliza, analiza y maneja los datos sobre los objetos y los procesos relacionados, lo cual se conoce como *Big Data*; para dar paso a la ejecución de operaciones inteligentes sobre las cosas, el IoT se relaciona con la Inteligencia Artificial.

El Internet de las Cosas ha sido el resultado de varios sucesos tecnológicos, como el gran crecimiento del Internet, el amplio uso del protocolo IP (*Internet Protocol*), la disminución en tamaño y costo de microcontroladores, actuadores y sensores, además de la vasta adopción de tecnologías inalámbricas como Bluetooth, WiFi y otros en la cotidianidad.

Existen múltiples razones para el futuro aumento del uso de IoT, tales como la necesidad de conocer y apreciar propiedades del entorno físico con miras a mejorar la eficiencia, la salud, la seguridad, etc. También se encuentran el bajo costo y fácil acceso a tecnologías que permiten recopilar y analizar datos, así como, el bajo costo de los dispositivos IoT que se caracterizan por tener capacidades suficientes para almacenamiento de datos, el bajo consumo de potencia, tamaños relativamente pequeños y adaptables en diversos entornos. El progreso actual y futuro del Internet de las Cosas se debe también a que el uso de tecnologías de Internet es abierto y algunas de ellas ya están estandarizadas.

En fin, el alcance del Internet de las Cosas solo está limitado por la imaginación del hombre, pues tiene todo el material necesario en su entorno para el desarrollo de diversas aplicaciones IoT en cualquier campo.

Uno de los campos en los cuales se puede aplicar el IoT, es la agricultura, donde mediante el uso de tecnologías IoT, se pueden optimizar varios procesos de este sector de la economía. Por ejemplo, para monitorizar y controlar el entorno de las plantaciones se pueden utilizar sensores y actuadores, esto con el fin de adaptar las condiciones de acuerdo a las necesidades de los agricultores.

Las soluciones IoT han empezado a ser muy comunes en diferentes escenarios, y se han vuelto agentes de cambio en la industria tecnológica, en la economía y en la sociedad en general. Además, varios de los problemas ambientales, económicos y sociales están siendo enfrentados gracias a las nuevas soluciones IoT, y se están convirtiendo en una mega tendencia en todo el mundo, la misma que se estima se mantendrá por varias generaciones. Sin embargo, existen varios desafíos en la implementación de soluciones IoT, desafíos tales como la interoperabilidad tanto a nivel de componentes de hardware

como de software, así como la interferencia entre dispositivos que operan en la misma banda de frecuencia o en bandas vecinas en el caso de las comunicaciones inalámbricas.

La interoperabilidad como un desafío del Internet de las Cosas se puede ver desde varios puntos de vista. La interoperabilidad sintáctica que tiene que ver con los formatos de datos y sintaxis en los mensajes de manera que se puedan intercambiar entre diferentes sistemas; la interoperabilidad semántica que tiene un valor especial para los usuarios finales, debido a que tiene que ver con la interpretación humana y comprensión del contenido producido por las soluciones IoT. La interoperabilidad organizacional que permite brindar escalabilidad; la interoperabilidad técnica que hace referencia a la habilidad para comunicar de forma efectiva y transferir datos significativos entre los sistemas a través de zonas geográficas variables [5].

### 1.3.2. IoT EN LA AGRICULTURA

La gran demanda de alimentos de calidad y en cantidad crece significativamente en todo el mundo, lo que conlleva a la necesidad de la modernización del sector agrícola. En este punto el Internet de las Cosas provee varias soluciones para suplir dicha necesidad. Diversos grupos científicos e instituciones de investigación desarrollan y entregan productos IoT a las empresas agrícolas. También tecnologías conocidas apoyan el despliegue de IoT en ésta y otras áreas; tecnologías como *Cloud Computing*[1], *Fog Computing*[2], proveen recursos y soluciones para almacenar, mantener y analizar grandes cantidades de información resultantes de IoT, conocida como *Big Data*. El *Big Data* puede ser procesado para alcanzar logros como: la automatización de procesos, la predicción de situaciones y el mejoramiento de actividades en tiempo real [6].

La Organización de las Naciones Unidas para la Alimentación (FAO) predijo que la población global alcanzará los 8 billones de personas para el 2025 y 9.6 billones de personas para el 2050 [7], esto se aprecia en la Figura 1.3.; lo que significa que deberá lograrse un gran aumento en la producción de alimentos en el mundo entero en los próximos 30 años. En este contexto, uno de los conceptos más prometedores para alcanzar la producción requerida en los próximos años es la agricultura de precisión, término que tiene como fin mejorar y optimizar procesos agrícolas garantizando la máxima productividad, mediante mediciones rápidas, confiables y periódicas.

[1] ***Cloud Computing*****:** Término que hace referencia a la dotación de recursos informáticos a través de Internet, de acuerdo a la solicitud del usuario.
[2] ***Fog Computing*****:** Tecnología en la cual el procesamiento de datos se concentra en el borde de la red para aliviar el trabajo en el *Cloud*.

En un nivel más alto, con *Big Data* y el conocimiento científico se podrán desarrollar algoritmos inteligentes para proporcionar una mejor visión de los procesos actuales, mediante un análisis de la situación actual para realizar predicciones, lanzar advertencias tempranas de peligros potenciales que amenazan los cultivos y producir señales de control automatizadas basadas en respuestas de las plantas.

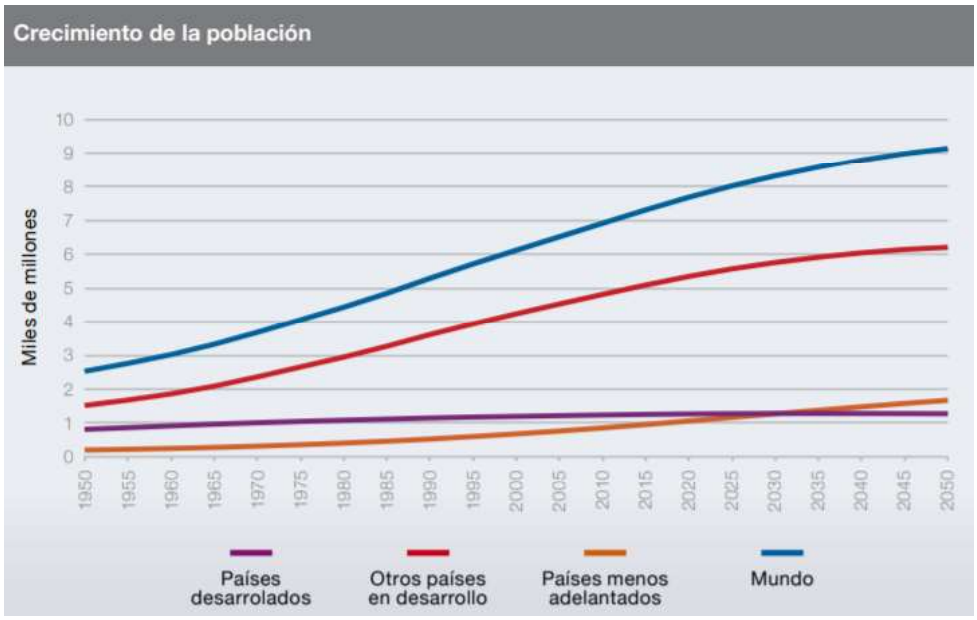



**Figura 1.3.** Predicción del crecimiento de la población mundial [7]

Las soluciones comerciales de sensado proporcionan una serie de funciones listas para usar, permitiendo a los investigadores ocuparse de otros aspectos de las implementaciones IoT, tales como el meta procesamiento de datos, el desarrollo de algoritmos inteligentes de monitoreo y control, la interoperabilidad en la nube, etc. [8]. Una gran ventaja para los desarrolladores de aplicaciones IoT, es la disponibilidad abierta de las soluciones programables, de modo que tienen flexibilidad para trabajar en el comportamiento de la red de nodos, sensores, actuadores y su interconectividad.

El IoT es aplicable en varios escenarios del sector agrícola, por ejemplo, en redes de sensores utilizados para detección y control de infraestructuras agrícolas, en redes de sensores multimedia que capturen y procesen imágenes para detección de plagas, en redes basadas en etiquetas para seguimiento del producto y control remoto. Todo esto ha sido impulsado gracias a la disponibilidad de dispositivos versátiles con altas capacidades computacionales, diseños más pequeños y bajos costos de adquisición; actualmente se pueden usar estos dispositivos con baterías para operar por largos períodos de tiempo. También los dispositivos integrados actuales poseen recursos suficientes para soportar

sensores más exigentes y protocolos de red tales como TCP/IP (*Transmission Control Protocol/Internet Protocol*).

Las aplicaciones IoT para este sector son exigentes, donde los sensores deben ser precisos para realizar mediciones apropiadas y además deben estar protegidos contra factores ambientales que pueden dar lugar a lecturas falsas o en el peor de los casos a la destrucción del sensor. Luego de una elección cuidadosa de los sensores, es importante desarrollar el código funcional basado en un profundo conocimiento de ingeniería de software, para luego pasar por pruebas suficientes con la finalidad de evitar fallas en la implementación.

### 1.3.3. *STACK* DE PROTOCOLOS DE IoT

La estandarización de la pila de protocolos del Internet de las Cosas aún no ha sido desarrollada, debido a las dificultades del entorno en que IoT se despliega. Aspectos tales como la baja disponibilidad de memoria, el bajo consumo de energía, el bajo requerimiento de ancho de banda y la alta pérdida de paquetes, no permiten que el modelo TCP/IP se pueda utilizar de manera fácil en el IoT. A pesar de todo esto, existen varios protocolos propietarios para IoT, los mismos que son compatibles con las alianzas de proveedores de productos correspondientes. Organismos como IEEE (*Institute of Electrical and Electronics Engineers*), IETF (*Internet Engineering Task Force*) y W3C (*World Wide Web Consortium*) han estandarizado ciertos protocolos, los cuales se muestran por cada capa del modelo TCP/IP en la Figura 1.4.

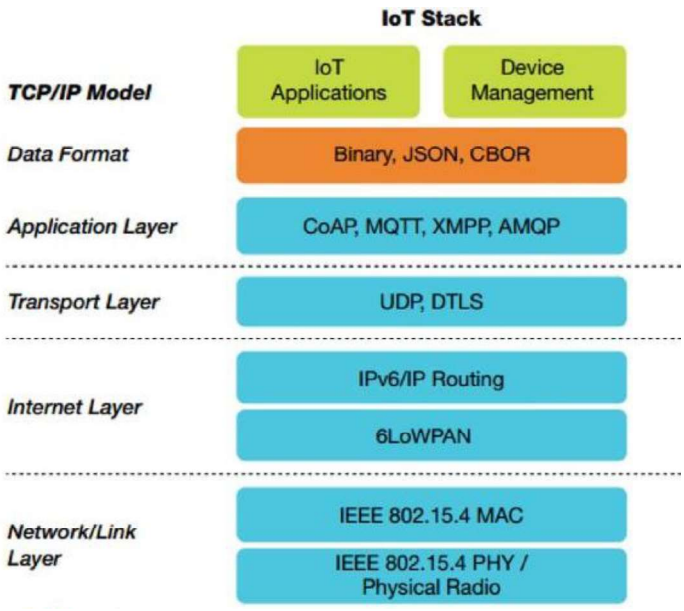



**Figura 1.4.** Stack de protocolos IoT [9]

En la capa de Enlace se tiene el estándar para comunicación inalámbrica de baja velocidad IEEE 802.15.4, donde se han definido el protocolo y la interconexión compatible para los dispositivos transmisores de datos que utilizan transmisiones de baja frecuencia de datos, baja potencia y baja complejidad de radiofrecuencia en una red de área personal inalámbrica (WPAN) [10]. Este estándar define dos subcapas, la capa Física (PHY), y la capa de Control de Acceso al Medio (MAC).

En IoT, las tecnologías inalámbricas dominantes principales pueden establecerse en seis categorías [11]:

- El Sistema Global para Comunicaciones Móviles (GSM) que ofrecen las operadoras autorizadas.
- Las Redes de Área Personal inalámbricas (WPAN).
- Las Redes de Área Regional inalámbricas (WRAN).
- Las redes *Mesh*.
- Las redes punto a punto (P2P).
- Las Redes de Área Extendida de baja potencia (LPN/LPWAN).

En la capa de Internet se tiene a 6LowPAN (*IPv6 over Low-power Wireless Personal Area Networks*), estandarizado por el IETF [12], cuyo acrónimo hace referencia al protocolo IPv6 sobre redes de área personal inalámbrica de baja potencia, y define una capa de adaptación que permite el transporte de paquetes IPv6 a través de enlaces basados en IEEE 802.15.4 de la capa Enlace.

En la capa de Transporte se tienen dos protocolos, UDP (*User Datagram Protocol*) y DTLS (*Datagram Transport Layer Security*). El protocolo UDP se utiliza en la mayoría de las soluciones IoT, debido a su ligereza, rapidez y tamaño de encabezado menor comparado con TCP, lo que hace que sea adecuado para el ambiente restringido en el que se encuentran los dispositivos y sensores. Mientras que DTLS, es un protocolo que provee seguridad en comunicaciones para protocolos que trabajan con datagramas como UDP.

En la capa Aplicación se tienen algunos protocolos tales como: CoAP (*Constrained Application Protocol*), MQTT (*Message Queue Telemetry Transport*), XMPP (*Extensible Messaging and Presence Protocol*) y AMQP (*Advanced Message Queuing Protocol*). CoAP es un protocolo de transferencia web diseñado para redes y nodos con restricciones en el Internet de las Cosas. MQTT es un protocolo de mensajería liviano basado en la suscripción y publicación para IoT. XMPP es una tecnología XML abierta para la comunicación en tiempo real que impulsa una amplia gama de aplicaciones como

mensajería instantánea, videoconferencia y colaboración. AMQP es un estándar abierto para el envío de mensajes entre aplicaciones u organizaciones, diseñado para brindar seguridad, confiabilidad e interoperabilidad.

Existen diferentes formatos de datos, entre los cuales están el binario, JSON (*JavaScript Object Notation*) y CBOR (*Concise Binary Object Representation*), donde JSON es un formato ligero de intercambio de datos, basado en lenguaje de programación Java*Script*; además usa convenciones familiares para C, C++, C#, Java, Perl, Python y otros más. CBOR es un formato de datos que incluye un tamaño pequeño de código y de mensaje.

### 1.3.4. *MESSAGE QUEUE TELEMETRY TRANSPORT*

MQTT es un protocolo de conectividad para el Internet de las Cosas, fue inventado en el año 1999 por el Dr. Andy Stanford-Clark de IBM y Arlen Nipper de Arcom (actualmente Eurotech). Fue diseñado para el transporte de mensajes extremadamente ligeros mediante la suscripción y publicación; es fácil de implementar y útil para conexiones en sitios remotos donde las redes posean poco ancho de banda y los dispositivos IoT trabajen con recursos limitados de memoria, batería y procesamiento. Además, intenta garantizar confiabilidad y seguridad en la entrega [13].

Su estandarización se encuentra en proceso desde marzo de 2013 en OASIS (*Organization for the Advancement of Structured Information Standards*), un consorcio sin fines de lucro que impulsa el desarrollo, convergencia y adopción de estándares abiertos para la sociedad de la información global. OASIS está conformada por más de 5000 representantes de 600 organizaciones y miembros individuales en más de 65 países [14]. La especificación del protocolo ha sido publicada abiertamente con licencia libre de regalías, en noviembre de 2011; además, IBM y Eurotech anunciaron su participación en el Grupo de trabajo de la industria Eclipse M2M y la donación del código MQTT al proyecto Eclipse Paho.

Los puertos estándar que usa MQTT son, el puerto TCP 1883 reservado por la IANA (*Internet Assigned Numbers Authority*) para su uso en MQTT y el puerto TCP 8883 registrado para usar MQTT sobre SSL (*Secure Sockets Layer).*

#### 1.3.4.1. Modelo suscripción y publicación

MQTT define dos tipos de entidades, un *broker* y varios clientes. La primera entidad es un servidor que recepta mensajes de los clientes y los redirige a los clientes destino. La segunda entidad es un objeto del mundo real que puede conectarse e interactuar con el *broker* para recibir sus mensajes; los clientes pueden ser desde sensores hasta centros de datos.

Los mensajes son organizados por temas o tópicos, así un cliente que se conecta al *broker* puede suscribirse a un tema de mensajería del *broker*, y puede publicar mensajes de un tema específico. El *broker* es el encargado de redirigir el mensaje a todos los clientes suscritos a un determinado tema; esta forma de comunicación se refleja en la Figura 1.5.

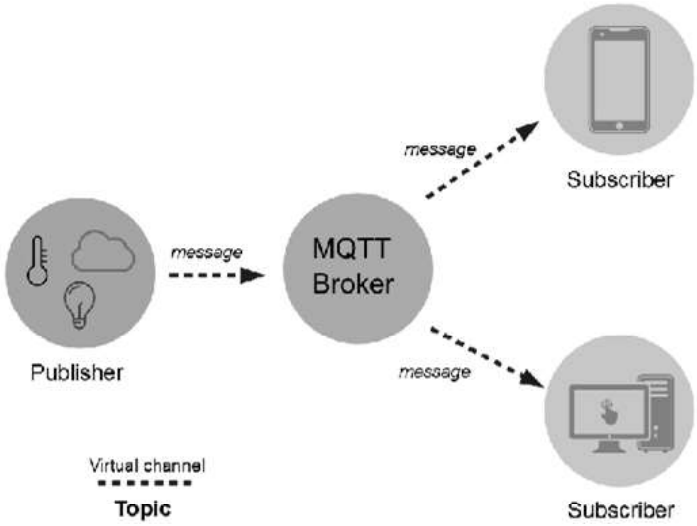



**Figura 1.5.** Modelo publicador/suscriptor [15]

Los temas son especificados por el desarrollador con el fin de que ciertos clientes interactúen con determinados mensajes, tomando una funcionalidad tipo canal virtual.

#### 1.3.4.2. Características de MQTT

MQTT provee una distribución de mensajes de uno a varios mediante el modelo publicador/suscriptor; el transporte de mensajes es agnóstico al contenido de la carga útil. Trabaja con calidad de servicio en la entrega de los mensajes, usando tres niveles que son:

- *At most once* (nivel 0): Este nivel de servicio garantiza una entrega del mejor esfuerzo, donde el destinatario no envía un acuse de recibo del mensaje y el remitente no almacena ni retransmite el mensaje, por lo que pueden ocurrir pérdidas o duplicación de mensajes.
- *At least once* (nivel 1): Este nivel garantiza que el mensaje será entregado al menos una vez al receptor, para esto el remitente almacena el mensaje hasta que reciba un mensaje de confirmación. En este nivel es probable que ocurra la duplicación de los mensajes.
- *Exactly once* (nivel 2): Este es el nivel de servicio más alto y garantiza que cada mensaje sea recibido una sola vez por el destinatario, por lo que es el nivel más seguro y también más lento debido a que hace uso de al menos dos flujos de solicitud/respuesta entre el remitente y el destinatario. Si el mensaje se pierde en el

camino, el remitente es el responsable de retransmitir el mensaje luego de una cantidad de tiempo determinada.

MQTT trabaja con una sobrecarga pequeña en el transporte; la longitud fija del encabezado es de 2 bytes, y provee un mecanismo para notificar la desconexión anormal de un cliente a los interesados mediante la característica *Last Will & Testament*[3].

#### 1.3.4.3. Formato del mensaje MQTT

Un mensaje MQTT consta de un encabezado fijo de 2 bytes que siempre está presente, un encabezado variable y el *payload* que no siempre están presentes [16].

Existen algunas versiones del protocolo, entre ellas la versión MQTT V3.1 que fue estandarizada por Eurotech e IBM, y a partir de ésta se tienen los siguientes formatos de mensaje.

- **Encabezado Fijo**

El encabezado de cada mensaje de comando MQTT sigue el formato de la Tabla 1.1.

**Tabla 1.1.** Formato de encabezado fijo MQTT [17]

| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| **Byte 1** | Tipo de mensaje | | | | Bandera DUP | Nivel QoS | | RETAIN |
| **Byte 2** | Longitud restante | | | | | | | |

**a.** El campo tipo de mensaje está representado como un valor sin signo de 4 bits; las opciones de tipos de mensajes para la versión MQTT V3.1 se muestran en la Tabla 1.2.

**b.** El campo de la bandera DUP se ubica en el bit 3 del byte 1, el mismo que se configura cuando el cliente o el servidor intentan volver a entregar un mensaje PUBLISH, PUBREL, SUBSCRIBE O UNSUBSCRIBE. Esta bandera se usa cuando el nivel de calidad de servicio es mayor a 0 y se requieren acuses de recibo.
Cuando el bit DUP es seteado a 1, el encabezado incluye un ID de mensaje para identificarlo.

**c.** El campo QoS ocupa los bits 1 y 2 del byte 1 e indica el nivel de seguridad en la entrega de un mensaje PUBLISH; los niveles QoS se muestran en la Tabla 1.3.

[3] ***Last Will* & *Testament*:** Es una característica que notifica a otros clientes acerca de la desconexión inesperada de un cliente.

**Tabla 1.2.** Campo Tipo de Mensaje[17]

| Mnemónico | Enumeración | Descripción |
|---|---|---|
| Reservado | 0 | Reservado |
| CONNECT | 1 | Cliente solicita conectarse al servidor |
| CONNACK | 2 | Acuse de recibo de conexión |
| PUBLISH | 3 | Mensaje a publicar |
| PUBACK | 4 | Acuse de recibo del mensaje PUBLISH |
| PUBREC | 5 | Recepción del mensaje PUBLISH |
| PUBREL | 6 | Descarte del mensaje PUBLISH |
| PUBCOMP | 7 | Publicación completa |
| SUBSCRIBE | 8 | Solicitud de suscripción del cliente |
| SUBACK | 9 | Acuse de recibo de suscripción |
| UNSUBSCRIBE | 10 | Solicitud de cancelación de suscripción del cliente |
| UNSUBACK | 11 | Acuse de recibo de cancelación de suscripción |
| PINGREQ | 12 | Solicitud de PING |
| PINGRESP | 13 | Respuesta de PING |
| DISCONNECT | 14 | Cliente está desconectándose |
| Reservado | 15 | Reservado |

**Tabla 1.3.** Campo QoS [17]

| Valor QoS | Bit 2 | Bit 1 | Descripción | |
|---|---|---|---|---|
| **0** | 0 | 0 | *At most once* | Dispara y olvida |
| **1** | 0 | 1 | *At least once* | Entrega con acuse de recibo |
| **2** | 1 | 0 | *Exactly once* | Entrega asegurada |
| **3** | 1 | 1 | Reservado | |

**d.** El campo RETAIN ocupa el bit 0 del byte 1, y se usa solamente en mensajes PUBLISH. Cuando un cliente envía un mensaje PUBLISH con esta bandera seteada a 1, el servidor debe conservar el mensaje luego de hacer la entrega a los suscriptores actuales, de manera que cuando se establezca una nueva suscripción al tema, el último mensaje retenido sobre dicho tema será enviado al nuevo suscriptor con la bandera RETAIN establecida; en caso de que no haya un mensaje retenido, no se envía nada.

**e.** El byte 2 representa el número de bytes restantes que quedan dentro del mensaje actual, incluidos los datos en el encabezado variable y la carga útil.

- **Encabezado variable**

Algunos tipos de mensajes de comando MQTT contienen un componente de encabezado variable que se ubica entre el encabezado fijo y la carga útil. El formato del encabezado variable se puede apreciar en la Tabla 1.4.

**Tabla 1.4.** Formato del encabezado variable MQTT [16]

| **Bytes** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|---|---|---|---|---|---|---|---|---|---|
| | Nombre del Protocolo | | | | | Versión del Protocolo | Banderas de Conexión | Keep Alive Timer | |

**a.** El campo nombre del protocolo se presenta en el encabezado variable de un mensaje MQTT CONNECT, es un campo tipo *string* que representa el nombre del protocolo.

**b.** El campo versión del protocolo está presente en el encabezado variable del mensaje CONNECT, campo que ocupa 8 bits de valor sin signo que representa el nivel de revisión del protocolo que se usa, así para la versión MQTT V3.1 el valor de este campo será 0x03.

**c.** Luego se tienen los indicadores de conexión conformados por las banderas *Clean session*, *Will Flag*, *Will QoS*, *Will Retain*, *User Name Flag* y *Password Flag*, las mismas que se presentan en un mensaje CONNECT. Su distribución en bits se puede observar en la Tabla 1.5.

**Tabla 1.5.** Campo banderas de conexión [17]

| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| | User Name Flag | Password Flag | Will Retain | Will QoS | | Will Flag | Clean Session | Reservado |

- El bit 0 de este byte no se usa en esta versión y está reservado para uso futuro.
- La bandera *Clean Session* ocupa el bit 1 entre las banderas de conexión; cuando está seteada a 0, el servidor debe almacenar las suscripciones de los clientes después de que se desconecten; además debe almacenar los mensajes con QoS 1 y 2 para los temas suscritos, de modo que se entreguen estos

mensajes cuando un cliente vuelva a conectarse. Si la bandera está seteada a 1, quiere decir que el servidor debe descartar toda información mantenida previamente acerca del cliente y se debe definir limpia la conexión. Esta bandera puede ser útil cuando un cliente no desea recibir información obsoleta, sin embargo, debe volver a suscribirse cada vez que se desconecte.

- La bandera *Will* ocupa el bit 2 y se encarga de que el servidor publique un mensaje a favor del cliente, cuando encuentre un error de entrada/salida durante la comunicación con el cliente, o si el cliente no se comunica dentro del cronograma del temporizador *Keep Alive*.
- La bandera *Will QoS* que ocupa los bits 3 y 4 se utiliza cuando un cliente que se conecta en un nivel específico de QoS configura estos bits para que se envíe un mensaje *Will,* en caso de que este cliente se desconecte de manera involuntaria. El mensaje *Will* se definirá en la carga útil de un mensaje CONNECT. Si la bandera *Will* está seteada a 0, entonces este campo *Will QoS* es obligatorio, de lo contrario este valor se ignora.
- La bandera *Will Retain* ocupa el bit 5 del byte de los indicadores de conexión, e indica si el servidor debería retener el mensaje *Will* que es publicado por el servidor en favor del cliente, en caso de que éste se desconecte inesperadamente. Al igual que la bandera *Will QoS*, esta bandera es obligatoria cuando la bandera *Will* esté seteada a 0, de lo contrario se ignora.
- Las banderas *Password* y *User Name* ocupan los bits 6 y 7 del byte de los indicadores de conexión. Estas banderas son útiles para un cliente que desea especificar un nombre de usuario y la contraseña correspondiente. Cuando la bandera *User Name* esté configurada, entonces el nombre de usuario será obligatorio; de la misma manera sucede con la bandera *Password*. No es válido proporcionar una contraseña sin establecer un nombre de usuario.

**d.** *Keep Alive Timer* es otro de los campos presentes en el encabezado variable de un mensaje MQTT CONNECT. Este campo define el intervalo de tiempo máximo medido en segundos entre mensajes recibidos desde un cliente, permitiendo que el servidor detecte la interrupción de la conexión de red a un cliente sin necesidad de esperar el tiempo estipulado por el protocolo TCP/IP. Este campo ocupa 16 bits que representan el número de segundos para el periodo de tiempo. El cliente tiene la responsabilidad de enviar un mensaje dentro de cada *Keep Alive Timer*; en caso de que no haya un mensaje relacionado con datos, el cliente envía un mensaje PINGREQ, que el servidor confirmará con un PINGRESP. Luego de un tiempo y medio de espera de recepción del mensaje desde el cliente, el servidor desconecta

al cliente como si hubiera recibido un mensaje de desconexión del cliente. Si un cliente no recibe el mensaje PINGRESP dentro de un tiempo de *Keep Alive* luego de haber enviado un mensaje PINGREQ, cierra la conexión del *socket* TCP/IP.

Existen otros campos que son parte de los mensajes CONNACK y PUBLISH, y son:

**e.** El campo código de retorno de la conexión se envía en el encabezado variable de un mensaje CONNACK. Define un byte de código de retorno sin signo, las opciones de este campo se encuentran en la Tabla 1.6.

**Tabla 1.6.** Campo Código de retorno de la conexión [17]

| Numeración | Valor HEX | Significado |
|---|---|---|
| 0 | 0x00 | Conexión aceptada |
| 1 | 0x01 | Conexión rechazada: versión de protocolo inválido |
| 2 | 0x02 | Conexión rechazada: identificador rechazado |
| 3 | 0x03 | Conexión rechazada: servidor no disponible |
| 4 | 0x04 | Conexión rechazada: nombre de usuario o contraseña incorrectos |
| 5 | 0x05 | Conexión rechazada: no autorizado |
| 6-255 | | Reservado para uso futuro |

**f.** El campo Nombre de tópico está presente en el encabezado variable de un mensaje PUBLISH, el cual define la clave para identificar el canal de información por el cual la carga útil es publicada; los suscriptores la usan para identificar los canales de información en donde desea recibir la información publicada. Este campo es de tipo *string* codificado en UTF-8 y su tamaño límite es de 32 767 caracteres.

Los mensajes MQTT que tienen carga útil son los mensajes CONNECT, SUBSCRIBE y SUBACK.

**g.** Los identificadores de mensajes (*Message ID*), están presentes en el encabezado variable de los mensajes PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE y UNSUBACK. Este campo está presente solo en los mensajes donde el encabezado fijo tiene el nivel de QoS mayor a 0; tiene una longitud de 16 bits, es de tipo entero sin signo y debe ser único entre todos los mensajes que están transmitiéndose en una dirección particular de comunicación. Tanto el cliente como el servidor mantienen una lista de los

identificadores de mensajes, y el orden de los bytes es del tipo *big-endian* (orden de bytes desde los bits más significativos a los menos significativos).

#### 1.3.4.4. Semántica de temas o tópicos

Para describir un tema, se utiliza un separador de nivel de temas que es el caracter especial *slash* "/". El uso del separador de nivel de temas es importante para separar cada nivel dentro de un árbol de temas y proporciona una estructura jerárquica.

Otro caracter utilizado como comodín multinivel es el signo numeral "#" que relaciona varios niveles dentro de un tema, este caracter puede representar cero o más niveles.

El comodín para un solo nivel es el caracter "+" y relaciona solo un nivel de tema; este comodín se puede utilizar en cualquier nivel del árbol de temas junto con el comodín multinivel.

Para la escritura de temas o tópicos se deberían seguir las siguientes recomendaciones:

- Un tópico debería tener al menos una cadena larga.
- Los nombres de los temas distinguen entre mayúsculas y minúsculas.
- En un tema se puede incluir el caracter de espacio.
- No se debe incluir el caracter nulo.
- Para un árbol de temas, no hay límites en el número de niveles.

### 1.3.5. PROYECTO ECLIPSE PAHO

Este proyecto fue creado para proveer implementaciones de código abierto escalables, para protocolos de mensajería abiertos como MQTT y MQTT-SN, así también para estándares destinados a aplicaciones de M2M (*Machine to Machine*) e Internet de las Cosas. El alcance de este proyecto incluye el desarrollo de herramientas que apoyan el uso efectivo, la integración y prueba de los componentes de mensajería [18].

El objetivo de este proyecto es brindar niveles efectivos de desacoplamiento entre dispositivos y aplicaciones, diseñados para mantenerlos abiertos a fin de alentar el crecimiento de aplicaciones web y empresariales escalables. Paho proporciona implementaciones de cliente MQTT con el modelo publicador/suscriptor para diferentes plataformas integradas con el soporte del servidor correspondiente, según la conveniencia del usuario.

El proyecto Paho trabaja con las bibliotecas cliente MQTT de código abierto y actualmente existen bibliotecas MQTT C, JAVA, Rust, Go, Python, C++ y Java*Script* [19].

#### 1.3.5.1. Cliente Paho MQTT C

El Cliente Paho MQTT C es un cliente MQTT con todas las funcionalidades escritas en el estándar ANSI C. Existen dos APIs (*Application Programming Interface*) C, una sincrónica y la otra asincrónica donde las llamadas a las APIs son MQTTClient y MQTTAsync respectivamente. La primera API es más sencilla y útil, debido a que algunas de las llamadas se bloquean hasta que una operación finalice, mientras que la API asíncrona solo bloquea una llamada. Utilizan devoluciones de llamada para notificar resultados [20].

Este cliente Paho soporta varias características como: las versiones MQTT V3.1, MQTT V3.1.1, MQTT V5.0, también SSL/TLS, persistencia del mensaje, reconexión automática, almacenamiento en búfer *offline*, soporte *WebSocket*, soporte del estándar TCP, alta disponibilidad y soporte de APIs bloqueantes y no bloqueantes.

Los clientes Paho C se conectan a un *broker* sobre una conexión TCP/IP, y está disponible tanto para ambientes Linux como Windows, en ambos casos es necesario instalar primero los repositorios y dependencias para el cliente MQTT.

#### 1.3.5.2. Servicio Android Paho

El Servicio Android Paho es un cliente MQTT codificado en Java para el desarrollo de aplicaciones en Android. Este cliente provee varias características como: soporte de las versiones MQTT V3.1, MQTT V3.1.1, soporte de LWT (*Last Will & Testament*), SSL/TLS, persistencia del mensaje, reconexión automática, almacenamiento en búfer *offline*, soporte *WebSocket*, soporte del estándar TCP, alta disponibilidad y soporte de APIs no bloqueantes [21].

Para iniciar con el desarrollo de este cliente es necesario utilizar el programa Android Studio, en el cual se deben añadir la definición del repositorio y la dependencia correspondientes a las bibliotecas, que Eclipse alberga en un repositorio Nexus y Maven para administrar sus dependencias.

### 1.3.6. ZOLERTIA RE-MOTE

Zolertia es una empresa desarrolladora de soluciones de hardware para el Internet de las Cosas, hacia el despliegue de aplicaciones inteligentes. Ofrece una línea de diversas plataformas IoT para crear prototipos y desarrollar dispositivos conectados, que fácilmente formen parte de una aplicación final; todo esto gracias a su módulo central, denominado Zoul. Provee plataformas para la Industria 4.0, abriendo un nuevo horizonte en las industrias para brindar mantenimiento preventivo, seguridad y seguimiento de maquinarias; también trabaja en plataformas para el proyecto de ciudades inteligentes, a fin de desarrollar soluciones para un mundo más eficiente. Asimismo, desarrolla plataformas para

el área de las construcciones inteligentes que requieren conectividad entre todos los equipos y sistemas; además, se enfoca en ofrecer soluciones para las universidades y centros de investigación que trabajan en el estudio y desarrollo de aplicaciones IoT.

El módulo Zoul en el que se basa Zolertia comprende los dispositivos integrados CC2538 y CC1200 en un formato de módulo único, lo que permite una rápida reutilización de los componentes principales de la plataforma RE-Mote en distintos formatos, dependiendo de los requisitos específicos de los usuarios y aplicaciones. La plataforma Zolertia RE-Mote se puede apreciar en la Figura 1.6.


**Figura 1.6.** Plataforma Zolertia RE-Mote [22]

Esta plataforma presenta entre sus principales características [22], las siguientes:

- Soporta el protocolo IEEE 802.15.4 y Zigbee, con transceptor a 2.4 GHz.
- Se basa en SoC[4] (*System On a Chip*) CC2538 ARM Cortex-M3, operación hasta 32 MHz, con 512 KB de memoria flash programable y 32 KB de memoria RAM.
- El rango máximo de alcance va desde 100 m y 20 Km, gracias a los dos radios y también a parámetros configurables como modulación, velocidad de datos, potencia de transmisión, etc.
- Consumo de apenas 150 nA en modo *shutdown*.
- Cargador de batería incorporado (500 mA), con facilidad para conectar paneles solares y baterías LiPo.
- Amplia gama de alimentación del voltaje DC de 3.3 a 16 V.
- Botón de usuario y *reset*.
- Antena externa de 2.4 GHz que se conecta a través del conector RP-SMA.

[4] **SoC:** Tendencia a integrar todos los módulos de un sistema en un solo chip o circuito a fin de obtener dispositivos de menor tamaño.

- Soportado en sistemas operativos de código abierto como Contiki, RIOT y OpenWSN.

La distribución de pines en la revisión B de la plataforma, es la que se muestra en la Figura 1.7.

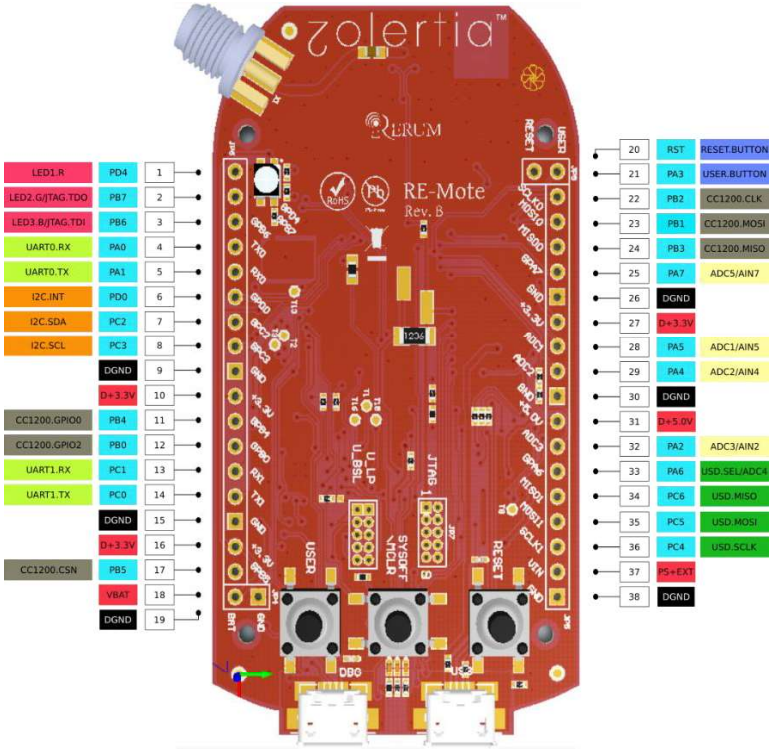



**Figura 1.7.** RE-Mote revisión B [23]

RE-Mote contiene 38 pines de diferentes tipos: pines digitales de entrada y salida, pines de salida y entrada del voltaje de alimentación y pines de tierra digital; además, tiene dos conectores ADC (*Analog to Digital Converter*), uno para sensores de 3 V y el otro para sensores de 5 V, y un conector digital que permite conectar dispositivos digitales usando protocolos como I2C (*Inter-Integrated Circuit*), SPI (*Serial Peripheral Interface*) o UART (*Universal Asynchronous Receiver Transmitter*). Para la programación del dispositivo Zolertia RE-Mote se usa el puerto micro USB de depuración y programación; existe un puerto micro USB adicional para aplicaciones USB 2.0.

### 1.3.7. CONTIKI

Contiki es un sistema operativo de código abierto orientado al Internet de las Cosas, para conectar microcontroladores de baja potencia, bajo costo y de tamaño pequeño a Internet; permite construir sistemas inalámbricos complejos mediante un conjunto de herramientas. Es compatible con los protocolos estandarizados IPv4 e IPv6, junto con los estándares

inalámbricos de bajo consumo: 6LowPAN, RPL (*Routing Protocol over Low-Power and Lossy Networks*), CoAP, etc. Contiki está escrito en lenguaje de programación C aunque también Java y Python se pueden usar en el entorno de ejecución, y mediante el simulador Cooja permite emular redes antes de su despliegue en un entorno real. Actualmente Contiki continúa creciendo gracias a una comunidad de desarrolladores grande y activa que incluye a profesionales, investigadores y académicos.

Abordando en la historia, Contiki inició gracias a que Adam Dunkels tuvo la necesidad de conectar cosas a Internet. En el año 2004 se introdujo el concepto de *protothreads* en Contiki; Instant Contiki y el sistema de archivos Coffee se introdujeron a principios del año 2008, más tarde en el mismo año, Cisco contribuyó con el *stack* IPv6 completamente certificado para Contiki. En los años 2009 y 2010 se agregaron varias plataformas nuevas y también se desarrollaron nuevos mecanismos de bajo consumo; en el año 2011 se añadieron ContikiRPL para enrutamiento IPv6 y ContikiMAC para enrutadores inactivos. En el año 2012 se fundó Thingsquare para conectar Contiki a la nube [24].

Entre las ventajas más relevantes de Contiki están, la necesidad de solo unos pocos kilobytes para su ejecución; además, incluye una interfaz gráfica de usuario, un software de red y un navegador web, lo que hace que sea más fácil ejecutar programas en chips pequeños y de baja potencia. Requiere al menos de 2 KB de memoria RAM y 30 KB de memoria ROM para ejecutarse.

Para el uso más fácil de Contiki, se ha creado Instant Contiki, el cual es un entorno de desarrollo de Contiki completo que se puede obtener en una sola descarga; es una máquina virtual Ubuntu Linux ejecutable en VMWare o VirtualBox que posee Contiki con todas sus herramientas de desarrollo, compiladores y simuladores.

Contiki posee un sistema de compilación destinado a facilitar la ejecución de Contiki directamente en el hardware, dicho sistema funciona para las distintas plataformas de hardware, de manera que los comandos de compilación son familiares entre diferentes plataformas. Este sistema de compilación consiste en un conjunto de archivos denominados *Makefiles*.

Existen varias plataformas en las que se puede trabajar con Contiki, entre ellas están RE-Mote, Wismote, Sky, Z1, Avr-atmega128rfa, Micaz, etc. Su código fuente se publica bajo una licencia de estilo BSD de 3 cláusulas, y bajo esta licencia Contiki se puede usar libremente tanto en sistemas comerciales como no comerciales, siempre que se mantenga el encabezado de *copyright* en los archivos de código fuente. Los derechos de autor del código fuente de Contiki son propiedad de las personas u organizaciones que lo

desarrollaron, todo esto bajo la licencia ya mencionada. La licencia de código abierta brinda libertad al desarrollador para compartir o no su código, solamente que, si desea compartirlo es necesario que el código esté bajo la misma licencia de Contiki.

#### 1.3.7.1. *Stack* de Contiki

El *stack* de red de Contiki, está definido por cuatro capas principales, donde se albergan algunos protocolos que se muestran en la Figura 1.8.

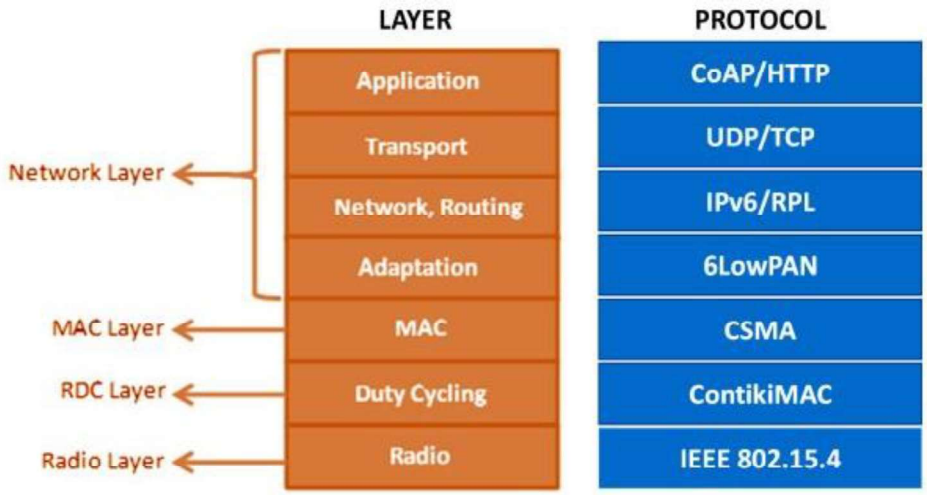



**Figura 1.8.** Stack de red de Contiki [25]

Cada una de las capas tienen sus funciones y protocolos [26], como se indica a continuación:

**a.** En la parte más baja del *stack* de protocolos de Contiki se tiene a la capa de Radio, donde se reciben los datos en bytes o como datos completos mediante un controlador de interrupciones, y luego son procesados en el búfer; mediante un proceso de poleo se enviará un evento especial para después pasar los datos a capas superiores. También aquí se define el controlador de interfaz de radio y la funcionalidad del hardware del transceptor. El protocolo que se usa en esta capa es el IEEE 802.15.4, estándar que define el nivel físico de las redes inalámbricas de área personal con bajas tasas de transmisión.

**b.** La segunda capa es la capa de Ciclo de Trabajo del Radio, RDC (*Radio Duty Cycling*), encargada de manejar el período de descanso de los nodos; decide cuándo los paquetes serán transmitidos y asegura que los nodos estén activos para que los paquetes sean recibidos. Esta capa utiliza *drivers* que tratan de mantener el radio apagado tanto como sea posible, revisando periódicamente el medio inalámbrico por alguna actividad de la radio; si detecta actividad, el radio se

mantiene encendido para revisar si tiene que recibir un paquete o volver al estado de descanso. La frecuencia de verificación del canal está dada en Hz y especifica el número de veces que el canal es comprobado por segundo.

El objetivo de esta capa es ahorrar la energía del dispositivo, para lograr una larga vida útil. El protocolo que se utiliza es ContikiMAC, porque el ciclo del radio se implementó en la capa MAC, sin embargo, Contiki decidió separarlo a su propia capa llamada RDC.

**c.** La capa de Control de Acceso al Medio (MAC), tiene como objetivo el evitar las colisiones en el medio inalámbrico mediante la técnica CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*). CSMA/CA sensa el medio antes de transmitir, si el medio está libre entonces transmite, pero si alguien está transmitiendo espera un tiempo definido; el tiempo de espera está determinado por la capa RDC. CSMA/CA recibe los paquetes desde la capa RDC y también envía paquetes a esta capa.

**d.** La capa de Red está conformada por cuatro subcapas, que son:

- La subcapa de Adaptación utiliza el protocolo 6LoWPAN, el mismo que proporciona la compresión y fragmentación del encabezado de IPv6 y UDP para transportar paquetes de IPv6 sobre IEEE 802.15.4; es decir, el protocolo 6LoWPAN define el estándar para la comunicación IPv6 sobre la tecnología de comunicación inalámbrica IEEE 802.15.4. Esta capa se denomina de adaptación, porque adecúa el estándar IPv6 al medio de comunicaciones inalámbricas de baja potencia con pérdidas, que ofrece IEEE 802.15.4.
- La subcapa de Red o Enrutamiento trabaja mediante el protocolo RPL, que define el diseño del protocolo IPv6 para redes con pérdidas y baja potencia; creado por el grupo ROLL (*Routing Over Low power and Lossy networks*) de IETF. RPL es un protocolo de vector distancia proactivo, que inicia encontrando las rutas tan pronto como la red es inicializada; utiliza etiquetas para identificar y mantener una topología. Además, RPL soporta confidencialidad e integridad en los mensajes mediante tres modos básicos de seguridad, no seguro, preinstalado y autenticado.
- En la subcapa de Transporte se definen los protocolos TCP y UDP, que permiten el intercambio de mensajes de una aplicación a otra mediante la división de la información en paquetes, y el añadido de una cabecera de paquete con información valiosa para su procesamiento en capas superiores. Estos protocolos utilizan puertos para identificar el destino de los mensajes. Existen diferentes usos según el protocolo; el protocolo UDP se utiliza para

realizar conexiones de datagrama entre aplicaciones de sistemas principales de Internet mientras que TCP proporciona entrega continua y confiable de datos entre sistemas de Internet.

- Por último, se encuentra la subcapa Aplicación, compuesta por varios protocolos como CoAP, HTTP (*Hypertext Transfer Protocol*) y MQTT, donde CoAP y MQTT están basados en estándares abiertos, se adaptan con mayor facilidad en escenarios con restricciones y están basados en IP. CoAP fue diseñado para interactuar con la Web, con el fin de trasladar el modelo HTTP al Internet de las Cosas, incluyendo *multicast*, baja sobrecarga y simplicidad. HTTP es un protocolo basado en petición-respuesta entre un cliente y un servidor que permite la transferencia de información en la Web; sin embargo, no es ideal para IoT debido a que es muy pesado para trabajar con dispositivos IoT. El protocolo MQTT se basa en la transferencia de mensajes a través del modelo publicador/suscriptor.

### 1.3.8. RASPBERRY PI

**Figura 1.9.** Raspberry Pi 3 Model B [27]

Es un pequeño ordenador, útil para programar mediante proyectos prácticos, de placa única y bajo costo desarrollado por la Fundación Raspberry Pi. Usa el sistema operativo Raspbian, versión adaptada de Debian; además existen las versiones tanto para Windows como para Mac. Los sistemas operativos se pueden obtener mediante descargas de la página oficial de Raspberry Pi, promoviendo así el aprendizaje de lenguajes de programación como Python, Tiny BASIC, C, Perl y Ruby.

Se han lanzado varios modelos, de los cuales en este proyecto se utilizará el modelo Raspberry Pi 3 Model B, el cual se muestra en la Figura 1.9. Entre sus características principales, están:

- Procesador *Broadcom* BCM2837 de 64 bits, *Quad Core* 1.2 GHz.
- 1 GB de memoria RAM.
- Módulo BCM43438 para Wi-Fi y *Bluetooth Low Energy* (BLE).
- Interfaz 100 Base Ethernet.
- 40 pines GPIO extendido.
- 4 puertos USB.
- Interfaz HDMI.
- Puerto micro SD para cargar el sistema operativo y almacenar datos.
- Fuente de alimentación micro USB de hasta 2.5 A.

Para su funcionamiento es necesario hacer las conexiones de pantalla, teclado, *mouse* y finalmente la instalación del sistema operativo elegido, directamente o mediante la aplicación NOOBS que facilita la instalación.

### 1.3.9. RIEGO EN LA AGRICULTURA [28]

El riego es la aplicación artificial al suelo de la cantidad de agua requerida por el cultivo, en el momento oportuno y de manera uniforme y eficiente; la cantidad de agua necesaria puede variar dependiendo del agua que recibe el cultivo por la lluvia, aunque este factor no es controlable por el hombre. El riego es un proceso controlable que se hace en el suelo, porque allí se acumula el agua y la planta puede absorber el agua para su desarrollo y crecimiento; además, el riego de agua debe ser constante, de modo que no le falte agua a la planta ni tampoco realice mucho esfuerzo para conseguirla.

Los cultivos deben recibir solo la cantidad necesaria de agua, debido a que dicha cantidad será beneficiosa. Si la cantidad de agua excede o es insuficiente traerá como resultados una menor producción, y consecuentemente se desperdiciará el agua, los nutrientes y fertilizantes; además, ocasionará la erosión del suelo y asfixia en las raíces de las plantas.

La periodicidad en el riego de agua depende de la necesidad de la planta y se debe tomar en cuenta que exista agua disponible para las plantas entre dos riegos consecutivos. Entre las principales consecuencias de un riego deficiente están: un menor rendimiento de los cultivos por exceso o falta de humedad, pérdida de agua, pérdida de nutrientes ocasionado por exceso de agua, salinización, encochado y erosión del suelo.

#### 1.3.9.1. Relación del riego con el suelo [28]

El suelo es una capa delgada formada desde hace siglos, a través de la desintegración de las rocas, gracias a la acción del agua, de los cambios de temperatura y del viento; además es el depósito de almacenamiento de agua, aire, nutrientes, minerales, materia orgánica, organismos vegetales y animales, que son producto de la descomposición de plantas y animales por acción de los microorganismos.

El agua es muy necesaria para la vida de las plantas, porque las plantas utilizan el agua para mantener sus tejidos, transportar los nutrientes, la respiración y la nutrición. El agua absorbida por las raíces se utiliza en el proceso de la fotosíntesis; además, la disolución de los minerales y materia orgánica en el agua facilita su captación por las plantas. Existen problemas por la falta o exceso de agua en las plantas; la falta de agua detiene el crecimiento de las plantas que pueden llegar a marchitarse y morir, mientras que, un exceso de agua desplaza el aire del suelo provocando falta de oxígeno para la respiración de las raíces. También el agua es necesaria para la vida de los microorganismos, quienes junto con la materia orgánica aportan y liberan nutrientes y unen partículas minerales entre sí, creando condiciones adecuadas para que las plantas respiren, absorban el agua y los nutrientes para su desarrollo.

#### 1.3.9.2. Propiedades de los suelos [29]

Para realizar un riego eficiente, es importante conocer ciertas características del suelo:

**a.** Composición: El suelo está compuesto por una fase sólida, la cual a su vez se conforma de materia orgánica y partículas minerales separadas por espacios denominados poros, los cuales contienen aire y agua en concentraciones variadas manteniendo un equilibrio entre ambos, esta fase se denomina el espacio poroso.

**b.** Textura: Se refiere a la cantidad de arena, limo y arcilla existente en el suelo, de acuerdo a las proporciones relativas de los diversos tamaños en que se presentan las partículas minerales. Existe una clasificación del suelo según la textura:

- El suelo arenoso, es el suelo que tiene mayor cantidad de arena que limo y arcilla.
- El suelo arcilloso, es aquel que tiene mayor cantidad de arcilla que arena o limo.
- El suelo franco, es aquel donde la proporción de arena, limo y arcilla es equilibrada.

**c.** Estructura: Está definida por la forma en que se agrupan las partículas individuales de arena, limo y arcilla; los agregados se forman cuando las partículas individuales se agrupan y forman partículas mayores, para lo cual es necesaria la materia orgánica.

Según las formas de los agregados se tienen distintas estructuras del suelo:

- Estructuras granulares, son partículas individuales de arena, limo y arcilla agrupadas en granos pequeños casi esféricos, donde el agua puede circular fácilmente. Este tipo de estructura tiende a encontrarse en los suelos francos.
- Estructuras laminares, compuestas por partículas de suelo agregadas en láminas o capas finas acumuladas horizontalmente una sobre otra, estas láminas pueden estar traslapadas lo que dificulta la circulación del agua.
- Estructuras en bloques, son partículas que se agrupan en bloques casi cuadrados o angulares, resistiendo la penetración de las raíces y el movimiento del agua.
- Estructuras prismáticas, son partículas de suelo en forma de columnas verticales separadas por fisuras verticales diminutas, donde el agua circula con mayor dificultad y el drenaje es deficiente.
- Los suelos masivos o sin estructura, son partículas sueltas, sin formar agregados o bien se encuentran unidas formando una masa endurecida, donde no se puede diferenciar un tipo de estructura.

Algunas de las causas de las deformaciones de las estructuras son las labores excesivas o inadecuadas, el escaso contenido de materia orgánica, la compactación ocasionada por el uso de maquinaria y el impacto de la lluvia; sin embargo, la estructura se puede mejorar mediante una buena preparación del suelo e incorporación de materia orgánica [28].

**d.** Profundidad: Es el espesor del suelo, donde mientras más profundo sea éste sostendrá mejor a la planta, de modo que, las raíces se puedan extender más profundamente permitiendo que exista un mayor almacenamiento de agua.

**e.** Porosidad: Es la parte del volumen total del suelo ocupado por los poros, o espacios de aire que tiene el suelo y está ocupado por agua. En los suelos arenosos se encuentran poros grandes, pero abarcan menor volumen en el suelo; mientras que en el suelo arcilloso existen poros pequeños. Conocer la porosidad del suelo sirve para saber la cantidad de agua que almacena un tipo de suelo, donde a mayor porosidad, mayor contenido de agua en el suelo.

**f.** Topografía: Es la forma tridimensional de un terreno, describe los cerros, valles, pendientes y elevación de la tierra, su importancia radica en que el agricultor pueda determinar el método de riego adecuado.

**g.** Fertilidad: Un suelo es considerado fértil cuando tiene los nutrientes necesarios para el adecuado desarrollo de la planta. Los nutrientes principales que requieren

las plantas son el nitrógeno, el fósforo, el potasio, el calcio y el magnesio, los mismos que deben estar presenten en cantidades y proporciones adecuadas.

**h.** Erosión: Es el desgaste, arrastre y pérdida de las partículas del suelo, debido a la acción de agua y viento. La erosión trae como consecuencias la pérdida de la fertilidad, la contaminación, la desaparición del suelo, disminución de la capacidad de retención de agua por la alteración en el contenido de la materia orgánica y en el porcentaje de minerales del suelo.

**i.** Infiltración: Es el movimiento del agua desde la superficie del suelo hacia abajo luego de una lluvia o riego; dicho movimiento depende del número de poros, de su tamaño, y continuidad, también depende de la textura y estructura del suelo. Cuando se tenga muchos poros grandes y continuos se favorecerá el movimiento del agua, es así que, el agua tendrá mayor velocidad de infiltración en un suelo arenoso que en un suelo arcilloso.

#### 1.3.9.3. Estado de humedad del suelo [28]

El suelo tiene la capacidad de retener agua para cuando las plantas la necesiten, y esta cantidad cambia continuamente, distinguiéndose los siguientes estados:

- Saturación: Es el contenido de humedad del suelo cuando se han llenado de agua todos los poros; esto sucede luego de una lluvia o riego abundante. En este punto, los poros no tienen aire y debido a esto, varios cultivos no pueden soportar períodos largos de tiempo en este estado porque se asfixian. El período de saturación de un suelo depende de la textura y velocidad de infiltración.
- Capacidad de campo: Es la cantidad de agua que retiene el suelo después de haber pasado por el estado de saturación, aquí los poros más grandes están con agua y aire, mientras que los poros pequeños siguen llenos de agua. Este estado es el ideal para el crecimiento de las plantas.
- Punto de marchitez: Es el contenido de humedad que tiene el suelo cuando la fuerza que hace la planta por absorber el agua, es igual a la fuerza con que el agua es retenida por el suelo. En este punto es imprescindible regar el agua para evitar que las plantas se marchiten y mueran.

El agua útil para la planta es el contenido de agua que el suelo es capaz de retener, entre los límites de capacidad de campo y el punto de marchitez. Cuando el contenido del agua se acerca al punto de marchitez, la planta tiene mayor dificultad de extraerla.

La medición de la humedad del suelo se puede hacer en un laboratorio, utilizando aparatos o por la apariencia del suelo. Este último se conoce como el método visual y de tacto, el

cual consiste en tomar en la mano una muestra de suelo, apretarla y estimar la humedad del suelo en base al tacto y a la experiencia; es muy sencillo, sin embargo, no se obtendrán resultados precisos del contenido de agua en el suelo.

#### 1.3.9.4. Riego por gravedad [30]

Es el riego donde el movimiento del agua se debe a su propio peso a lo largo del suelo, para lo cual es necesario preparar y nivelar el suelo a fin de que este movimiento no tenga inconvenientes. Este tipo de riego abarca el riego por escurrimiento, conocido también como riego por canteros, el cual es apto para regar una superficie con ligeras pendientes para facilitar el avance del agua. En la parte más alta tiene las entradas del agua y al final o parte más baja tiene una acequia que recibe el agua sobrante. Este tipo de riego se usa mayormente en los terrenos con abundante cantidad de agua para riego, el cual se puede observar en la Figura 1.10.

**Figura 1.10.** Riego por gravedad [30]

En los suelos arcillosos y con pendientes, se puede tener una velocidad alta del agua, siendo conveniente rebajar el caudal de riego y construir los surcos en zig-zag a fin de controlar dicha velocidad. En los suelos arenosos se requieren surcos pequeños para disminuir los tiempos de aplicación con poco volumen de agua.

El riego por gravedad presenta varias ventajas:

- Evita que las hojas se mojen disminuyendo la probabilidad de enfermedades en las hojas.
- La aplicación de fertilizantes y pesticidas al follaje no son lavadas por lo que se puede combinar ambas actividades al mismo tiempo.
- Sirve para lavar las sales del suelo.
- Existe un buen control sobre el agua de riego por la visibilidad.

- Pone a disposición de las raíces un gran volumen de suelo mojado.
- El riego se puede realizar a cualquier hora debido a que el viento no es un factor determinante.
- Bajo costo porque no necesita energía para mover el agua, ni equipos especiales.

También existen algunas desventajas con respecto a este tipo de riego, el desperdicio de un considerable volumen de agua, el previo nivelado del suelo y consecuentemente mayor costo de mano de obra, no aplicable en terrenos arenosos y provoca pérdidas en la fertilidad del suelo, que deberán ser compensadas con la adición de abonos.

#### 1.3.9.5. Problemática del riego en el Ecuador [31]

El Ecuador tiene el 16% de las reservas de agua dulce superficial del total mundial, y el riego es la actividad que tiene mayor demanda de volumen de agua, calculándose, en un 82%. Por este motivo es importante analizar los problemas de la disponibilidad de agua como el de la calidad de agua; una de las razones de estos problemas es la contaminación que genera impactos significativos en los ciclos vitales de los elementos naturales, ecosistemas y de la humanidad. Dicha contaminación es producida por las descargas de aguas servidas de poblados e industrias, derrames de petróleo y uso masivo de agroquímicos.

Gran parte de los ríos del país tiene una preocupante calidad bacteriológica, lo que ha causado la prohibición del consumo y contacto directo con estas aguas; además, la pérdida de cobertura vegetal que acarrea el uso agrícola del suelo, lo hace vulnerable a la erosión hídrica originando un problema de desestabilización del suelo y arrastre de materiales hacia el agua. El uso de agroquímicos tiene graves implicaciones tanto en la salud humana como en los agro ecosistemas; los residuos de los fertilizantes muchas veces tienden a depositarse en el suelo y agua. La SENPLADES ha indicado que algunos de los pesticidas cuya importación está prohibida, son utilizados en tareas de fumigación.

Otra problemática nacional tiene que ver con los procesos progresivos de degradación de los suelos, donde procesos de erosión estarían afectando al 48% de la superficie del país. Esto se da como consecuencia de la pérdida de la cobertura vegetal en bosques, humedales y páramos andinos, el uso intensivo e inadecuado de la mecanización agrícola, la práctica de monocultivos, el uso inadecuado del agua de riego, el uso de agroquímicos, la debilidad institucional en la regulación, seguimiento y control de prácticas inadecuadas y contaminantes del agua, suelo y tierra.

En la región Sierra, el 95% de las comunidades aproximadamente utilizan el método por inundación o por surcos; sin embargo, debido a la escasez de agua y demandas de riego

presurizado está creciendo un método alternativo, que requiere una inversión apreciable, un manejo más intensivo y especializado. Según el estudio realizado por la ESPOCH (Escuela Superior Politécnica de Chimborazo) para el Plan Nacional de Riego y Drenaje, la Sierra presenta una media de eficiencia de aprovechamiento del agua de riego a nivel parcelario, del 60.55%, donde los sistemas comunitarios aprovechan la pendiente y ubicación de las fuentes de agua en las zonas altas para conducir el agua mediante tubos y acequias hasta las parcelas [31].

### 1.3.10. SENSORES Y ACTUADORES

Un sensor se define como un dispositivo de entrada que provee una salida de la variable física medida, en forma de señales eléctricas analógicas o digitales [32].

Existen varias clasificaciones de sensores, una de ellas está dada según la variable física a medir. Se tienen así, sensores de posición, de velocidad y aceleración, de nivel y proximidad, de humedad y temperatura, de fuerza y deformación, entre otros.

A la hora de elegir un sensor es necesario analizar sus características principales, entre las cuales se tiene:

- Sensitividad: Definida como la entrada mínima necesaria para provocar una salida detectable.
- Rango: Es el intervalo comprendido entre el valor máximo y mínimo de la variable física a medir.
- Exactitud: Es la diferencia entre la salida actual del sensor y el valor real de la variable medida, y se expresa en porcentaje.
- Linealidad estática: Es la desviación existente entre la curva proporcionada por el fabricante en condiciones controladas y la curva de la salida del sensor.
- Tiempo de respuesta: Es el período de tiempo transcurrido durante el cambio producido en la variable física y el momento que el sensor lo detecta y registra.

Para la adquisición de sensores que se usen en aplicaciones de IoT, es recomendable analizar tanto las características anteriores, así como el factor de forma [5], protocolos de comunicación que usan, convertidores de señal analógico/digital o viceversa y costos.

Los sensores utilizados en el presente proyecto se pueden observar en la Figura 1.11.

[5] **Factor de forma:** Conjunto de estándares que definen características físicas, mecánicas y eléctricas de distintos dispositivos.

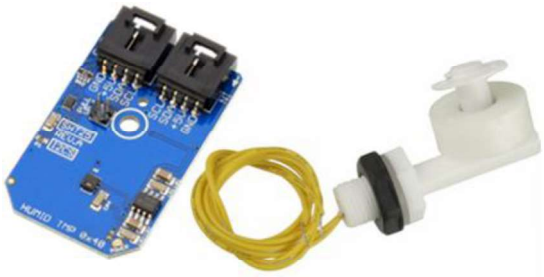


**Figura 1.11.** Sensores usados en el proyecto

Un actuador se define como un dispositivo capaz de generar una fuerza, utilizada para ejercer un cambio de estado sobre un elemento mecánico mediante la transformación de energía. En una de las clasificaciones de actuadores se tienen dos grupos, actuadores por el tipo de energía utilizada y por el tipo de movimiento que generan. Dentro de cada grupo existe una subdivisión; así se tienen a los actuadores neumáticos, hidráulicos y eléctricos en el primer grupo y a los actuadores lineales y rotatorios en el segundo.

Los actuadores más adquiridos por el mercado son los eléctricos que transforman la energía eléctrica en mecánica, teniendo como fuente de alimentación la energía eléctrica. En cambio, los actuadores neumáticos o hidráulicos requieren compresores para generar energía neumática o hidráulica respectivamente.

Los sensores y actuadores están compuestos por transductores, siendo un transductor un dispositivo capaz de convertir una forma de energía en otra. Existen dos tipos de transductores, los de entrada y salida. Los primeros se usan para medir una variable física para su posterior procesamiento, mientras que los segundos convierten la señal procesada en una acción.

El actuador que se usa en este proyecto se puede apreciar en la Figura 1.12.


**Figura 1.12.** Relé de estado Sólido SSR-100 DA [33]

#### 1.3.10.1. Sensor de Humedad y Temperatura

En el mercado existen varios tipos de sensores de humedad y temperatura, entre los cuales, a partir de una comparación de sus características y del análisis de las librerías que utilizan para su codificación, se escoge el sensor SHT25 de Sensirion. Las características de este sensor se describen en las Tablas 1.7. y 1.8.

**Tabla 1.7.** Parámetros de Humedad SHT25 [34]

| **Humedad Relativa** | | | |
|---|---|---|---|
| **Parámetro** | Condición | Valor | Unidades |
| **Resolución** | 12 bits | 0.04 | %RH |
| | 8 bits | 0.7 | %RH |
| **Exactitud** | típica | ±1.8 | %RH |
| **Tiempo de respuesta** | 63% | 8 | s |
| **Rango de operación** | extendido | 0 a 100 | %RH |

**Tabla 1.8.** Parámetros de Temperatura SHT25 [34]

| **Temperatura** | | | |
|---|---|---|---|
| **Parámetro** | Condición | Valor | Unidades |
| **Resolución** | 14 bits | 0.01 | °C |
| | 12 bits | 0.04 | °C |
| **Exactitud** | típica | ±0.2 | °C |
| **Tiempo de respuesta** | 63% | 5 a 30 | s |
| **Rango de operación** | extendido | -40 a 125 | °C |

El sensor SHT25 integra un sensor de humedad de tipo capacitivo y un sensor de temperatura de intervalo de banda con un circuito integrado analógico digital especializado. La resolución es modificable por comandos según se indica en las tablas anteriores; además, cada sensor está calibrado y probado individualmente. Todas estas características del sensor SHT25 lo hacen confiable, preciso y exacto en las mediciones. El sensor SHT25 se observa en la Figura 1.13.

Utiliza cuatro pines para su conexión, SDA, SCL, VDD y GROUND, donde SDA se usa para la transferencia de datos dentro y fuera del sensor, SCL se emplea para sincronizar la comunicación entre el microcontrolador y el sensor, VDD es utilizado para suministrar voltaje para el funcionamiento del sensor desde 2.1 a 3.6 V y GROUND funciona como la conexión a tierra del sensor.

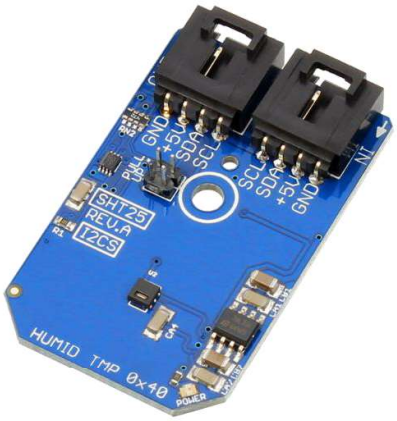



**Figura 1.13.** Sensor SHT25 [35]

El protocolo que usa el sensor SHT25 para la comunicación con un microcontrolador es I2C; para la codificación se definen las librerías correspondientes, y se usan códigos binarios para las mediciones de humedad y temperatura, los mismos que se muestran en la Tabla 1.9. Existen dos modos de operación para comunicarse con el sensor, *Hold Master* y *No Hold Master*. En el primer modo, la línea SCL se bloquea durante el proceso de medición, mientras que, en el segundo la línea SCL se mantiene abierta en caso de que ocurra otra comunicación, mientras el sensor está en proceso de medición.

**Tabla 1.9.** Comandos básicos SHT25 [34]

| Comando | Modo de Operación | Código |
|---|---|---|
| Medir Temperatura | *Hold Master* | 1110 0011 |
| Medir Humedad | *Hold Master* | 1110 0101 |
| Medir Temperatura | *No Hold Master* | 1111 0011 |
| Medir Humedad | *No Hold Master* | 1111 0101 |
| Escribir registro de usuario | | 1110 0110 |
| Leer registro de usuario | | 1110 0111 |
| Reiniciar | | 1111 1110 |

Para cada uno de los modos se tienen diferentes tipos de mensaje, sin embargo, ambos abarcan los comandos anteriores, acuses de recibo, datos y *checksum*.

#### 1.3.10.2. Sensor de nivel de agua

El sensor de nivel de agua es un interruptor de tipo flotador que permite medir si el agua ha llegado a un nivel determinado. Está compuesto de un interruptor y un flotador magnético de láminas; este último se ubica encapsulado en una barra guía. Cuando el

flotador sube y baja con el nivel de agua, sus imanes internos atraen el interruptor, de esta manera permite detectar si el agua ha llegado a un nivel predeterminado [36].

Externamente el material del sensor es polímero, el voltaje máximo de contacto es de 50 V DC, la máxima corriente de contacto es 0.5 A. Sus dimensiones son, 53.7 mm de largo, 28.2 mm de altura. Además, posee dos cables que permiten su conexión a otro dispositivo para obtener las lecturas de nivel. El sensor se puede ver en la Figura 1.14.

**Figura 1.14.** Sensor de nivel de agua [37]

### 1.3.11. ANDROID STUDIO

Android Studio es un entorno de desarrollo integrado (IDE) que provee herramientas para la creación de apps, para los dispositivos que tengan Android como sistema operativo; trabaja con una interfaz gráfica amigable, donde se puede diseñar, compilar y ejecutar las apps mediante los distintos bloques del editor como la barra de herramientas, propiedades, paleta, árbol de componentes, sección de ajuste de diseño.

Entre las opciones que se pueden configurar en la barra de herramientas están, la orientación de la pantalla, el tipo y tamaño del dispositivo, la versión de Android, el idioma, etc. Para diseñar apps se puede trabajar de dos formas, editando texto en XML en el editor de texto y a través del editor de diseño que es gráfico y se pueden arrastrar los componentes necesarios para el aplicativo. El área de trabajo de Android Studio se muestra en la Figura 1.15.

Este IDE ha sido actualizado continuamente y una de las últimas versiones disponibles es Android Studio 3.3, que incluye entre sus principales características [38], las siguientes:

- Mejoras de IntelliJ IDEA 2018.2.2 (una de las últimas versiones del IDE apropiado para *Java Virtual Machine*, JVM), incluye un análisis inteligente del código y nuevas refactorizaciones[6].
- Nuevo editor de navegación para visualizar y construir apps de forma más rápida.

[6] **Refactorización:** Técnica que mejora la optimización del código escrito previamente, alterando su estructura interna, pero sin cambiar el comportamiento del mismo

- Compilación incremental mejorada de Java.
- Sincronización de proyectos optimizada.
- Capacidad para crear paquetes de aplicaciones de Android con soporte completo para aplicaciones instantáneas.
- Descarga automática de paquetes NDK faltantes.

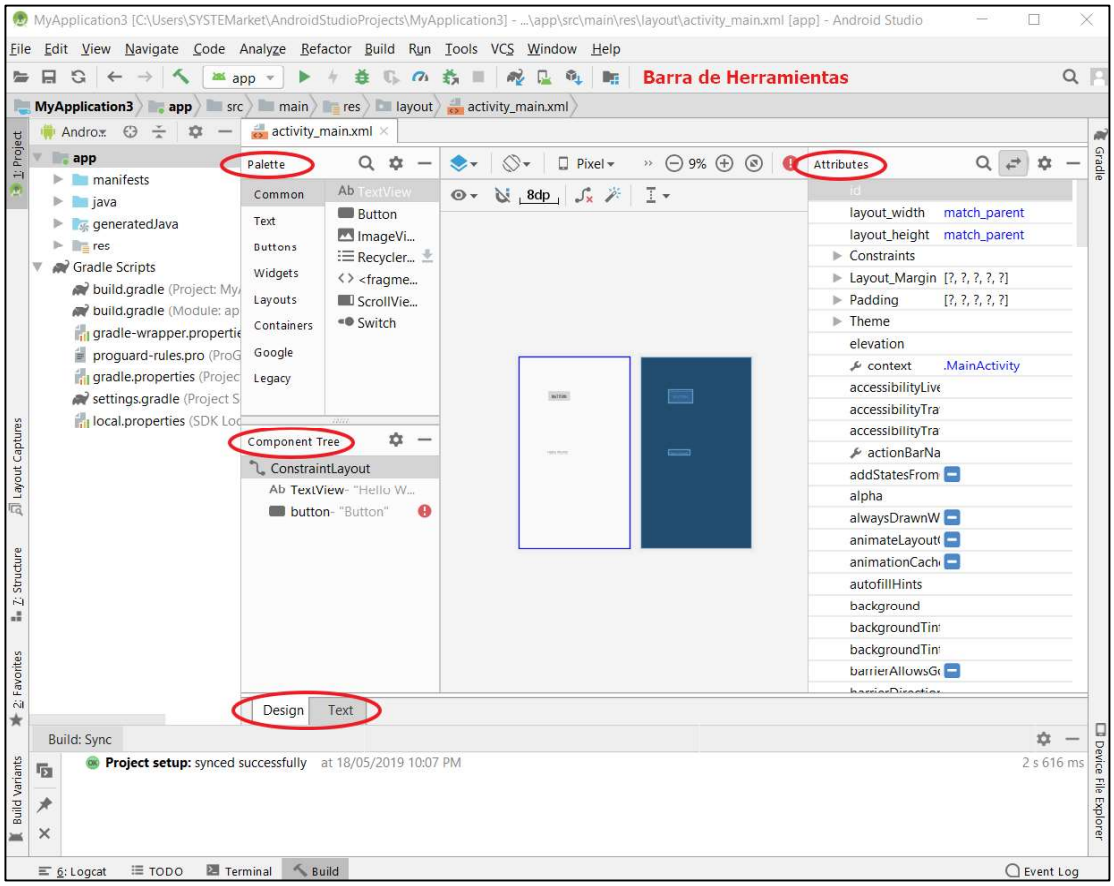



**Figura 1.15.** Interfaz de Android Studio

El gran desarrollo, despliegue y aceptación de Android Studio se debe al trabajo conjunto que realiza con varios sistemas, herramientas y lenguajes de programación. Este IDE trabaja con JAVA y XML para codificar las apps; el sistema de compilación está basado en Gradle (herramienta de compilación de código abierto flexible); utiliza Android SDK (Kit de desarrollo de software) Tools para brindar un conjunto completo de herramientas para desarrollo y depuración; además, incluye la familia de herramientas de código abierto CMake para compilar, probar y empaquetar software. Otras grandes ventajas de utilizar Android Studio son, la corrección rápida de código mediante la herramienta Lint, la posibilidad de crear e implementar aplicaciones instalables y experiencias instantáneas

mediante Android App Bundle. Y al igual que otros IDEs se puede trabajar con varias librerías para la codificación de clases y métodos para la app.

### 1.3.12. ESTUDIO DEL ENTORNO PARA LA IMPLEMENTACIÓN

El terreno para la implementación del prototipo se encuentra ubicado en la comunidad Gatazo Chico perteneciente a la parroquia Cajabamba, cantón Colta, provincia de Chimborazo, cuya forma es rectangular y sus medidas son 60 x 50 metros. La comunidad se encuentra en una zona geográficamente privilegiada, donde varios factores como el clima, la temperatura y humedad ayudan a la producción de hortalizas; entre los principales cultivos están, brócoli, coliflor, lechuga, cebolla, zanahoria, cilandro, papa, choclo y arveja.

El terreno se halla en una pendiente y muy cerca del mismo existe un pozo de agua subterránea, la misma que es succionada para riego mediante una bomba de agua agrícola, y con tubos PVC conectados de forma que llegan al terreno. El área de cultivo y el pozo de agua se pueden observar en la Figura 1.16.

Debido a que su textura es de tipo franco y su estructura es granular, se ha implementado el riego por canteros o surcos, donde el agua circula sin inconvenientes; otras dos de las razones por las cuales se utiliza este tipo de riego es el costo de la instalación de este sistema, pues es relativamente bajo en comparación con otros sistemas de riego y la cantidad de agua disponible para riego es suficiente para este tipo de riego.


**Figura 1.16.** Terreno y pozo de agua usados en el proyecto

La bomba de agua instalada trabaja con 5 HP a 60 Hz y 3450 RPM, el voltaje y corriente aproximados que utiliza es de 240 V, y 16 A respectivamente; la conexión para su encendido es monofásica. La bomba se puede apreciar en la Figura 1.17.

Consta de una salida y una entrada, ambas de 3 pulgadas de diámetro con el fin de transportar el agua por medio de tubos hacia el terreno; la distancia ideal para la succión es de 9 metros. Esta bomba de agua utiliza dos condensadores en su conexión interna, uno de 600 uF/125 V y otro de 50 uF/350V.


**Figura 1.17.** Bomba de agua W125275 [39]

### 1.3.13. METODOLOGÍA KANBAN

La metodología Kanban se enfoca en mejorar el diseño y administración de los sistemas de flujos de trabajo, mediante mecanismos de señalización visual para controlar el trabajo en curso de productos intangibles. Este método es aplicable a cualquier entorno de trabajo de conocimiento.

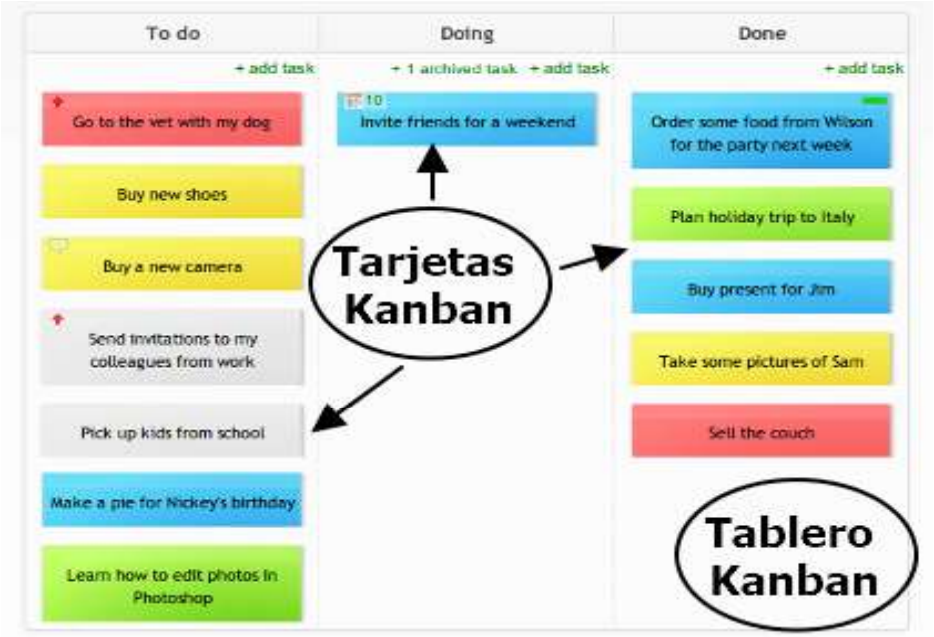



**Figura 1.18.** Tablero y tarjetas Kanban

Kanban se basa en tres principios fundamentales que son: la visualización del flujo de trabajo actual, la delimitación de la cantidad de trabajo en progreso con el fin de evitar la sobrecarga de trabajo y el mejoramiento del flujo de trabajo.

Las herramientas para trabajar con esta metodología son los tableros y tarjetas Kanban físicos o virtuales como se observa en la Figura 1.18. El tablero Kanban permite visualizar las tareas y optimizar el flujo de trabajo, el cual consta de tres pasos: por hacer, en curso y hecho, pasos que se reflejan en tres columnas respectivamente; las tarjetas Kanban representan un elemento de trabajo distinto, las mismas que son posteadas en las distintas columnas del tablero, de forma que todo el equipo esté enterado del progreso del trabajo de manera visual. Cada tarjeta presenta información clave sobre el elemento de trabajo, proporcionando visibilidad acerca de la persona encargada de ese elemento, información breve sobre el trabajo a realizar, duración estimada para su cumplimiento, etc. Para mejorar el flujo de trabajo se parte identificando los factores bloqueantes y eliminándolos.

# 2. METODOLOGÍA

Para el desarrollo de este trabajo de Titulación se desarrolla una investigación aplicada empleando la metodología ágil Kanban, la misma que se basa en el tablero y las tarjetas Kanban, con el fin de cumplir tareas predeterminadas que se complementan entre sí para el buen término del proyecto. En forma general, el tablero Kanban consta de algunas partes que son, objetivos, cola de tareas, aceptación y elaboración, desarrollo, pruebas, implementación y listo. La información del tablero es visible para todos, permitiendo que todo el flujo de trabajo y su progreso continuo sea claro para todos.

Para la determinación de ciertas tareas se han realizado entrevistas a los agricultores interesados en este proyecto; dichas entrevistas han permitido recoger información valiosa acerca del sistema actual de regadío que poseen los agricultores, a fin de que la implementación del proyecto se pueda adecuar a sus necesidades. Del análisis de las entrevistas realizadas se han derivado los requerimientos del proyecto, que han permitido extraer algunas tareas para la implementación del prototipo.

Luego, cada una de las tareas ha sido colocada en un tablero Kanban, describiendo a breves rasgos cada tarea; más tarde, durante el desarrollo de las tareas, éstas son removidas a través de las columnas del tablero hasta llegar a la columna listo, donde se reflejan las tareas que ya están completamente terminadas.

En la parte del desarrollo de las tareas se elaboran distintos diagramas UML (*Unified Modeling Language*) que permitirán visualizar de forma sintetizada, estructurada y ordenada los procesos o tareas a ejecutar. Además, se diseña la interfaz del aplicativo móvil en base a los requerimientos del usuario; finalmente se revisa la información acerca de los sensores adecuados para la implementación del proyecto.

## 2.1. LEVANTAMIENTO DE INFORMACIÓN

Los requerimientos del proyecto se derivan del análisis de las respuestas de los agricultores en las entrevistas ejecutadas en el mes de febrero de 2019; el modelo de la entrevista se puede observar en el Anexo A. El análisis se describe a continuación.

La primera pregunta tuvo como finalidad encontrar los parámetros más importantes que los agricultores tienen en cuenta al momento de realizar el proceso de riego de agua. Los entrevistados respondieron según se observa en la Figura 2.1; de las respuestas se concluye que la mayoría toma en cuenta principalmente la humedad del suelo y la etapa de desarrollo de la planta, seguidamente de la temperatura ambiental y finalmente la época del año. La lluvia no es tomada en cuenta en este proceso.

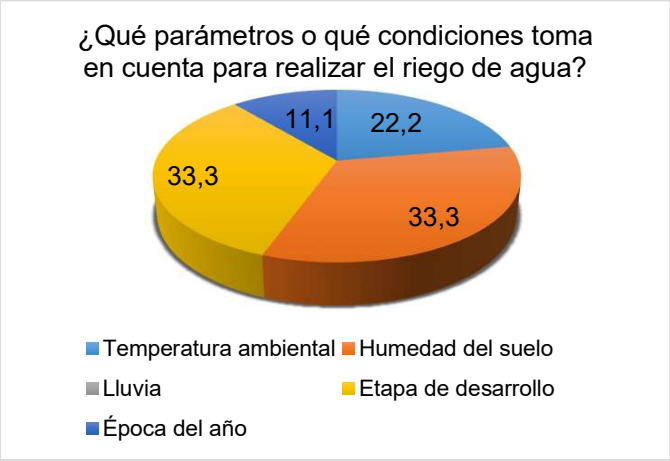



**Figura 2.1.** Resultados de la pregunta 1

La finalidad de la pregunta 2 fue identificar el tipo de riego mayormente utilizado; las respuestas a esta pregunta se las puede apreciar en la Figura 2.2, de las que se concluye que el total de las personas entrevistadas utilizan el riego por inundación.


**Figura 2.2.** Resultados de la pregunta 2

La pregunta 3 fue formulada con el fin de conocer el porcentaje de personas que tienen un pozo de agua a su disposición. Según se observa de sus respuestas que se presentan en la Figura 2.3., el 80% de las personas entrevistadas tiene a disposición un pozo de agua, en contraste con un 20% que no lo dispone.


**Figura 2.3.** Resultados de la pregunta 3

La pregunta 4 fue formulada con el fin de conocer si existe regularidad entre los riegos de agua o más bien depende del cultivo; a lo que las personas respondieron de acuerdo a como se observa en la Figura 2.4. Se puede concluir que el tiempo entre riego y riego depende de la necesidad del cultivo.


**Figura 2.4.** Resultados de la pregunta 4

La pregunta 5 tuvo como finalidad conocer la disponibilidad de agua de regadío al día; las respuestas se pueden apreciar en la Figura 2.5., de donde se entiende que más de la mitad de agricultores tiene a disposición entre 2 y 4 horas por día, y el resto entre 1 y 2 horas.


**Figura 2.5.** Resultados de la pregunta 5

Las siguientes 5 preguntas se enfocaron en descifrar el interés que las personas tienen en el desarrollo del proyecto. Así, la pregunta 6 fue formulada para conocer el estado actual de los sistemas de riego. Las respuestas se ven reflejadas en la Figura 2.6. de las que se deduce que nadie tiene instalado un sistema de riego automatizado.


**Figura 2.6.** Resultados de la pregunta 6

La pregunta 7 tuvo como finalidad, percibir el interés que se tiene en dos parámetros importantes al momento de realizar el riego mediante una bomba de agua. De las respuestas que se reflejan en la Figura 2.7. se concluye que la gran mayoría si considera necesario accionar la bomba de agua dependiendo de la temperatura y la humedad del suelo.

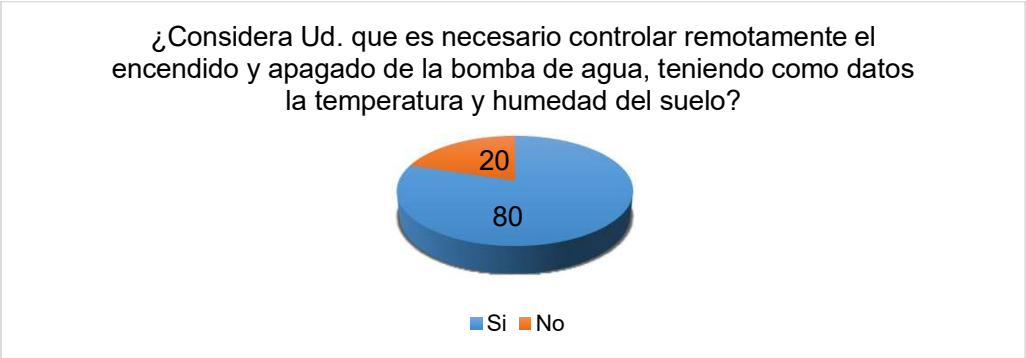



**Figura 2.7.** Resultados de la pregunta 7

El objetivo de la pregunta 8 fue acercar al agricultor con el diseño del proyecto y las opciones tecnológicas que ofrece el mismo. Las respuestas a esta pregunta se pueden observar en la Figura 2.8., de donde se percibe que a un gran porcentaje de agricultores les interesa la opción del control a distancia del accionamiento de la bomba de regadío, acorde a las necesidades, opción seguida por el control mediante el teléfono celular.

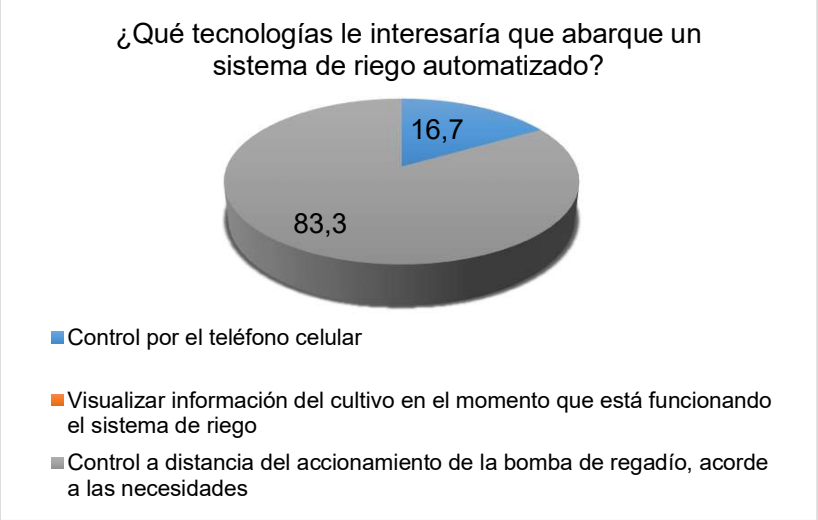



**Figura 2.8.** Resultados de la pregunta 8

En la pregunta 9 se desea conocer el grado de interés en adquirir el sistema propuesto. Las respuestas a esta pregunta se aprecian en la Figura 2.9., de donde se concluye que la totalidad de personas entrevistadas si están interesadas en adquirirlo.


**Figura 2.9.** Resultados de la pregunta 9

La última pregunta trata de llegar a conocer el costo que los clientes estarían dispuestos a pagar para disponer el sistema de riego propuesto; las respuestas de los agricultores se pueden ver en la Figura 2.10. El 40% de los entrevistados indica que estarían dispuestos a pagar 200 USD; otro 40% indica que pagaría 100 USD y el 20% restante 150 USD.

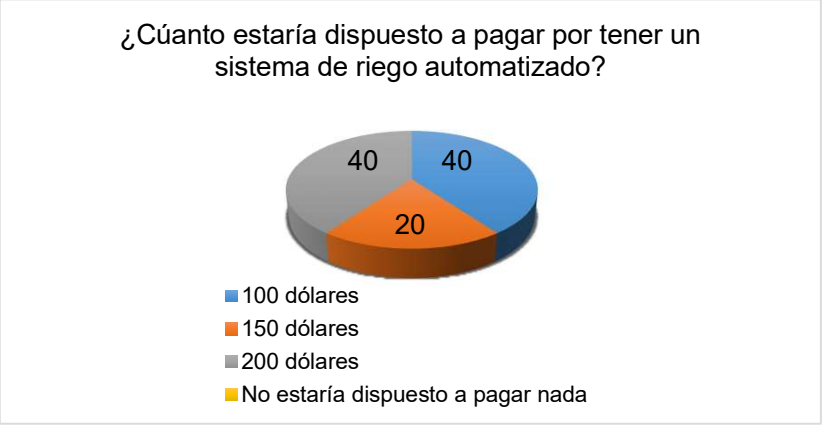



**Figura 2.10.** Resultados de la pregunta 10

### 2.1.1. HISTORIAS DE USUARIO

En base al análisis de las entrevistas del proyecto se han definido las historias de usuario que se muestran en la Tabla 2.1.

**Tabla 2.1.** Historias de Usuario

| ID | Título | Descripción |
|---|---|---|
| HU_01 | Controlar remotamente la bomba de agua. | El cliente requiere controlar el encendido y apagado de la bomba de agua de forma remota en cualquier instante. |
| HU_02 | Control mediante el teléfono celular. | El cliente necesita tener el control del sistema de riego de agua a través de una aplicación móvil instalada en el teléfono celular. |
| HU_03 | Visualizar información del cultivo en tiempo real | El cliente necesita ver los parámetros del cultivo, como la temperatura y la humedad del suelo en tiempo real. |

### 2.1.2. REQUERIMIENTOS

En base a las historias de usuario y a los requerimientos iniciales con los que se diseñó el proyecto, se tienen los siguientes requerimientos para la implementación del prototipo.

1. Controlar el sistema de riego de agua mediante una aplicación móvil en el teléfono móvil.
2. Controlar de forma remota el encendido y apagado de la bomba de agua.
3. Verificar el nivel de agua del pozo.
4. Tener a disposición la información actual de los parámetros del cultivo.

De los requerimientos generales mencionados se desprenden los requerimientos funcionales y no funcionales del proyecto.

#### 2.1.2.1. Requerimientos Funcionales

- La aplicación permita visualizar información de temperatura y humedad del suelo en tiempo real.
- La aplicación posibilite obtener información del nivel de agua en el pozo.
- La aplicación provea al usuario el control del encendido de la bomba de regadío.

#### 2.1.2.2. Requerimientos No Funcionales

- Detrás de las funciones que ofrece la aplicación, se receptarán y transmitirán mensajes a través de la comunicación MQTT.
- El sistema estará disponible para los usuarios de forma indefinida.
- El sistema contará con una conexión a Internet estable y continua.
- El tiempo de respuesta del sistema será ágil y efectivo.
- El sistema presentará una alta fiabilidad.

#### 2.1.2.3. Descripción de Tareas

De los requerimientos ya descritos se desglosan algunas tareas para el cumplimiento de cada uno de dichos requerimientos. Las tareas se presentan a continuación:

1. Crear una conexión MQTT entre un cliente MQTT ubicado en las motas y otro en la Raspberry Pi, mediante un *broker* privado.
2. Diseñar una aplicación móvil en Android con una interfaz amigable con el cliente.
3. Crear una conexión MQTT entre un cliente MQTT ubicado en la Raspberry Pi y otro en la aplicación móvil, mediante un *broker* público.
4. Enviar datos de humedad y temperatura periódicamente desde un cliente MQTT ubicado en las motas hasta un cliente MQTT situado en la aplicación móvil, pasando por el cliente en la Raspberry Pi.

5. Mostrar los datos de humedad y temperatura recibidos por un cliente MQTT en la aplicación móvil.
6. Solicitar información acerca del nivel de agua desde un cliente MQTT en la aplicación móvil hasta otro ubicado en la mota conectada a los sensores de nivel, pasando por el cliente MQTT en la Raspberry Pi.
7. Enviar la información del nivel de agua desde un cliente MQTT ubicado en la mota conectada a los sensores de nivel, hasta un cliente MQTT en la aplicación móvil, a través de un cliente MQTT en la Raspberry Pi.
8. Comandar el encendido o apagado de la bomba de agua desde un cliente MQTT en la aplicación móvil hasta otro situado en la mota conectada a la bomba, mediante un cliente MQTT ubicado en la Raspberry Pi.

## 2.2. TABLERO KANBAN

Cada una de las tareas se colocan en la columna To Do del tablero Kanban, para lo cual se usó la aplicación en línea Kanbatool. El tablero Kanban se muestra en la Figura 2.11.

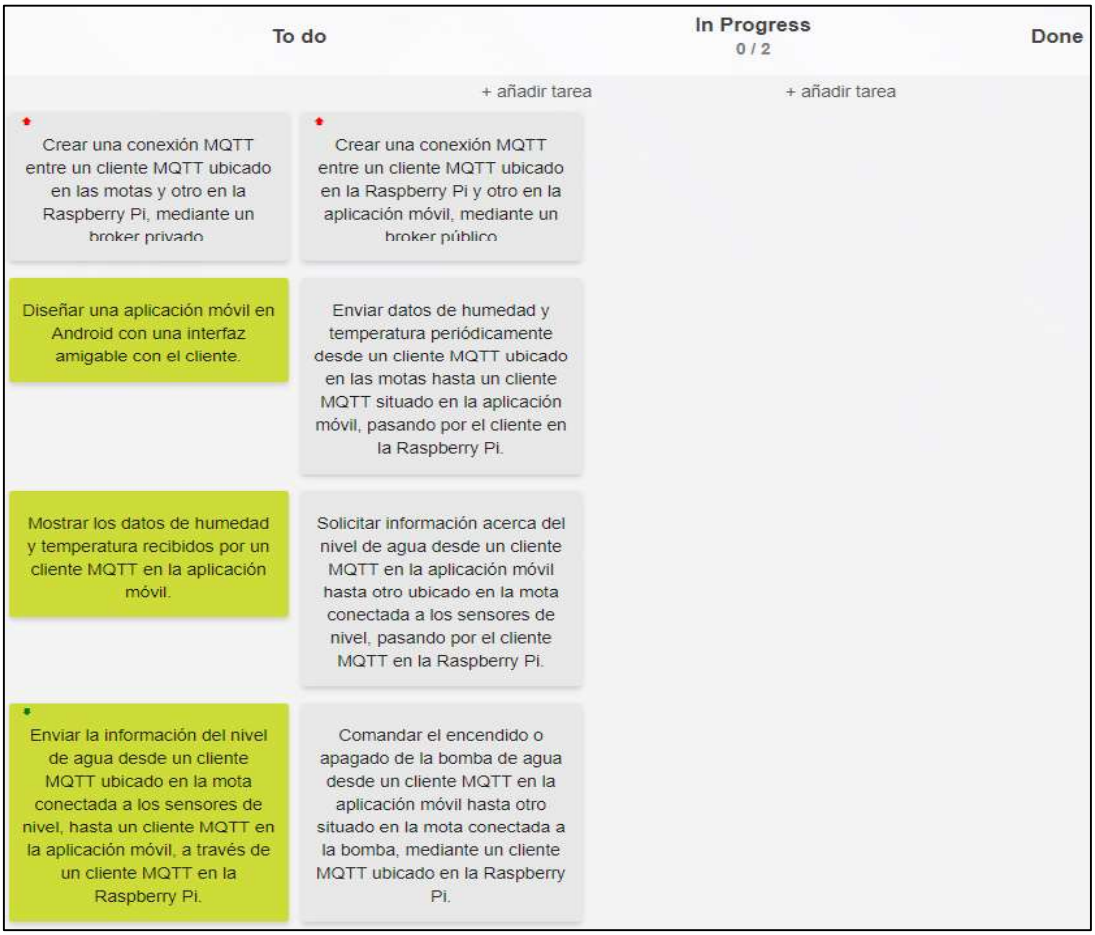



**Figura 2.11.** Tablero Kanban

Las tarjetas Kanban contienen la descripción de las tareas a realizar, en esta fase de inicio las actividades aún no se han ejecutado, por lo cual se encuentran en la primera columna y su ubicación señala el nivel de prioridad; las primeras tarjetas son las de mayor prioridad, por lo que tienen que ser realizadas primero.

## 2.3. DIAGRAMAS UML

### 2.3.1. DIAGRAMA DE CLASES

El diagrama de clases de la aplicación móvil se muestra en la Figura 2.12, la misma que permite observar las clases, sus atributos, métodos y la relación entre ellos. Las clases existentes son ClienteMQTT, BombaAgua y Adaptador; cada una posee atributos de distinto tipo, todos los atributos son privados porque su uso está confinado solamente dentro de la clase donde están definidos.

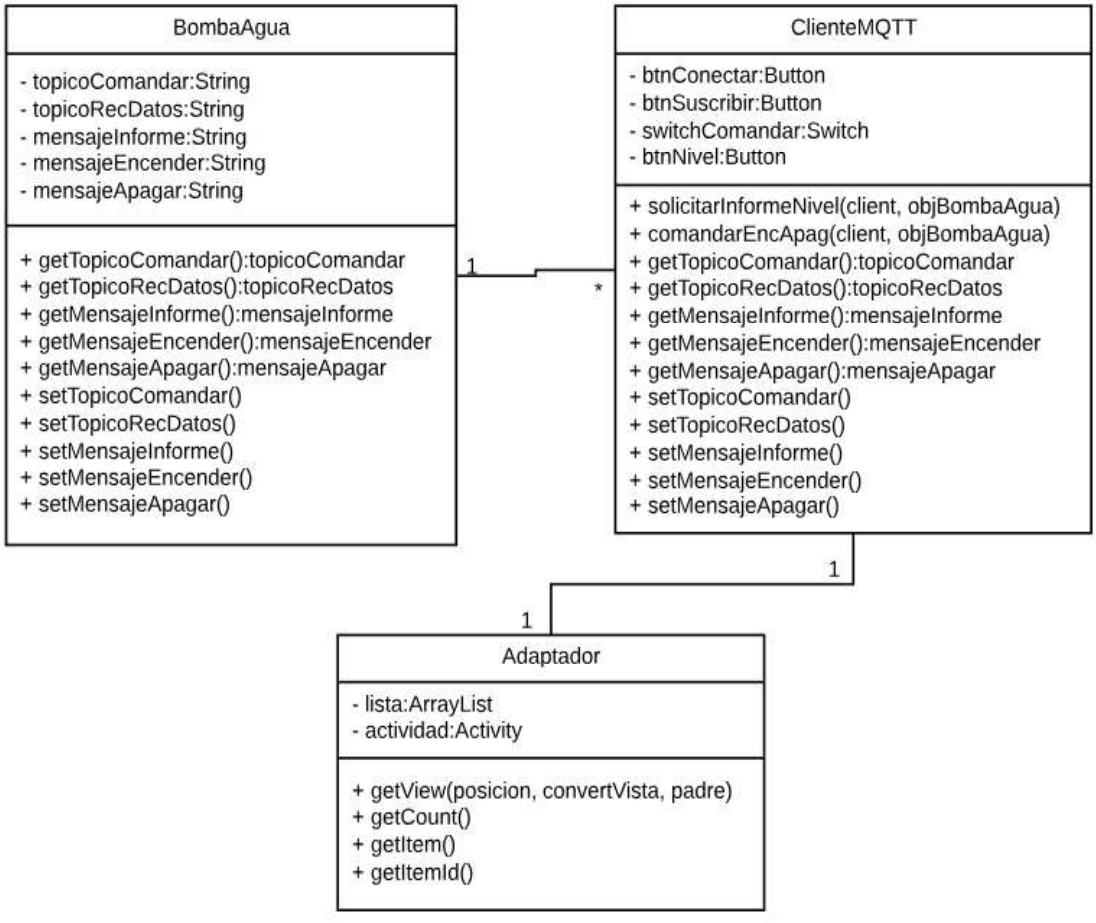



**Figura 2.12.** Diagrama de Clases de la Aplicación Móvil

Los métodos de la clase BombaAgua generalmente son *getters* y *setters* declarados como públicos, para poder acceder a ellos desde cualquier clase. La clase ClienteMQTT

básicamente se compone de métodos que permiten solicitar información del nivel de agua y comandar el encendido y apagado de la bomba de regadío; todas estas actividades se realizan mediante controles como botones y *switches*. La clase Adaptador permite al ClienteMQTT mostrar los mensajes que llegan en un tópico determinado desde el *broker* público Mosquitto en un control *ListView* de forma organizada.

### 2.3.2. DIAGRAMA DE ACTIVIDADES

El diagrama de actividades del usuario con la aplicación móvil se presenta en la Figura 2.13. Este diagrama permite visualizar de forma secuencial los procesos que el cliente ejecuta dentro de la aplicación móvil.

El cliente al ingresar a la aplicación se conecta al *broker* público y luego se suscribe a un tópico para obtener los mensajes que le lleguen desde la red de nodos. Estas dos actividades se ejecutan al instante que el usuario ingresa a la aplicación, sin necesidad de presionar algún control de la aplicación; luego el cliente recibirá los mensajes de forma visual en un control de la aplicación. El cliente tiene la posibilidad de solicitar conocer el nivel de agua del pozo al presionar un control en la aplicación móvil y recibirá la información requerida. Por último, si el cliente considera necesario accionará la bomba de agua mediante un control de la aplicación.

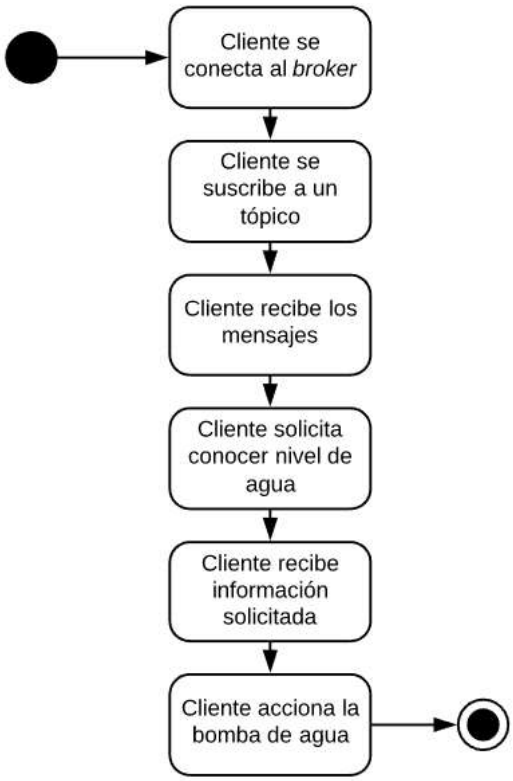



**Figura 2.13.** Diagrama de actividades

### 2.3.3. DIAGRAMAS DE FLUJO

El diagrama de flujo de la Figura 2.14. describe de forma estructurada el código que se ejecuta en la plataforma Zolertia RE-Mote, con el fin de sensar temperatura y humedad, para luego publicar los datos a través de un tópico hacia el *broker* privado ubicado en la Raspberry Pi.

En el principio del flujo, se declaran algunas variables que se utilizarán a lo largo del código, luego se activa y configura el sensor de humedad y temperatura para que pueda sensar los parámetros que se requieran. Después se crea un cliente MQTT, para el cual se construye un ID del cliente y también se construye el tópico para publicación. Luego el código entra en una estructura *switch* donde se evalúa el valor de la variable *estado*, y se inicia con el caso *STATE_INIT*, pasando por *STATE_REGISTERED*, *STATE_PUBLISHING* y *STATE_DISCONNECTED*. Mediante estos casos, el cliente MQTT se conectará al *broker*, obtendrá los datos sensados, publicará los datos sensados y se desconectará del *broker*.

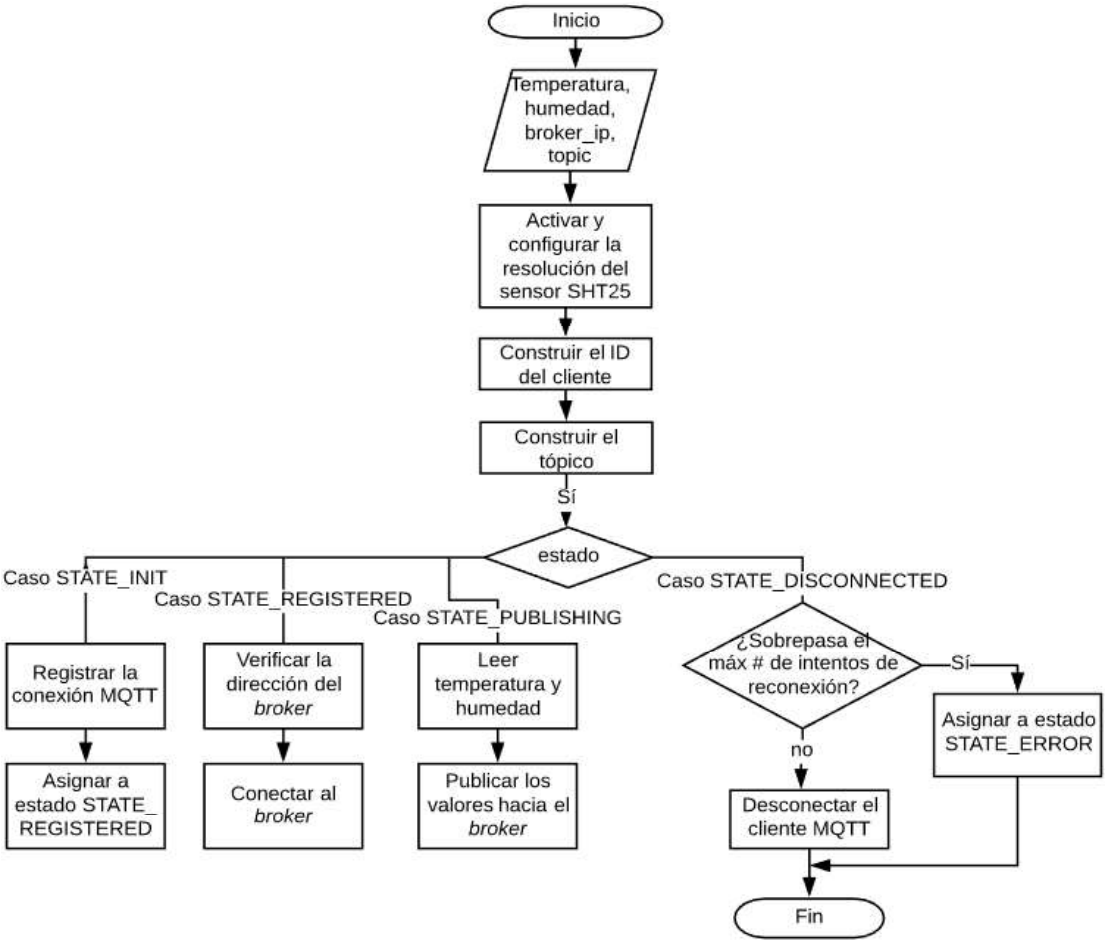



**Figura 2.14.** Diagrama de Flujo del código ejecutado en la mota con el sensor SHT25

El diagrama de flujo de la Figura 2.15., representa al código que permite la conexión entre un cliente MQTT ubicado en la mota con el sensor y otro cliente en la aplicación móvil. Este código se ejecuta en la Raspberry Pi, y está escrito en lenguaje C.

Al principio de este diagrama de flujo, se declaran variables importantes para crear dos clientes MQTT y para que puedan comunicarse con otros clientes. Los clientes se conectan a dos *brokers*, uno es privado y está instalado en la misma Raspberry Pi, mientras que el otro es público y se accede a él mediante la dirección URL *iot.eclipse.org*. La razón por la cual estos clientes se conectan a dos *brokers*, es que actúan como intermediarios entre la red de nodos y la aplicación móvil; entonces uno de los clientes recibe los mensajes desde el *broker* privado, esto es, los mensajes de la red de nodos y el otro cliente publica dichos mensajes hacia el *broker* público, esto es, hacia la aplicación móvil.

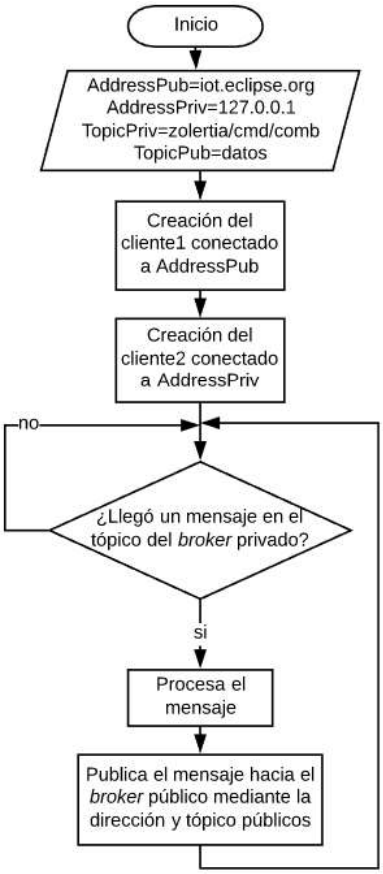



**Figura 2.15.** Diagrama de Flujo del código ejecutado en la Raspberry Pi

Del mismo modo, existe otro código similar en el que se crean otros dos clientes MQTT que, en lugar de recibir mensajes desde la red de nodos, los reciben desde la aplicación móvil para publicarlos a la red de nodos. A través de estos cuatro clientes MQTT ubicados en la Raspberry Pi existe comunicación total entre la red de nodos y la aplicación móvil.

## 2.4. DISEÑO DE SOFTWARE

Se parte desde el desarrollo de software que permite la comunicación de los nodos con el servidor público. El código de los clientes MQTT creados tanto en la red de nodos como en la Raspberry Pi y en la aplicación móvil ha sido escrito basándose en el código fuente que provee Eclipse Paho.

### 2.4.1. NODOS SENSORES

En primera instancia, el sistema operativo Contiki es instalado en el software VMWare, en Ubuntu Linux. Luego de la instalación, esta máquina virtual posee todas las herramientas de desarrollo para la programación de la red de nodos de sensores.

Para la codificación de los sensores se utilizan varios archivos adicionales al programa principal, los mismos que permiten la ejecución del programa principal; uno de ellos es el archivo *project-conf.h*, el cual contiene varios valores de configuración a usarse en el programa principal; el código de este archivo se observa en el Código 2.1.

```
/*---------------------------------------------------------------------------*/
#ifndef PROJECT_CONF_H_
#define PROJECT_CONF_H_
/*---------------------------------------------------------------------------*/
/* User configuration */
#define DEFAULT_ORG_ID            "mqtt-demo"
#define MQTT_DEMO_BROKER_IP_ADDR  "aaaa::1/64"
/*---------------------------------------------------------------------------*/
/* Valores de configuración por defecto */
#define DEFAULT_EVENT_TYPE_ID        "datos_sensor"
#define DEFAULT_SUBSCRIBE_CMD_TYPE   "bomb"
#define DEFAULT_BROKER_PORT          1883
#define DEFAULT_PUBLISH_INTERVAL     (5 * CLOCK_SECOND)
#define DEFAULT_KEEP_ALIVE_TIMER     60

#undef IEEE802154_CONF_PANID
#define IEEE802154_CONF_PANID        0xABCD

#undef CC2538_RF_CONF_CHANNEL
#define CC2538_RF_CONF_CHANNEL       26

#define BUFFER_SIZE                  64
#define APP_BUFFER_SIZE              512
/*---------------------------------------------------------------------------*/
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC          nullrdc_driver

/* Tamaño máximo del segmento TCP para segmentos que salen del socket */
#define MAX_TCP_SEGMENT_SIZE       32

#endif /* PROJECT_CONF_H_ */
/*---------------------------------------------------------------------------*/
/** @} */
```

**Código 2.1.** Archivo *project-conf.h*

Entre los parámetros de este archivo está la dirección IPv6 del *broker* que es definida aquí mediante la variable `MQTT_DEMO_BROKER_IP_ADDR`, esta dirección es la que se utiliza para conectarse con el *broker* privado Mosquitto, instalado en la Raspberry Pi. También se definen los tópicos, el primero se carga en la variable `DEFAULT_EVENT_TYPE_ID`, el cual es usado para la transmisión de información desde los sensores hasta el *broker* privado,

mientras que mediante la variable DEFAULT_SUBSCRIBE_CMD_TYPE se permitirá comandar el encendido o apagado de la bomba de regadío. El número de puerto para utilizar MQTT es 1883, el cual se define a través de la variable DEFAULT_BROKER_PORT y el canal por defecto que usa la plataforma es el 26 definido por la variable CC2538_RF_CONF_CHANNEL; además, en este archivo se define el tamaño de los *buffers*.

El siguiente archivo se denomina *Makefile*, éste es necesario para que la aplicación se pueda compilar, el contenido de este archivo se puede apreciar en el Código 2.2. La palabra *all*, especifica la aplicación que debe compilar el sistema, es decir, que el programa principal se llama mqtt-sensores; luego se indica el nombre del archivo fuente que contiene la implementación del sensor SHT25, a través de la variable CONTIKI_TARGET_SOURCEFILES, librería que es añadida al programa principal para adquirir los datos del sensor. También se añade el *driver* de la aplicación mqtt, del cual se usan las funciones para realizar la comunicación entre las motas y la aplicación móvil.

Mediante la variable CONTIKI_WITH_IPV6 se especifica que se trabaja con el protocolo IPv6 entre las motas; luego se indica el nivel de identación respecto a la carpeta raíz Contiki; finalmente este archivo apunta al archivo *Makefile* Contiki de todo el sistema, mediante la directiva include.

```
DEFINES+=PROJECT_CONF_H=\"project-conf.h\"

all: mqtt-sensores

CONTIKI_TARGET_SOURCEFILES += sht25.c
APPS += mqtt
SMALL = 1

CONTIKI_WITH_IPV6 = 1

CONTIKI = ../../../..

include $(CONTIKI)/Makefile.include
```

**Código 2.2.** Archivo *Makefile*

El programa principal se puede apreciar en el Código 2.3., este código está basado en el ejemplo *mqtt-example.c* que provee Contiki, el cual se ejecuta mediante un proceso denominado proceso_mqtt, donde dentro del hilo se inicia el proceso a través de la macro PROCESS_BEGIN().

Luego se activa el sensor SHT25 mediante la macro SENSORS_ACTIVATE(sht25); dado que este sensor puede trabajar con dos tipos de resoluciones, es necesario configurarla, en este caso se utilizará la resolución de 14 y 12 bits para medir la temperatura y la humedad respectivamente, esto a través de la función sht25.configure(). Después se

llama a la función `update_config()`, la cual a su vez llama a otras funciones para construir el identificador del cliente MQTT y el tópico para publicar datos. Luego se llama a la función `state_machine()`.

```
PROCESS_THREAD(proceso_mqtt, ev, data)
{

  PROCESS_BEGIN();

  printf("Proceso MQTT \n");

  if(init_config() != 1) {
    PROCESS_EXIT();
  }

  SENSORS_ACTIVATE(sht25);
  sht25.configure(SHT25_RESOLUTION, SHT2X_RES_14T_12RH);
  update_config();

  while(1) {
    PROCESS_YIELD();

    if((ev == PROCESS_EVENT_TIMER && data == &publish_periodic_timer) ||
       ev == PROCESS_EVENT_POLL) {
      state_machine();
    }
  }

  PROCESS_END();
}
```

**Código 2.3.** Programa principal

El programa principal *mqtt-sensores.c* consta básicamente del proceso principal y de varias funciones. En el inicio se declara la estructura para la configuración del cliente MQTT, esta estructura se puede observar en la Figura 2.16. Consta de diferentes tipos de datos, entre los cuales están el nombre del tópico en el que se publicarán los datos, representado por `event_type_id`, luego la dirección del *broker* identificado como `broker_ip`, y el número del puerto para la conexión MQTT. Mediante esta estructura se podrá registrar una conexión MQTT más adelante.

```
typedef struct mqtt_client_config {
  char event_type_id[CONFIG_EVENT_TYPE_ID_LEN];
  char broker_ip[CONFIG_IP_ADDR_STR_LEN];
  clock_time_t pub_interval;
  uint16_t broker_port;
} mqtt_client_config_t;
```

```
static int
init_config()
{
  /* Llenar la configuración con los valores por defecto */
  memset(&conf, 0, sizeof(mqtt_client_config_t));
  memcpy(conf.event_type_id, DEFAULT_EVENT_TYPE_ID,
         strlen(DEFAULT_EVENT_TYPE_ID));
  memcpy(conf.broker_ip, broker_ip, strlen(broker_ip));

  conf.broker_port = DEFAULT_BROKER_PORT;
  conf.pub_interval = DEFAULT_PUBLISH_INTERVAL;

  return 1;
}
```

**Figura 2.16.** Declaración y asignación de la estructura del cliente MQTT

La asignación de los valores por defecto en la estructura del cliente MQTT se realiza mediante la función `init_config()`. La instancia de la estructura es llamada `conf`, y los valores se cargan en la estructura mediante la función `memcpy()`; los valores están definidos previamente en el archivo *project-conf.h*,

Se construye el nombre del tópico para la publicación de los datos mediante la función `construct_pub_topic()`, y además, se construye el identificador del cliente con la dirección MAC de la mota Zolertia RE-Mote. Ambas funciones se pueden observar en el Código 2.4., donde la función `construct_pub_topic()` construye el nombre del tópico en el que se publicarán los datos de los sensores; el tópico se asigna a la variable `pub_topic` mediante la función `snprintf()`, el nombre del tópico es *zolertia/datos_sensor.*

La función `construct_client_id()` permite crear el identificador del cliente MQTT, que lo identificará al momento de la conexión, este identificador se basa en la dirección MAC de la mota, la misma que se obtiene de la variable `linkaddr_node_addr`; este identificador es almacenado en la variable `client_id`.

```
static int
construct_pub_topic(void)
{
  int len = snprintf(pub_topic, BUFFER_SIZE, "zolertia/%s", conf.event_type_id);
  if(len < 0 || len >= BUFFER_SIZE) {
    printf("Pub Topic too large: %d, Buffer %d\n", len, BUFFER_SIZE);
    return 0;
  }

  return 1;
}

/*-----------------------------------------------------------------------------*/
/* Crea el identificador del nodo cliente con la direccion mac*/
static int
construct_client_id(void)
{
  int len = snprintf(client_id, BUFFER_SIZE, "d:%02x%02x%02x%02x%02x%02x",
                     linkaddr_node_addr.u8[0], linkaddr_node_addr.u8[1],
                     linkaddr_node_addr.u8[2], linkaddr_node_addr.u8[5],
                     linkaddr_node_addr.u8[6], linkaddr_node_addr.u8[7]);

  /* len < 0: Error. Len >= BUFFER_SIZE: Buffer demasiado pequeño */
  if(len < 0 || len >= BUFFER_SIZE) {
    printf("Client ID: %d, Buffer %d\n", len, BUFFER_SIZE);
    return 0;
  }

  return 1;
}
```

**Código 2.4.** Funciones `construct_pub_topic()` y `construct_client_id()`

La función `state_machine()` básicamente está codificada como una máquina de estados finita, es decir, pasa a través de varios estados como `STATE_INIT`, `STATE_REGISTERED`, `STATE_CONNECTING`, `STATE_CONNECTED`, `STATE_PUBLISHING`, etc. Un fragmento de esta función se aprecia en el Código 2.5.

```
static void state_machine(void)
{
  switch(state) {
  case STATE_INIT:
    /* Configura el registro de conexión MQTT */
    mqtt_register(&conn, &proceso_mqtt, client_id, mqtt_event,
                  MAX_TCP_SEGMENT_SIZE);

    conn.auto_reconnect = 0;
    connect_attempt = 1;

    state = STATE_REGISTERED;
    printf("Init\n");

    /* No hay break para continuar con el estado REGISTERED */
  case STATE_REGISTERED:
    if(uip_ds6_get_global(ADDR_PREFERRED) != NULL) {
      /* Registrado y con una IP pública. Connect */
      printf("Registered. Connect attempt %u\n", connect_attempt);
      connect_to_broker();

    } else {
      leds_on(LEDS_GREEN);
      ctimer_set(&ct, NO_NET_LED_DURATION, publish_led_off, NULL);
    }
    etimer_set(&publish_periodic_timer, NET_CONNECT_PERIODIC);
    return;
    break;

  case STATE_CONNECTING:
    leds_on(LEDS_GREEN);
    ctimer_set(&ct, CONNECTING_LED_DURATION, publish_led_off, NULL);
    /* Aun no conectado. Espere */
    printf("Connecting (%u)\n", connect_attempt);
    break;
```

**Código 2.5.** Fragmento de la función `state_machine`

En el caso `STATE_INIT` primero se registra la conexión MQTT mediante la macro `mqtt_register()`, y cambia el estado a `STATE_REGISTERED`, dentro del cual se verifica la validez de la dirección IPv6 del *broker* privado y se conecta al mismo; luego se pasa al estado `STATE_CONNECTING` y a continuación a `STATE_PUBLISHING` donde se llama a la función `publish()`.

La función `publish()` se muestra en el Código 2.6., donde la primera información que se publica es el identificador de la mota, representado por los últimos 4 números de la dirección MAC. Luego se toman los valores sensados del sensor SHT25, a través de las variables temperatura y humedad, y se llama a las funciones de la librería *sht25.c*, funciones que reciben los parámetros `SHT25_VAL_TEMP` y `SHT25_VAL_HUM`. Finalmente, estos valores son publicados en el tópico correspondiente hacia el *broker* privado Mosquitto mediante la función `mqtt_publish()`, esta función recibe como parámetros: `conn` que es la conexión MQTT, `pub_topic` el tópico en el que se publican los datos, `app_buffer` el *buffer* de datos, `MQTT_QOS_LEVEL_0` el nivel de calidad de servicio de MQTT y `MQTT_RETAIN_FLAG` la bandera Retain.

Todos los valores que se quieren publicar se encuentran en el *buffer* `app_buffer` y cada vez que se cargan datos al *buffer* se verifica que su tamaño de almacenamiento sea adecuado, de lo contrario se muestra un mensaje de advertencia; el nivel de calidad de

servicio utilizado es 0 porque los datos se envían periódicamente y una entrega del mejor esfuerzo es adecuada para este tipo de transmisión.

```
static void publish(void)
{
  int len;
  int remaining = APP_BUFFER_SIZE;

  seq_nr_value++;
  buf_ptr = app_buffer;

  len = snprintf(buf_ptr, remaining, "%02x%02x",
                   linkaddr_node_addr.u8[6], linkaddr_node_addr.u8[7]);

  if(len < 0 || len >= remaining) {
    printf("Buffer too short. Have %d, need %d + \\0\n", remaining, len);
    return;
  }

  remaining -= len;
  buf_ptr += len;

  int16_t temperatura, humedad;
  temperatura = sht25.value(SHT25_VAL_TEMP);
  humedad = sht25.value(SHT25_VAL_HUM);
  len = snprintf(buf_ptr, remaining, ",\%02d.%02d\,\%02d.%02d\                ",
temperatura / 100, temperatura % 100, humedad / 100, humedad % 100);

  remaining -= len;
  buf_ptr += len;

  if(len < 0 || len >= remaining) {
    printf("Buffer too short. Have %d, need %d + \\0\n", remaining, len);
    return;
  }

  remaining -= len;
  buf_ptr += len;

  mqtt_publish(&conn, NULL, pub_topic, (uint8_t *)app_buffer,
               strlen(app_buffer), MQTT_QOS_LEVEL_0, MQTT_RETAIN_OFF);

  printf("APP - Publish to %s: %s\n", pub_topic, app_buffer);
}
```

**Código 2.6.** Función `publish()`

Luego de publicar los valores, el estado pasa a `STATE_DISCONNECTED`, donde se cuestiona si los intentos de conexión ya han sido sobrepasados, si esto es verdad se determina un estado de error o se desconecta el cliente MQTT.

### 2.4.2. NODO ACTUADOR

El nodo actuador está conectado a la bomba de regadío mediante un relé y además está conectado a sensores de nivel de agua, los cuales se encuentran ubicados en el pozo con el fin de indicar el nivel de agua; esta información es enviada a la aplicación móvil, de modo que el usuario pueda decidir si es prudente encender o no la bomba de agua.

La codificación del programa para este nodo es similar a la de los nodos sensores, con algunas diferencias. El programa principal se muestra en el Código 2.7., aquí inicia el proceso, y se configuran tres pines de la plataforma Zolertia RE-Mote para controlarlos vía software con el número de pin y el puerto; dicha configuración se la realiza mediante la

macro `GPIO_SOFTWARE_CONTROL`. Dos de los pines configurados permitirán tomar los datos del nivel de agua, mediante los sensores conectados y el otro pin servirá para comandar el encendido de la bomba de regadío. Los dos primeros pines son configurados como entradas a través de la macro `GPIO_SET_INPUT()` para obtener los valores sensados, y el tercer pin se configura como pin de salida mediante la macro `GPIO_SET_OUTPUT()`. Luego de esta configuración el programa entra en un bucle `while()`, donde empieza a ejecutar la función `state_machine()`, la misma que pasa a través de varios estados mediante la estructura `switch()`; esta función es semejante a la de los nodos sensores.

```
PROCESS_THREAD(test_sensor_agua, ev, data)
{

  PROCESS_BEGIN();

  printf("Proceso MQTT \n");

  if(init_config() != 1) {
    PROCESS_EXIT();
  }
  GPIO_SOFTWARE_CONTROL(PORT, PIN);
  GPIO_SOFTWARE_CONTROL(PORT2, PIN2);
  GPIO_SOFTWARE_CONTROL(PORT3, PIN3);
  GPIO_SET_INPUT(PORT, PIN);
  GPIO_SET_INPUT(PORT2, PIN2);
  GPIO_SET_OUTPUT(PORT3, PIN3);

  update_config();

  while(1) {
    PROCESS_YIELD();

    if((ev == PROCESS_EVENT_TIMER && data == &publish_periodic_timer) ||
       ev == PROCESS_EVENT_POLL) {
      state_machine();
    }
  }

  PROCESS_END();
}
```

**Código 2.7.** Proceso principal del nodo actuador

En este programa el cliente MQTT se suscribe al tópico `zolertia/cmd/bomb`, mediante la función `mqtt_subscribe()` de la librería MQTT, donde el tópico es almacenado en la variable `sub_topic`. Esta función se observa en el Código 2.8.

```
static void subscribe(void)
{
  mqtt_status_t status;

  status = mqtt_subscribe(&conn, NULL, sub_topic, MQTT_QOS_LEVEL_0);

  printf(" Subscribing to %s\n", sub_topic);
  if(status == MQTT_STATUS_OUT_QUEUE_FULL) {
    printf(" Tried to subscribe but command queue was full!\n");
  }
}
```

**Código 2.8.** Función `subscribe()`

Si recibe mensajes desde el tópico al que se suscribió entonces la función `pub_handler()` es llamada para manejar el mensaje recibido. En el Código 2.9. se observa un fragmento de esta función, donde primero verifica que el nombre del tópico sea el correcto, luego recibe el mensaje y lo procesa de la siguiente manera: si el mensaje recibido es 0, entonces mediante el comando `GPIO_CLR_PIN()` se enviará como salida un nivel bajo en el pin y puerto 3, lo que equivaldrá a apagar la bomba de regadío; si el mensaje recibido es 1, entonces mediante la función `GPIO_SET_PIN()`, se enviará como salida un valor alto en el pin y puerto 3, lo que significará el encendido de la bomba; por último si el mensaje recibido es igual a 2, entonces se enviará el valor del nivel de agua a través del tópico *zolertia/datos_sensor* a un cliente MQTT en la Raspberry Pi.

```
if(strncmp(&topic[13], "bomb", 4) == 0) {
if(chunk[0] == '0') {
  GPIO_CLR_PIN(PORT3, PIN3);
  printf("Turning bomb off!\n");
} else if(chunk[0] == '1') {
  GPIO_SET_PIN(PORT3, PIN3);
  printf("Turning bomb on!\n");
}else if(chunk[0] == '2') {
  len = snprintf(buf_ptr, remaining, "%02x%02x",
                 linkaddr_node_addr.u8[6], linkaddr_node_addr.u8[7]);

  if(len < 0 || len >= remaining) {
      printf("Buffer too short. Have %d, need %d + \\0\n", remaining, len);
    return;
  }

  remaining -= len;
  buf_ptr += len;

  uint16_t valor, valor2;
  valor = GPIO_READ_PIN(PORT, PIN);
  valor2 = GPIO_READ_PIN(PORT2, PIN2);
  if(valor == 0 && valor2 == 0){
      len = snprintf(buf_ptr, remaining, ",            \"Escaso      ");
  }
  if(valor > 0 && valor2 > 0){
      len = snprintf(buf_ptr, remaining, ",            \"Suficiente  ");
  }
  if(valor > 0 && valor2 == 0 ){
    len = snprintf(buf_ptr, remaining, ",           \"Insuficiente");
  }

  remaining -= len;
  buf_ptr += len;

  if(len < 0 || len >= remaining) {
    printf("Buffer too short. Have %d, need %d + \\0\n", remaining, len);
  return;
  }

  mqtt_publish(&conn, NULL, pub_topic, (uint8_t *)app_buffer,
              strlen(app_buffer), MQTT_QOS_LEVEL_0, MQTT_RETAIN_OFF);
```

**Código 2.9.** Fragmento de la función `pub_handler()`

Cuando el dato recibido desde la aplicación es 2, quiere decir que el usuario ha solicitado conocer el nivel de agua del pozo. En este programa se analizan los datos que envían dos sensores de nivel de agua discretos y la información es clasificada como se muestra en la Tabla 2.2.

Luego de hacer esta clasificación, se publican estos mensajes a través del *broker* privado Mosquitto, para su posterior envío al *broker* público

**Tabla 2.2.** Interpretación de niveles de agua en el pozo

| Sensor 1 | Sensor 2 | Nivel |
|---|---|---|
| 0 | 0 | No hay agua. Mensaje: Escaso |
| 1 | 1 | Existe suficiente agua. Mensaje: Suficiente |
| 0 | 1 | No hay suficiente agua. Mensaje: Insuficiente |

### 2.4.3. NODO *ROUTER* DE BORDE

La mota conectada a la Raspberry Pi actúa como un *router* de borde para permitir la conexión entre la red de nodos de sensores y la aplicación móvil, a través de dos *broker*s, uno privado instalado en la Raspberry y otro público disponible en la nube.

Este nodo *router* de borde se encuentra en el borde de la red y el programa que se carga tiene el nombre de *border-router.c*, programa que se encuentra ubicado en la dirección */contiki/examples/ipv6/rpl-border-router*.

Al ejecutar este código se implementa una interfaz de red tunelizada mediante la conexión serial entre la mota y la Raspberry Pi, y usa el prefijo IPv6 por defecto *aaaa::1/64*, prefijo que es declarado en el archivo *project-conf.h* de la aplicación ejecutada en los nodos de sensores.

Este *router* de borde usa el mecanismo de autoconfiguración SLAC (*Stateless Auto Configuration*) para crear su dirección IPv6 usando un prefijo IPv6 predeterminado. Luego de la ejecución de este programa se crea una interfaz en la Raspberry Pi denominada *tun0*, desde la cual existe conexión al *router* de borde; la funcionalidad de esta interfaz se muestra en la Figura 2.17., donde se puede verificar que utiliza el prefijo IPv6 predeterminado.

```
tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 127.0.1.1  netmask 255.255.255.255  destination 127.0.1.1
        inet6 aaaa::1  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::ea3f:72b6:fce0:7b05  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::1  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 1507  bytes 124399 (121.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1527  bytes 91832 (89.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

**Figura 2.17.** Interfaz tun0

### 2.4.4. *BROKER* PRIVADO

El *broker* privado permite la comunicación entre la red de nodos de sensores hacia un cliente MQTT ubicado en la Raspberry Pi.

En la Raspberry Pi se instala el *broker* privado Mosquitto, que es un servidor de mensajería del protocolo MQTT, es ligero y es de código abierto, que se lo descarga desde los repositorios principales de Raspberry Pi; todo el proceso de instalación se desarrolla mediante comandos.

Para abrir la aplicación de Mosquitto se ejecuta el comando `mosquitto` en la línea de comandos y ésta iniciará con la configuración por defecto y quedará a la espera de conexiones y recepción de mensajes mediante el puerto 1883, como se aprecia en la Figura 2.18.; la versión de Mosquitto que se utiliza es la 1.4.5.

```
root@raspberrypi:/home/pi# mosquitto
1558558609: mosquitto version 1.5.4 starting
1558558609: Using default config.
1558558609: Opening ipv4 listen socket on port 1883.
1558558609: Opening ipv6 listen socket on port 1883.
```

**Figura 2.18.** Broker privado Mosquitto

La dirección IP del *broker* privado Mosquitto, que se utilizará para poder enviar los mensajes entre diferentes clientes a través de tópicos, será la 127.0.0.1.

### 2.4.5. *BROKER* PÚBLICO

El *broker* público se utiliza para la comunicación de la aplicación móvil y el cliente MQTT en la Raspberry Pi. De esta forma, los mensajes que lleguen desde la red de nodos de sensores hasta dicho cliente serán reenviados a la aplicación móvil vía el *broker* público; de la misma manera los mensajes que lleguen desde la aplicación móvil serán transmitidos hacia el cliente MQTT.

El *broker* público que se utiliza es Mosquitto y el *hostname* que se usa para la conexión es *iot.eclipse.org*, que es accesible mediante el puerto 1883. Para usar este *broker* no es necesario instalarlo, ya que está disponible en la nube, lo único que se debe hacer es, colocar el *hostname* indicado en la dirección de *broker* desde el cliente MQTT que se quiere conectar; este *broker* aceptará la conexión y posteriormente si le llegan mensajes los reenviará en el tópico que corresponda.

### 2.4.6. CLIENTES EN LA RASPBERRY PI

Se han codificado 2 *script*s en lenguaje C para la conexión bidireccional entre la aplicación móvil y la red de nodos de sensores. El primer *script* permite la comunicación desde la red

de nodos de sensores hacia la aplicación móvil, y el segundo *script* la comunicación desde la aplicación móvil hacia la red de nodos. Los *script*s son similares debido a que ambos reciben los mensajes desde un *broker* y tópico, para enviarlos a otro *broker* y tópico; la diferencia entre ellos radica básicamente en las direcciones IP de los *broker*s y los nombres de los tópicos.

Para crear el cliente MQTT, es necesario primero descargar Paho C de los repositorios de Eclipse e instalarlo en la Raspberry Pi; luego de la instalación, se tiene a disposición varios archivos en el *host* de la Raspberry Pi, los cuales se usan para la codificación, compilación y posterior ejecución de los *script*s.

El primer *script* se compone de llamadas a librerías, entre ellas `MQTTClient.h`, que permitirá usar algunas funciones para crear el cliente MQTT; también se definen algunas variables imprescindibles para realizar la conexión. Las llamadas a las librerías y las definiciones se pueden apreciar en el Código 2.10.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "MQTTClient.h"

#define ADDRESSPUB      "iot.eclipse.org:1883"
#define ADDRESSPRIV     "127.0.0.1:1883"
#define CLIENTID1       "Client1"
#define CLIENTID2       "Client2"
#define TOPICPRIV       "zolertia/cmd/bomb"
#define TOPICPUB        "command"
#define PAYLOAD         ""
#define QOS             1
#define TIMEOUT         10000L
```

**Código 2.10.** Definición de variables del *script* 1

Entre las diferentes variables que se definen, se encuentran la dirección de los *brokers* público y privado seguido del puerto que utilizan; también se especifican los identificadores de los clientes, los tópicos tanto del privado como del público y el nivel de calidad de servicio de MQTT con el que se trabaja. En el tópico `command` se enviarán los mensajes desde la aplicación móvil hasta el cliente MQTT `Client1` y en el tópico `zolertia/cmd/bomb` se transmitirán los mensajes desde el cliente MQTT `Client2` hacia la red de nodos.

Este *script* permitirá enviar el comando de accionamiento de la bomba de agua desde la aplicación móvil hasta la mota. El programa principal que se muestra en el Código 2.11. crea dos clientes, con distintos identificadores, uno está conectado al *broker* privado y el otro al público. El cliente `Client1` se suscribe al tópico `command` esperando por mensajes que lleguen desde la aplicación móvil; al final este cliente termina la suscripción, se desconecta y destruye el cliente.

```
int main(int argc, char* argv[])
{
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    int rc1, ch;

    MQTTClient_create(&cliente, ADDRESSPUB, CLIENTID1, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    MQTTClient_create(&cliente2, ADDRESSPRIV, CLIENTID2, MQTTCLIENT_PERSISTENCE_NONE, NULL)
    conn_opts.keepAliveInterval = 20;
    conn_opts.cleansession = 1;

    MQTTClient_setCallbacks(cliente, NULL, connlost, msgarrvd, delivered);

    if ((rc1 = MQTTClient_connect(cliente, &conn_opts)) != MQTTCLIENT_SUCCESS)
    {
        printf("Failed to connect, return code %d\n", rc1);
        exit(EXIT_FAILURE);
    }
    if ((rc1 = MQTTClient_connect(cliente2, &conn_opts)) != MQTTCLIENT_SUCCESS)
    {
        printf("Failed to connect, return code %d\n", rc1);
        exit(EXIT_FAILURE);
    }

    printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"
           "Press Q<Enter> to quit\n\n", TOPICPUB, CLIENTID1, QOS);
    MQTTClient_subscribe(cliente, TOPICPUB, QOS);

    do{
        ch = getchar();
    } while(ch!='Q' && ch != 'q');

    MQTTClient_unsubscribe(cliente, TOPICPUB);
    MQTTClient_disconnect(cliente, 10000);
    MQTTClient_destroy(&cliente);
    return rc1;
}
```

**Código 2.11.** Programa principal del *script* 1

Dentro del programa principal se hace una llamada a la función `MQTTClient_setCallbacks()`, la misma que a su vez llama a la función `msgarrvd()` que se encarga del manejo de los mensajes que llegan del *broker* público, es decir, desde la aplicación móvil, en el tópico `command`. Esta función se puede visualizar en el Código 2.12., en la cual se recibe el mensaje y se almacena en la cadena `payloadptr`, luego se imprime este mensaje e inmediatamente el cliente `Client2` lo publica en el tópico `zolertia/cmd/bomb` mediante el *broker* privado Mosquitto. Luego de haber publicado el mensaje se libera el mensaje y también el tópico.

```
int msgarrvd(void *context1, char *topicName1, int topicLen1, MQTTClient_message *message1)
{
    int i;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName1);
    printf("   message: ");

    payloadptr = message1->payload;
    for(i=0; i<message1->payloadlen; i++)
    {
        putchar(*payloadptr++);
    }
    MQTTClient_publish(cliente2, TOPICPRIV, message1->payloadlen, message1->payload, QOS, 1,
&token1);
    putchar('\n');
    MQTTClient_freeMessage(&message1);
    MQTTClient_free(topicName1);
    return 1;
}
```

**Código 2.12.** Función `msgarrvd` del *script* 1

El siguiente *script* tiene la misma estructura que el anterior, sin embargo, algunos parámetros cambian, los cuales se observan en el Código 2.13., en la definición de las variables globales del *script*. Los identificadores de los clientes son `Client3` y `Client4`, el tópico para recibir mensajes de la red de nodos es `zolertia/datos_sensor` y el tópico para enviar los datos a la aplicación móvil es `datos`; las demás variables son las mismas.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "MQTTClient.h"

#define ADDRESSPUB    "iot.eclipse.org:1883"
#define ADDRESSPRIV   "127.0.0.1:1883"
#define CLIENTID1     "Client3"
#define CLIENTID2     "Client4"
#define TOPICPRIVA    "zolertia/datos_sensor"
#define TOPICPUBL     "datos"
#define PAYLOAD       ""
#define QOS           1
#define TIMEOUT       10000L
```

**Código 2.13.** Definición de variables en el *script* 2

El programa principal crea dos clientes, el cliente `Client3`, que se conecta al *broker* privado y el cliente `Client4` que se conecta al *broker* público. El cliente `Client3` se suscribe al tópico `zolertia/datos_sensor` a la espera de mensajes de la red de nodos, y al final del *script* quita la suscripción y se desconecta el cliente; el código del programa principal del *script* 2 se muestra en el Código 2.14.

```
int main(int argc, char* argv[])
{
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    int rc1, ch;

    MQTTClient_create(&cliente, ADDRESSPRIVA, CLIENTID1, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    MQTTClient_create(&cliente2, ADDRESSPUBL, CLIENTID2, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    conn_opts.keepAliveInterval = 20;
    conn_opts.cleansession = 1;

    MQTTClient_setCallbacks(cliente, NULL, connlost, msgarrvd, delivered);

    if ((rc1 = MQTTClient_connect(cliente, &conn_opts)) != MQTTCLIENT_SUCCESS)
    {
        printf("Failed to connect, return code %d\n", rc1);
        exit(EXIT_FAILURE);
    }
    if ((rc1 = MQTTClient_connect(cliente2, &conn_opts)) != MQTTCLIENT_SUCCESS)
    {
        printf("Failed to connect, return code %d\n", rc1);
        exit(EXIT_FAILURE);
    }

    printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"
           "Press Q<Enter> to quit\n\n", TOPICPRIVA, CLIENTID1, QOS);
    MQTTClient_subscribe(cliente, TOPICPRIVA, QOS);

    do{
        ch = getchar();
    } while(ch!='Q' && ch != 'q');

    MQTTClient_unsubscribe(cliente, TOPICPRIVA);
    MQTTClient_disconnect(cliente, 10000);
    MQTTClient_destroy(&cliente);
    return rc1;
}
```

**Código 2.14.** Programa principal del *script* 2

A la función `msgarrv()` se llama cuando llega un mensaje desde la red de nodos, en el tópico `zolertia/datos_sensor` a través del *broker* privado Mosquitto; cuando llega un mensaje se lo almacena en una cadena y luego el cliente `Client4` publica el mensaje en el tópico `datos` mediante el *broker* público a la aplicación móvil. Luego de publicar el mensaje se libera el mensaje y el tópico. Esta función se observa en el Código 2.15.

```
int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)
{
    int i;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr1 = message->payload;
    for(i=0; i<message->payloadlen; i++)
    {
        putchar(*payloadptr1++);
    }
    MQTTClient_publish(cliente2, TOPICPUBL, message->payloadlen, message->payload, QOS, 1, &token);
    putchar('\n');
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}
```

**Código 2.15.** Función `msgarrv()` del *script* 2

Cada uno de los *scripts* debe ser compilado y ejecutado en la Raspberry Pi, con el fin de establecer la comunicación entre la red de nodos y la aplicación móvil en ambas direcciones. Todos los mensajes recibidos son redireccionados hacia la red de nodos o a la aplicación móvil según corresponda; de esta manera el usuario final estará enterado de todos los datos del cultivo, donde los nodos de sensores están enviando información periódicamente. En caso de que el usuario final desee conocer el nivel de agua o accionar la bomba de regadío, enviará un mensaje que, al llegar al nodo enviará un informe o accionará la bomba, según corresponda.

### 2.4.7. APLICACIÓN MÓVIL

La aplicación móvil se codifica y diseña en el IDE Android Studio, y utiliza las herramientas que provee el Servicio Paho Android de Eclipse para establecer la comunicación gracias al protocolo MQTT. La aplicación móvil permitirá visualizar los datos entrantes desde la red de nodos, y mediante controles se podrá solicitar información del nivel de agua y accionar la bomba de regadío.

Dentro de cada proyecto en Android Studio existen archivos de código fuente y recursos para las aplicaciones, así como archivos de compilación. Todos estos archivos se encuentran organizados por módulos en la pestaña *Project,* ubicada en la parte izquierda de la interfaz de Android Studio. Para utilizar la fuente de código que provee Paho Android, es necesario añadir el repositorio desde Maven y la dependencia en el archivo *build.gradle*, ubicada en el Módulo *Gradle Scripts*, como se muestra en la Figura 2.19.

```
repositories {
    maven {
        url "https://repo.eclipse.org/content/repositories/paho-releases/"
    }
}
dependencies {
    implementation('org.eclipse.paho:org.eclipse.paho.android.service:1.0.2') {
        exclude module: 'support-v4'
    }
```

**Figura 2.19.** Repositorio y dependencia para Paho Android

Luego en el archivo de compilación *AndroidManifest.xml* localizado en el módulo *Application/manifest*, se deben declarar algunos permisos para que se pueda usar el Servicio Paho Android. El primer permiso `android.permission.WAKE_LOCK` permite que la aplicación controle el estado de energía del dispositivo, con el fin de mantener la conexión con el *broker*, evitando que el dispositivo entre en el modo de suspensión y se desconecte del *broker*. El siguiente permiso `android.permission.INTERNET` permite a las aplicaciones abrir los *sockets* de red, con la finalidad de conectarse al *broker* público Mosquitto. El permiso `android.permission.ACCESS_NETWORK_STATE` permite que la aplicación acceda a información acerca de las redes, para que la aplicación conozca los cambios en la red como, la conexión o desconexión. El último permiso `android.permission.READ_PHONE_STATE` permite acceder en modo sólo lectura al estado del teléfono *host*. Además, el servicio Paho Android debe ser declarado en el mismo archivo, como se observa en la Figura 2.20.

```
<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" />
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="FarmerApp"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    tools:ignore="GoogleAppIndexingWarning">
    <activity android:name=".ClienteMQTT">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <service android:name="org.eclipse.paho.android.service.MqttService" >
    </service>
</application>
```

**Figura 2.20.** Declaración del Servicio Paho Android

Después de realizar los pasos anteriores se diseña la interfaz gráfica del aplicativo móvil en el archivo *activity_main.xml*; este archivo tiene dos formas de implementación, diseño y texto. Diseño permite gráficamente diseñar la actividad y texto permite visualizar todo el diseño en texto XML. Este archivo es la actividad principal de la aplicación donde se muestra la interfaz que debe contener los controles necesarios para que el usuario visualice la información que llega desde el cliente MQTT en la Raspberry Pi, y también pueda accionar la bomba de agua.

Para el cumplimiento de las tareas del Tablero Kanban, la interfaz de la aplicación móvil se diseña según se observa en la Figura 2.21. Esta interfaz se desarrolla mediante algunos tipos de controles para que el usuario lo use fácilmente.


**Figura 2.21.** Interfaz gráfica de la aplicación móvil

Cada uno de los controles tiene identificadores y una finalidad asociada, esta información se la puede observar en la Tabla 2.3.

**Tabla 2.3.** Controles de la aplicación móvil

| Nombre | Control | Tarea |
|---|---|---|
| imgView1 | ImageView | Contiene la identificación de la Escuela Politécnica Nacional. |
| txtView1 | TextView | Contiene el nombre del proyecto resumido. |
| btnNiv | Button | Permite solicitar la información de nivel de agua. |
| switch1 | Switch | Permite accionar el encendido y apagado de la bomba de regadío. |
| listViewInforme | ListView | Permite visualizar la información que llega desde el *broker* público. |

Android Studio opera con Java, lenguaje orientado a objetos, por lo que se trabaja mediante clases. En esta aplicación se emplean tres clases, la primera es *ClienteMQTT*, aquí se codifica el programa principal, las otras clases *BombaAgua* y *Adaptador* proveen datos y métodos a utilizar en el programa principal.

**a. Clase ClienteMQTT**

La clase *ClienteMQTT* contiene la declaración de objetos, el método `onCreate()` y dos métodos creados para dar funcionalidad a los controles. Los objetos que se declaran son de tipo *private* porque se usan únicamente dentro de esta clase *ClienteMQTT*. El método `onCreate()` es llamado cuando se crea la actividad principal *activity_main.xml*; es por esto que aquí se inicializan los controles de la actividad y se implementan las funciones que desempeñan cada uno de éstos. Dentro de este método también se llama al método `setContentView()`, para definir el diseño de la actividad principal, contenida en el archivo *activity_main.xml*. Dentro del método `onCreate()` también se instancian los objetos y se relacionan los controles de la actividad principal con los objetos declarados mediante la función `findViewById()`.

Para la creación del cliente MQTT primero se genera un identificador del cliente mediante la función `MqttClient.generateClientId()`, luego se crea un objeto `client` del tipo de dato `MqttAndroidClient` y se lo instancia a través de su constructor con tres parámetros de entrada. El primero es el contexto en el cual se instancia, luego la dirección URI del *broker* público MQTT al que se quiere conectar el cliente MQTT, y finalmente el identificador del cliente generado anteriormente. Este código se muestra en el Código 2.16.

La conexión del cliente se codifica dentro un bloque `try{}` con la finalidad de gestionar errores en esta porción de código; el cliente se conecta llamando a la función `client.connect()`, la cual entrega un *token* que sirve como método de sincronismo entre las acciones que se ejecutan. Mediante este *token* se llama a la función `setActionCallback()` para registrar un *Listener* que notifique cuando se complete la acción que se describe dentro de esta función; aquí se definen dos funciones `onSuccess()` y `onFailure()`, la primera se ejecutará cuando la conexión haya resultado exitosa y la segunda cuando no se haya logrado la conexión del cliente.

En la función `onSuccess()` se codifica un mensaje que se mostrará en el dispositivo móvil mediante el método `makeText()` de la clase `Toast`, donde se pasa como parámetros el contexto de la Clase *ClienteMQTT*, el mensaje a mostrar y la duración del mensaje en pantalla.

Luego se codifica la suscripción del cliente en el tópico correspondiente con la calidad de servicio declarada. La suscripción se maneja de igual manera que la conexión, así es que la función `suscribe()` devuelve otro *token*, que se utiliza para registrar otro *Listener* que notifique cuando la acción que se describa dentro del método `setActionCallback()` se cumpla. Si la suscripción es exitosa se ejecuta la función `onSuccess()` donde lo primero que se codifica es la notificación de un mensaje en el teléfono mediante la clase `Toast`. Para recibir y manejar un mensaje que llegue en el tópico suscrito desde el *broker* al que se conectó, se requiere llamar al método `setCallback()` del cual se implementa el método `messageArrived()` que recibe como argumentos, el tópico y un objeto `message` del tipo `MqttMessage`.

En caso de que la suscripción o la conexión sean fallidas, entonces se ejecutará la función `onFailure()`, donde se codifica un mensaje de notificación mediante la clase `Toast`, indicando que la conexión o suscripción son fallidas.

```
String clientId = MqttClient.generateClientId();
final MqttAndroidClient client = new MqttAndroidClient(
        context: ClienteMQTT.this , serverURI: "tcp://iot.eclipse.org:1883", clientId);
try {
    IMqttToken token = client.connect();
    token.setActionCallback(new IMqttActionListener() {

        public void onSuccess(IMqttToken asyncActionToken) {
            Toast.makeText( context: ClienteMQTT.this,  text: "Conectado", Toast.LENGTH_SHORT).show();
            objBombaAgua.setTopicoRecDatos("datos");
            String topic = objBombaAgua.getTopicoRecDatos();
            int qos = 0;

            try {
                IMqttToken subToken = client.subscribe(topic, qos);
                subToken.setActionCallback(new IMqttActionListener() {
                    public void onSuccess(IMqttToken asyncActionToken) {
                        Toast.makeText( context: ClienteMQTT.this,  text: "Suscrito", Toast.LENGTH_SHORT).show();
                        client.setCallback(new MqttCallback() {
                            public void messageArrived(String topic, MqttMessage message) throws Exception {
```

**Código 2.16.** Fragmento de código de conexión del cliente MQTT

Para solicitar la información del nivel de agua se crea una función denominada `solicitarInformeNivel()`, los parámetros que recibe son, el objeto `client` y el objeto `objBombaAgua`, que es una instancia de la clase *BombaAgua*; esta función se puede observar en el Código 2.17. Para enviar el mensaje se configuran tanto el tópico como el mensaje en el objeto `objBombaAgua`. El mensaje a enviar se codifica en un tipo de dato `byte[]` y se asigna a un objeto `MqttMessage`; luego a través del método `publish()` se publica el mensaje en el tópico indicado. El mensaje enviado corresponde a 2 porque del lado de la mota que tiene conectado los sensores de nivel de agua, el mensaje significará que debe enviar información acerca del nivel de agua.

```
public void solicitarInformeNivel(final MqttAndroidClient client, final BombaAgua objBombaAgua){
    objBombaAgua.setTopicoComandar("command");
    String topic = objBombaAgua.getTopicoComandar();
    objBombaAgua.setMensajeInforme("2");
    String payload = objBombaAgua.getMensajeInforme();
    byte[] encodedPayload = new byte[0];
    int qos = 0;
    try {
        encodedPayload = payload.getBytes( charsetName: "UTF-8");
        MqttMessage message = new MqttMessage(encodedPayload);
        client.publish(topic, message);
    } catch (UnsupportedEncodingException | MqttException e) {
        e.printStackTrace();
    }
}
```

**Código 2.17.** Función `solicitarInformeNivel()`

Para que el usuario pueda accionar la bomba de agua, se utiliza el control *Switch*. Para codificar esta funcionalidad se ha creado una función llamada `comandarEncApag()`, la cual se puede apreciar en el Código 2.18. Esta función se compone de dos partes, la primera se ejecuta cuando el *switch* se encienda y la otra cuando se apague; en ambos casos el procedimiento es similar. Primero se describe el mensaje a publicar y luego se obtiene el tópico; el mensaje también es codificado en un tipo de dato `byte[]`, y luego es asignado a un objeto del tipo `MqttMessage` para su posterior publicación.

```
public void comandarEncApag(final MqttAndroidClient client,final BombaAgua objBombaAgua){
    if (switchComandar.isChecked()) {
        String topic = objBombaAgua.getTopicoComandar();
        objBombaAgua.setMensajeEncender("1");
        String payload = objBombaAgua.getMensajeEncender();
        byte[] encodedPayload = new byte[0];
        try {
            encodedPayload = payload.getBytes( charsetName: "UTF-8");
            MqttMessage message = new MqttMessage(encodedPayload);
            client.publish(topic, message);
        } catch (UnsupportedEncodingException | MqttException e) {
            e.printStackTrace();
        }
    } else {
        String topic = objBombaAgua.getTopicoComandar();
        objBombaAgua.setMensajeApagar("0");
        String payload = objBombaAgua.getMensajeApagar();
        byte[] encodedPayload = new byte[0];
        try {
            encodedPayload = payload.getBytes( charsetName: "UTF-8");
            MqttMessage message = new MqttMessage(encodedPayload);
            client.publish(topic, message);
        } catch (UnsupportedEncodingException | MqttException e) {
            e.printStackTrace();
        }
    }
}
```

**Código 2.18.** Función `comandarEncApag()`

Los mensajes enviados cuando el *switch* se encienda o se apague, corresponden a valores que del lado de la mota conectada a la bomba de agua significarán que se encenderá o apagará la bomba de regadío; en este caso el mensaje 1 será para el encendido mientras que 0 será para el apagado de la bomba.

**b. Clase BombaAgua**

La clase *BombaAgua* contiene atributos que permitirán el accionamiento de la bomba de agua, entre los cuales se encuentran, el tópico que permitirá comandar el encendido y apagado, el tópico que permitirá recibir la información de nivel de agua, los mensajes que permitirán encender y apagar la bomba de agua, así como solicitar el informe del nivel de agua. En esta clase también se definen métodos que son básicamente *getters* y *setters*, a fin de asignar y obtener los datos de los atributos ya declarados.

**c. Clase Adaptador**

Todos los mensajes que llegan al cliente MQTT creado en esta aplicación, se deben mostrar en un control *ListView*, de modo que se puedan ver todos los mensajes. Para emplear este control es necesario crear una clase *Adaptador* que permita pasar los datos al *ListView*. Esta clase tiene como atributos una lista del tipo *ArrayList* y una actividad del tipo *Activity*; además contiene su constructor y otros métodos, que serán utilizados en el programa principal. Dentro de esta clase se crea una clase denominada *ViewHolder* que almacena las referencias de los controles que van dentro del *ListView*, para poder usarlos después; en la aplicación se utilizan 5 columnas para cada mensaje que llega desde el *broker*, es por esto que en la clase *ViewHolder* se declaran 5 controles *TextView* con sus respectivos nombres.

Los métodos que usa esta clase son `getView()`, `getCount()`, `getItem()` y `getItemId()`. El método `getView()`, se aprecia en el Código 2.19. Este método usa como parámetro `convertView` del tipo `View` con el fin de mostrar los datos anteriores reciclados en el control *ListView*; otro parámetro que utiliza es la posición del ítem del conjunto de mensajes que llegan al cliente MQTT y el último parámetro que recibe es del tipo `ViewGroup`, el cual representa al grupo de vistas y permite organizarlas para su visualización.

Dentro del método `getView()`, se relacionan los controles `TextView` con los cinco elementos de la clase `ViewHolder`, elementos que representan las cinco columnas de cada ítem de la lista. Este código tiene como objeto que los elementos de la lista se instancien una sola vez, para que luego sean mostrados en la lista; cada vez que llegue un mensaje desde el *broker*, se llamará a este método de la clase *Adaptador*.

```
public View getView(int posicion, View convertVista, ViewGroup padre) {
    ViewHolder holder;

    LayoutInflater inflater = actividad.getLayoutInflater();

    if (convertVista==null) {
        convertVista = inflater.inflate(R.layout.column_row, root: null);
        holder = new ViewHolder();
        holder.txtPrimero = (TextView) convertVista.findViewById(R.id.TxtPrimero);
        holder.txtSegundo = (TextView) convertVista.findViewById(R.id.TxtSegundo);
        holder.txtTercero = (TextView) convertVista.findViewById(R.id.TxtTercero);
        holder.txtCuarto = (TextView) convertVista.findViewById(R.id.TxtCuarto);
        holder.txtQuinto = (TextView) convertVista.findViewById(R.id.TxtQuinto);

        convertVista.setTag(holder);
    }else {
        holder =(ViewHolder)convertVista.getTag();
    }
    HashMap<String, String> map = lista.get(posicion);
    holder.txtPrimero.setText(map.get(PRIMERA_COLUMNA));
    holder.txtSegundo.setText(map.get(SEGUNDA_COLUMNA));
    holder.txtTercero.setText(map.get(TERCERA_COLUMNA));
    holder.txtCuarto.setText(map.get(CUARTA_COLUMNA));
    holder.txtQuinto.setText(map.get(QUINTA_COLUMNA));

    return convertVista;
}
```

**Código 2.19.** Método `getView()`

Despues de instanciar los elementos de la lista, cada elemento o columna carga el texto a mostrar, para lo cual se hace uso de un objeto tipo `HashMap` que permite relacionar una clave con un valor, ambos en este caso son del tipo `String`, donde la clave única será el nombre de la columna y el valor variará según arriben los mensajes y se los clasifique en las cinco columnas existentes. Como retorno, este método devuelve la vista de un mensaje del *ListView*.

# 3. RESULTADOS Y DISCUSIÓN

## 3.1. IMPLEMENTACIÓN

Para la implementación el proyecto cuenta con tres bloques que se pueden apreciar en la Figura 3.1. El primer bloque contiene a los nodos sensores y al nodo actuador; el segundo bloque abarca el *router* de borde y el tercer bloque se refiere al servidor y la aplicación móvil.

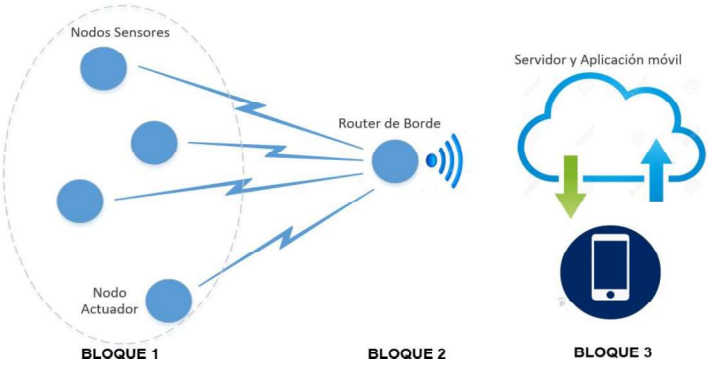



**Figura 3.1.** Bloques del proyecto

La implementación del prototipo se realiza en un área de cultivo de dimensiones 60x50 metros, ubicado en la comunidad Gatazo Chico, parroquia Cajabamba, cantón Colta, provincia Chimborazo. El despliegue del proyecto se muestra diagramado en la Figura 3.2.

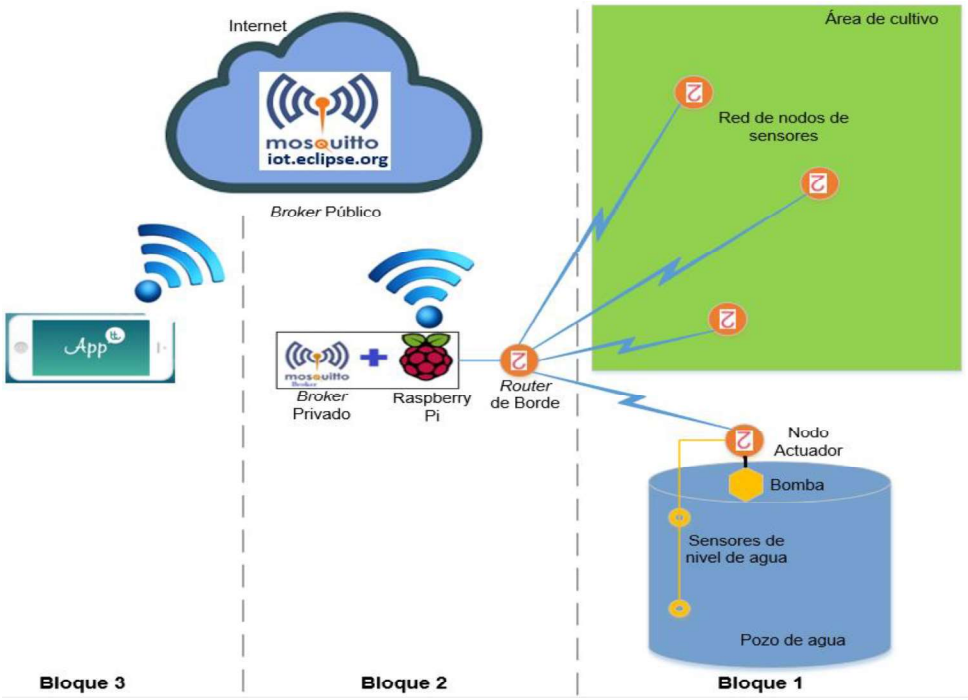


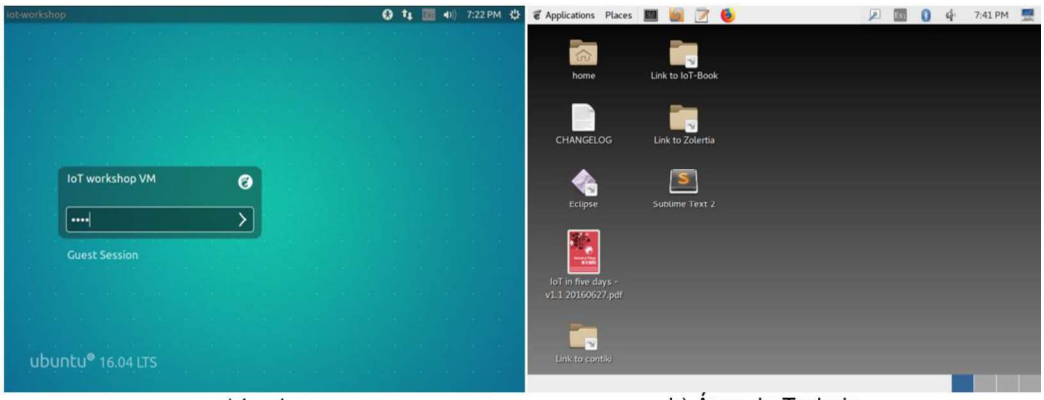

**Figura 3.2.** Despliegue del prototipo

Los nodos sensores están ubicados en diferentes lugares dentro del área de cultivo, los mismos que se conectan al *router* de borde mediante el estándar IEEE 802.15.4; el *router* de borde a su vez se conecta a la Raspberry Pi, donde se aloja el *broker* privado Mosquitto.

Gracias a que la Raspberry Pi se conecta mediante WiFi al *broker* público Mosquitto, la red de nodos se conecta con la aplicación móvil; por otro lado, el nodo actuador se conecta a la bomba de regadío y a los sensores de nivel de agua, también se conecta al *router* de borde al igual que los demás nodos a fin de conectarse con la aplicación móvil pasando por los dos *brokers* tanto privado como público.

Finalmente, la aplicación móvil mediante WiFi se conecta al *broker* público para que el usuario interactúe con el sistema.

### 3.1.1. INSTALACIÓN DE CONTIKI




a) Login b) Área de Trabajo

**Figura 3.3.** Contiki

En primer lugar, se instala Contiki en una máquina virtual Ubuntu Linux descargada de la página oficial de Contiki *www.contiki-os.org/start.html*. Ésta se ejecuta en VMWare Player, la versión que utiliza es Instant Contiki y la contraseña de ingreso es *user;* la vista de *login* es la que se muestra en la Figura 3.3.a., mientras que el área de trabajo de Contiki se encuentra en la Figura 3.3.b.

En Contiki se trabaja con varios archivos, los cuales se muestran dentro de la carpeta *contiki*; entre ellos se ubica la carpeta *examples*, en la cual se almacenan varios ejemplos prácticos que se pueden ejecutar en distintas plataformas, entre ellas la plataforma Zolertia RE-Mote.

La carpeta principal Contiki contiene todos los archivos que se muestran en la Figura 3.4., entre los cuales están los archivos de compilación y ejecución para distintas plataformas. Todos los archivos están escritos basados en lenguaje C.

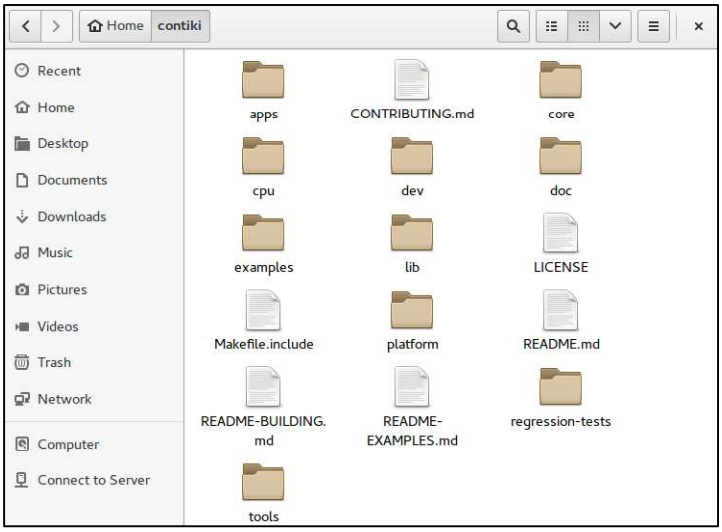



**Figura 3.4.** Carpeta Contiki

Para la programación de las motas se usa el terminal de comandos de Contiki, y se emplean los comandos que se muestran en la Tabla 3.1.

**Tabla 3.1.** Comandos Contiki

| Comando | Descripción |
|---|---|
| gedit mqtt-sensores.c & | Permite abrir el archivo mediante el editor de texto Gedit, y seguir utilizando el terminal. |
| make motelist | Muestra las plataformas conectadas y su ubicación. |
| make TARGET=zoul savetarget | Permite crear el archivo *Makefile* indicando que se cargará en un nodo tipo Zolertia RE-Mote. |
| make PORT=/dev/ttyUSB0 | Especifica el dispositivo con el que se desea trabajar. |
| make mqtt-sensores.upload login | Carga el programa *mqtt-sensores* en el nodo especificado y además muestra la salida de mensajes de la mota. |
| make clean | Borra el último programa cargado en la mota. |

### 3.1.2. BLOQUE 1: RED DE NODOS Y NODO ACTUADOR

El primer bloque se compone de 4 nodos, éstos son las motas de la plataforma Zolertia RE-Mote; tres de estas motas tienen conectados sensores de temperatura y humedad, las mismas que se encuentran ubicadas en el área del cultivo a distancias lejanas a fin de cubrir toda el área.

La cuarta mota es la que actúa como nodo actuador, ésta se encuentra ubicada cerca del pozo de agua, debido a que tiene conectado sensores de nivel que se encuentran a distintos niveles en el pozo de agua y también se conecta a la bomba a través de un relé de estado sólido.

Todas las motas se conectan mediante el estándar IEEE 802.15.4 al nodo *router* de borde. Este primer bloque se puede observar en la Figura 3.5.

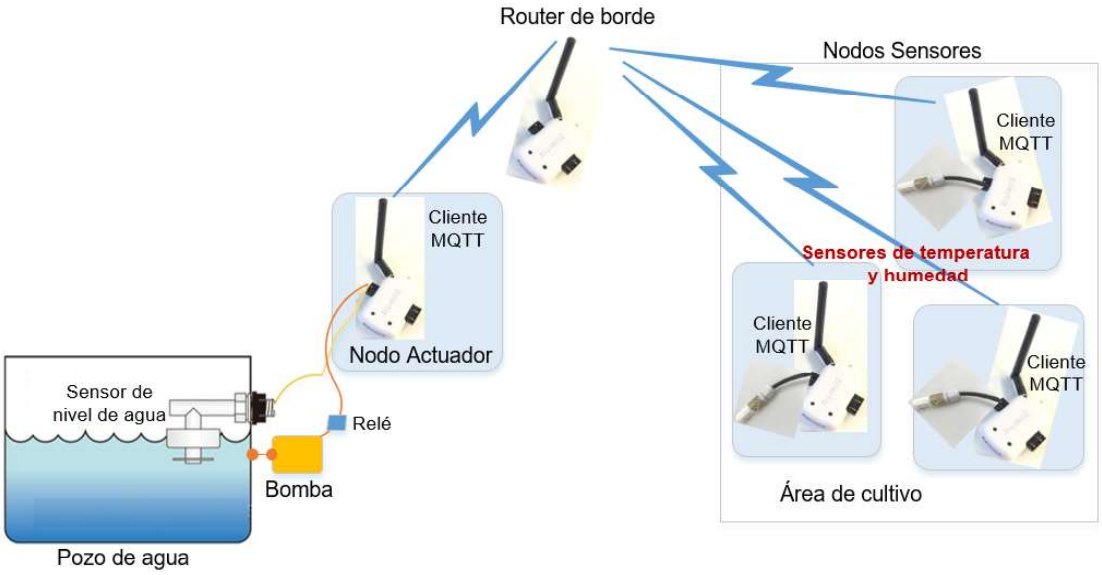



**Figura 3.5.** Primer Bloque del prototipo

El programa que se carga en las motas que se ubicarán en el área de cultivo es *mqtt-sensores.c*. Mediante este programa se crea un cliente MQTT que publica al tópico *zolertia/datos_sensor*, los datos tomados desde el sensor SHT25. El *broker* al que se conecta es Mosquitto, el cual está instalado en la Raspberry Pi. La salida del nodo se muestra en la Figura 3.6, donde se observa la dirección física del nodo; en las líneas siguientes se indica que ya se ha establecido una conexión MQTT y empieza a publicar los datos de humedad y temperatura en el tópico mencionado anteriormente. Esta conexión tiende a demorar un corto tiempo es por eso que se indica que el nodo está en estado *Connecting*.

```
Rime configured with address 00:12:4b:00:06:0d:61:c5
 Net: sicslowpan
 MAC: CSMA
 RDC: nullrdc
Proceso MQTT
Init
Registered. Connect attempt 1
Connecting (1)
Connecting (1)
Connecting (1)
Connecting (1)
Connecting (1)
Connecting (1)
Connecting (1)
Connecting (1)
Connecting (1)
Connecting (1)
Connecting (1)
Application has a MQTT connection
Publishing
APP - Publish to zolertia/datos_sensor: 61c5,17.66,60.80             ,"Seq no":1,
Publishing
```

**Figura 3.6.** Salida del nodo sensor

Las motas se colocan en los surcos del cultivo, como se observa en la Figura 3.7. Cada una tiene una batería de litio como fuente de energía, y se conecta a un sensor de temperatura y humedad SHT25.

**Figura 3.7.** Mota ubicada en el cultivo conectada al sensor

Los pines que se utilizan para conectar el sensor SHT25 son cuatro, cuyas funciones se describen en la Tabla 3.2.

**Tabla 3.2.** Funcionalidad de los pines utilizados por el sensor SHT25

| Pin | Descripción |
|---|---|
| PC2 | Es un pin digital de entrada y salida, el cual utiliza el protocolo de comunicación I2C y se usa para la transmisión de datos. |
| PC3 | Es un pin digital de entrada y salida que se usa como señal de reloj para la transmisión de datos y utiliza el protocolo de comunicación I2C. |
| GND | Es un pin que funciona como la conexión a tierra. |
| VCC | Es un pin digital de salida de potencia de 3.3 V DC, que alimenta al sensor para que funcione adecuadamente. |

Cada una de las motas tiene una antena que trabaja a 2.4 GHz que está conectada a través del conector RP-SMA; esta antena permite que la comunicación mediante el estándar IEEE 802.15.4 sea establecida.

El nodo actuador se ubica cerca del pozo de agua, porque tiene conectado dos sensores de nivel de agua colocados en el pozo; estos sensores informan el nivel de agua en el pozo y publican esta información al tópico *zolertia/datos_sensor*. Esta mota se puede observar en la Figura 3.8.


**Figura 3.8.** Nodo actuador en el pozo

Esta mota adicionalmente se conecta a la bomba de regadío mediante dos dispositivos electrónicos, un relé de estado sólido y un convertidor de voltaje DC-DC MT3608, con el fin de proveer el voltaje y corriente necesarios para el accionamiento de la bomba; esto se muestra en la Figura 3.8.c. En la Figura 3.8.b se observa el pozo de agua y un tubo que contiene los sensores de nivel de agua. El control del accionamiento de la bomba está en la mota, la cual al recibir un mensaje en el tópico *zolertia/cmd/bomb* encenderá o apagará la bomba de regadío. El programa que se carga en esta mota es *sensor-agua.c*, mediante el cual se crea un cliente MQTT que se suscribe al tópico *zolertia/cmd/bomb* y publica al tópico *zolertia/datos_sensor*.

### 3.1.3. INSTALACIÓN DE RASPBIAN

La Raspberry Pi debe estar conectada a los dispositivos de salida y entrada, esto es al teclado, monitor y mouse; luego es importante que la memoria SD sea formateada correctamente y que almacene el sistema operativo Raspbian, el cual se descarga de la página oficial de Raspberry Pi *https://www.raspberrypi.org/downloads/*.

Una vez que la memoria contiene Raspbian, se inserta en la Raspberry Pi y se conecta la fuente de alimentación al Raspberry Pi; a continuación, empezará la instalación del sistema operativo, la misma que toma algunos minutos. La interfaz que presenta Raspbian luego de la instalación, es la que se muestra en la Figura 3.9.

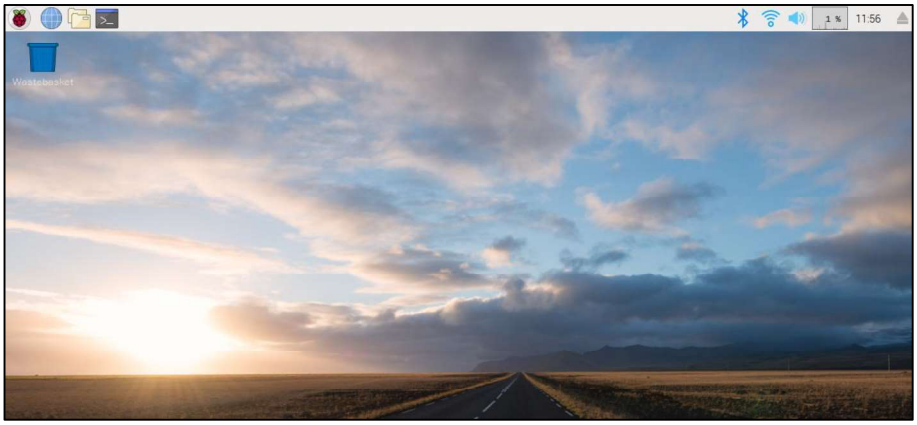


**Figura 3.9.** Interfaz de Raspbian

Una vez que el sistema operativo está instalado, se instala Contiki mediante comandos en el terminal; de igual manera se lo descarga de la página principal de Contiki y se instalan otras dependencias necesarias para trabajar con Contiki. Al final de la instalación de Contiki se obtiene una carpeta denominada *contiki*, que contiene varios archivos que permitirán programar distintos dispositivos; esta instalación se realiza para cargar el programa de *router* de borde en una mota, de modo que pueda conectarse con los demás nodos y sea el punto de conexión con la red externa. La carpeta *contiki* se muestra en la Figura 3.10.

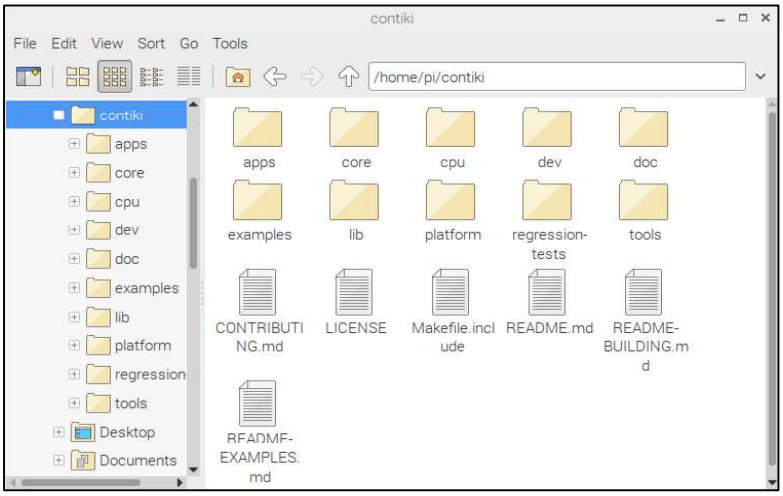



**Figura 3.10.** Carpeta contiki en Raspbian

### 3.1.4. INSTALACIÓN DE PAHO CLIENT C

Paho Client C es una herramienta importante para permitir la comunicación con el servidor público, por ello se la instala en la Raspberry Pi, donde se crearán clientes MQTT. La instalación se la realiza mediante el terminal de comandos; los pasos para esta instalación

se los puede tomar de la página oficial de Cliente Paho C https://www.eclipse.org/paho/clients/c/.

Luego de ejecutar todos los pasos se reinicia la Raspberry para terminar con la instalación. Como resultado se habrá creado una carpeta denominada *paho-c* que contiene otra carpeta *paho.mqtt.c*, donde se encuentran todas las herramientas que permitirán crear clientes y conexiones MQTT. Además, existen ejemplos que se pueden utilizar; esta carpeta y su contenido se muestran en la Figura 3.11.

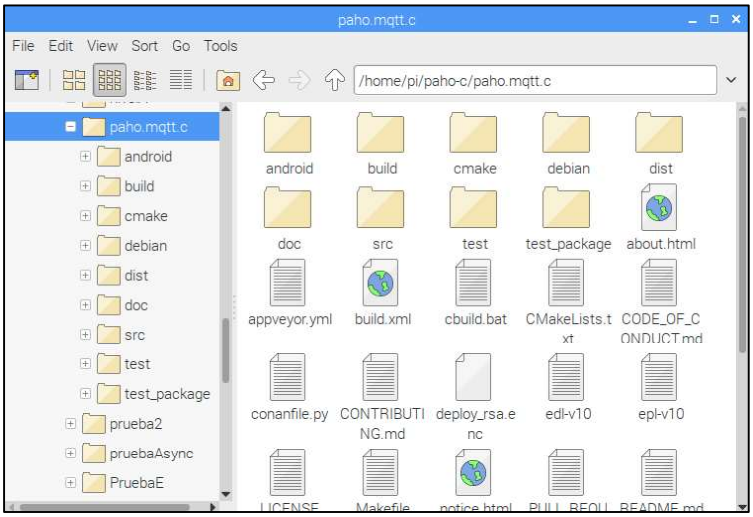



**Figura 3.11.** Carpeta paho-c

### 3.1.5. BLOQUE 2: *ROUTER* DE BORDE

El segundo bloque corresponde a la implementación del *router* de borde, que consta de una mota conectada a una Raspberry Pi, mediante la cual se conecta a un *broker* privado y luego a un *broker* público, a fin de comunicarse con la aplicación móvil del tercer bloque. El segundo bloque se muestra en la Figura 3.12.

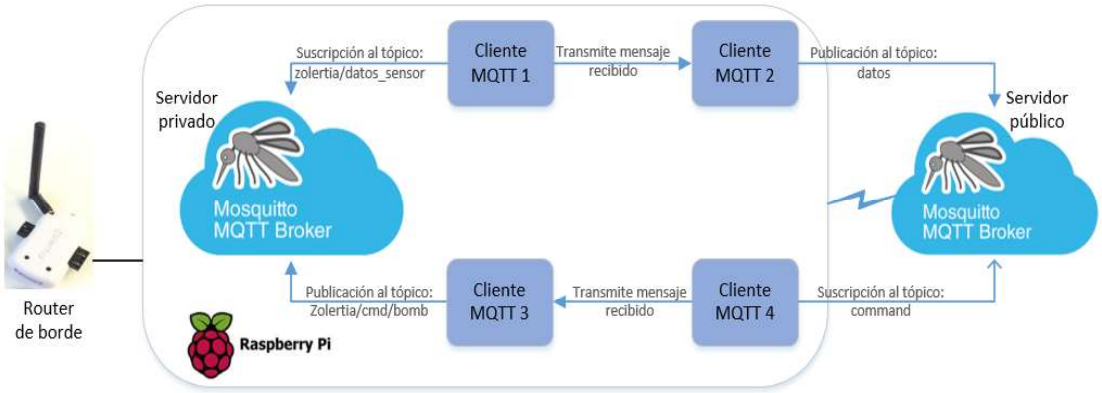



**Figura 3.12.** Segundo Bloque del prototipo

La mota que corresponde al *router* de borde se conecta de forma serial a la Raspberry Pi y ejecuta el programa *border-router*, que se encuentra en la dirección *contiki/examples/ipv6/rpl-border-router*, programa mediante el cual la red de nodos sensores o red 6LowPAN puede salir hacia la red externa. El comando que se ejecuta dentro de la ruta especificada es `make border-router.upload connect-router`.

La ejecución de este programa y la conexión serial permiten la creación de la interfaz virtual *tun0*, mediante la cual los nodos sensores pueden salir a Internet. La salida de la ejecución del programa en la mota se puede observar en la Figura 3.13.

```
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::212:4b00:60d:61ad
 fe80::212:4b00:60d:61ad
slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::212:4b00:60d:6230
```

**Figura 3.13.** Salida del programa Router de Borde

En esta figura se observa que el prefijo utilizado es *aaaa::*, y que permite junto con la dirección MAC de la mota donde se ejecuta este programa, formar la dirección IPv6 global mediante la cual se podrá establecer conexión con la red de nodos. La dirección *aaaa::212:4b00:60d:61ad* es la dirección IPv6 del *router* de borde; en la última línea se indica que se ha establecido conexión con otra mota, cuya dirección es *aaaa:212:4b00:60d:6230*, la cual contiene el prefijo predeterminado y su dirección física. Cada vez que detecte un nodo, se conectará a éste y le proveerá del prefijo, y con la dirección física formará la dirección IPv6 correspondiente.

**Figura 3.14.** Router de borde y Raspberry Pi

La conexión del nodo *router* de borde con la Raspberry Pi se muestra en la Figura 3.14.; de igual manera que la red de nodos, esta mota tiene conectada una antena para la comunicación con la red 6LowPAN y obtiene su alimentación de la Raspberry Pi, pues se conecta de forma serial con ésta.

### 3.1.6. INSTALACIÓN DEL SERVIDOR PRIVADO

En la Raspberry Pi se instala el *broker* Mosquitto, que también se denomina servidor privado; éste se instala a través de la línea de comandos, ingresando al modo super usuario. Según se muestra en la Figura 3.15., los comandos a ingresar permiten actualizar el sistema, luego se instala el servidor Mosquitto y además se instala el cliente para probar que el servidor realmente funciona.

```
apt-get update
apt-get install mosquitto
apt-get install mosquitto-clients
```

**Figura 3.15.** Comandos de instalación del broker privado Mosquitto

Como parte de la implementación del *broker* Mosquitto, se comprueba que en realidad recibe conexiones MQTT, para lo cual primero se ejecuta el *broker* Mosquitto en un terminal con el comando `mosquitto` y a continuación se hace uso del cliente MQTT.

Mediante el comando `mosquitto_sub -h 127.0.0.1 -t prueba -i Cliente1` se crea una conexión hacia el *broker* privado con dirección 127.0.0.1, desde un cliente llamado Cliente1, que se suscribe al tópico prueba. Una vez que esta línea de comando se ejecute en el terminal donde se ejecuta el *broker* Mosquitto, la salida mostrará que existe una nueva conexión tal como se muestra en la Figura 3.16., indicando el nombre que identifica al nuevo cliente y que la conexión se ha establecido a través del puerto 1883.

```
root@raspberrypi:/home/pi# mosquitto
1558559222: mosquitto version 1.5.4 starting
1558559222: Using default config.
1558559222: Opening ipv4 listen socket on port 1883.
1558559222: Opening ipv6 listen socket on port 1883.
1558559297: New connection from 127.0.0.1 on port 1883.
1558559297: New client connected from 127.0.0.1 as Cliente1 (c1, k60).
```

**Figura 3.16.** Salida del broker privado Mosquitto

### 3.1.7. CREACIÓN DE LOS *SCRIPT*S

Se crean dos *script*s de código para establecer la comunicación entre la red 6LowPAN y la aplicación móvil. Ambos *script*s están basados en uno de los ejemplos de Paho Client C, donde se utilizan las funciones que brinda Paho para implementar la conexión.

El primer *script* permite crear dos clientes MQTT; el cliente 3 se suscribe al tópico *zolertia/datos_sensor* del *broker* privado Mosquitto y la información que recibe desde este servidor (la información recibida es generada por la red de nodos) se transmite al cliente 4, y este último publica dicha información en el tópico *datos* hacia el servidor público Mosquitto. El segundo *script* crea dos clientes MQTT; el cliente 1 se suscribe al tópico *command* y recibe información desde el *broker* público y lo transmite al cliente 2, donde

éste lo publica en el tópico *zolertia/cmd/bomb*. La conexión hacia el *broker* público tiene lugar gracias a que la Raspberry Pi se conecta a Internet.

Para la ejecución de cada *script* es necesario que en la misma ubicación del programa se encuentren dos archivos de cabecera, éstos son *MQTTClient.h* y *MQTTPersistence.h*. Para su compilación y ejecución se ingresan dos comandos, cuyas salidas se muestran en la Figura 3.17., donde queda a la espera de mensajes desde otro cliente que puede ser de la red de nodos o de la aplicación móvil.

```
root@raspberrypi:/home/pi/paho-c/RxMensajeNodos# gcc -L /home/pi/paho-c/paho.mqt
t.c/build/output Conexion1.c -l paho-mqtt3c
root@raspberrypi:/home/pi/paho-c/RxMensajeNodos# ./a.out

Connected to iot.eclipse.org
Subscribing to topic command
for client Client1 using QoS0

Press Q<Enter> to quit
```

**Figura 3.17.** Compilación y ejecución de un script

Los dos *script*s deben mantenerse en ejecución de forma indefinida, debido a que los mensajes que se transmiten desde la red de nodos de sensores a la aplicación móvil son periódicos, y además los mensajes que viajan desde la aplicación móvil al nodo actuador son transmitidos cuando el usuario final lo considere necesario, es decir, en cualquier momento.

### 3.1.8. INSTALACIÓN DE ANDROID STUDIO

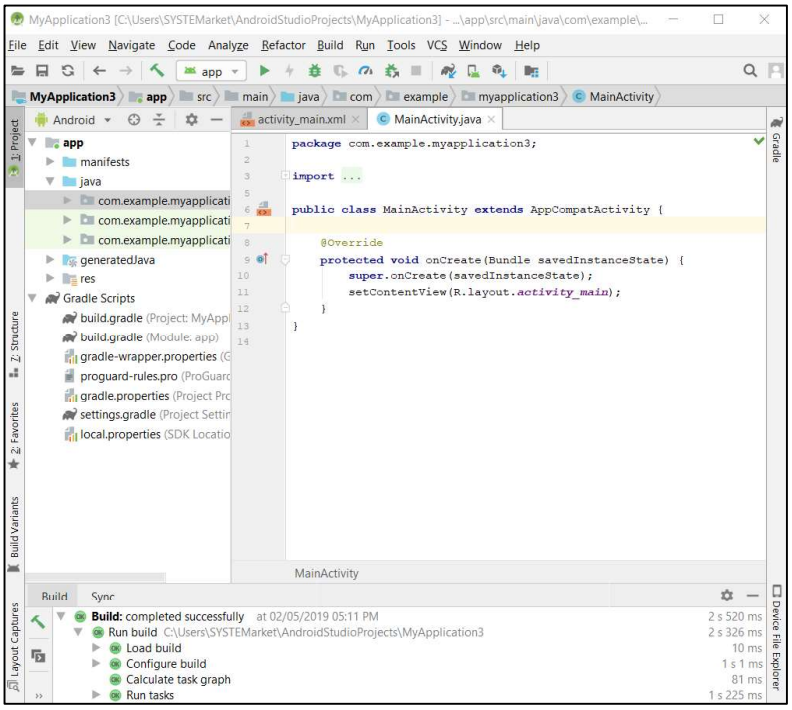



**Figura 3.18.** Nuevo Proyecto en Android Studio

Android Studio se instala en un ordenador, su descarga está disponible en la página oficial de Android Studio *https://developer.android.com/studio/?hl=es-419* en su versión más reciente. Luego de descargarla se lo instala y al final se tendrá el IDE, en el cual se crea un nuevo proyecto vacío para diseñar y codificar la aplicación móvil. El proyecto creado es el que se muestra en la Figura 3.18.

Después de la instalación y creación de un nuevo proyecto es importante incorporar el servicio Paho Android que ya fue explicado en la sección 2.4.7 del Capítulo 2.

### 3.1.9. BLOQUE 3: APLICACIÓN MÓVIL

El tercer bloque se muestra en la Figura 3.19., y comprende una aplicación móvil, la cual está diseñada para que el usuario final visualice información acerca del cultivo y además pueda accionar la bomba de agua; todo esto lo podrá realizar mediante la conexión al *broker* público Mosquitto.


**Figura 3.19.** Tercer bloque del prototipo

La codificación de la aplicación móvil se realiza en el IDE Android Studio, y mediante el servicio Paho Android, se crea un cliente MQTT. El cliente que se conecta al servidor público, se suscribe al tópico *datos*, a fin de recibir los datos de la red de nodos y también publica información al tópico *command*. Todas las acciones que el usuario final puede realizar se logran mediante los controles codificados en la aplicación.

La ejecución de la aplicación se puede realizar conectando un teléfono móvil al ordenador, luego de lo cual, se debe configurar el teléfono móvil para que permita la depuración de la aplicación a través del cable USB. A continuación, se presiona el botón de ejecución en Android Studio, donde mostrará el teléfono conectado como opción para su ejecución y se presiona Aceptar para que la aplicación se ejecute en el teléfono móvil. Mientras se van realizando pruebas y cambios, éstos se mostrarán cada vez que se ejecute la aplicación.

Para que el usuario pueda gestionar los datos en la aplicación móvil es necesario que se conecte a Internet, donde se encuentra el *broker* público. La implementación de la aplicación móvil se puede observar en la Figura 3.20.


**Figura 3.20.** Implementación de la aplicación móvil

Tanto la conexión como la suscripción son procesos que se realizan inmediatamente luego de que el usuario accede a la aplicación, es por esto que la aplicación muestra dos mensajes *Conectado* y *Suscrito*, informando que estos dos pasos son exitosos. Mediante la suscripción al tópico, se pueden observar todos los datos que llegan, a través de un control *ListView;* estos datos serán la temperatura y humedad, el identificador del nodo que envía estos datos y la fecha y hora de llegada de la información. Esta acción de la aplicación móvil se muestra en la Figura 3.21.


**Figura 3.21.** Información de los sensores

Luego el usuario tiene a disposición un control *Button* que permite solicitar información del nivel de agua del pozo; al presionarlo recibirá un mensaje que se visualizará en el control *ListView* indicando el nivel de agua, como se muestra en la Figura 3.22.


**Figura 3.22.** Información Nivel de agua

La información que se presenta luego de presionar el botón *NIVEL DE AGUA* se muestra en un ítem del *ListView* en tres columnas de este control, la fecha y hora de llegada del mensaje, la identificación de la mota que envía el mensaje y finalmente el mensaje en la columna *Nivel Agua*.

Por último, el usuario tiene la opción de accionar la bomba de agua a través de un control *Switch*, el cual al encenderlo activará la bomba de agua, dando lugar al riego del cultivo y al apagarlo desconectará la bomba. Esta acción de la aplicación se puede observar en la Figura 3.23.


**Figura 3.23.** Encendido de la bomba

### 3.1.10. ACTUALIZACIÓN DEL TABLERO KANBAN

Luego de realizar cada una de las actividades se actualiza el Tablero Kanban y al final se tiene el tablero que se muestra en la Figura 3.24.

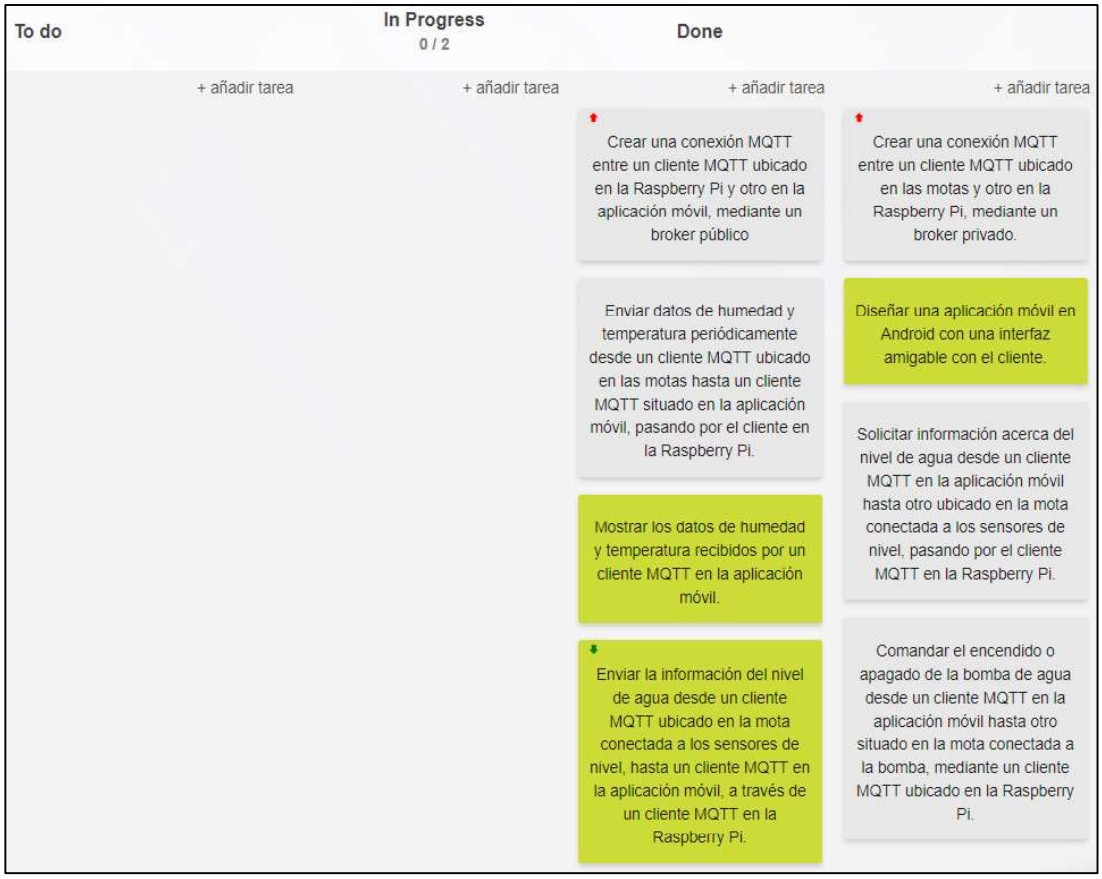



**Figura 3.24.** Actualización del Tablero Kanban

## 3.2. PRUEBAS

Las pruebas se han dividido en varias partes, en las cuales se mostrará el funcionamiento del proyecto por partes y en conjunto.

### 3.2.1. PRUEBAS POR SEPARADO

#### a. Conexión de las motas al *router* de borde

Al momento de ejecutar los programas, las motas esperan a ser detectadas por el nodo *router* de borde, el cual al detectarlas se conecta con cada una de las motas e imprime una corta información de cada una. En la Figura 3.25. se puede apreciar que se han conectado dos nodos cuya dirección IPv6 es provista por el programa *router* de borde.

```
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::212:4b00:60d:61ad
 fe80::212:4b00:60d:61ad
slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::212:4b00:60d:6230
slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::212:4b00:60d:6205
```

**Figura 3.25.** Conexión de las motas con el router de borde

**b. Conexión MQTT de las motas al servidor privado**

Las motas después de haberse conectado con el *router* de borde, se conectarán al *broker* privado Mosquitto, de manera que a la salida del *broker* se imprimirá información de los nodos clientes con los que se ha establecido una conexión MQTT.

En la Figura 3.26., se observa que se han conectado dos motas que se identifican por su dirección física; la dirección IPv6 es la dirección que el *router* de borde ha creado mediante el prefijo *aaaa::* y la dirección física de la mota.

```
root@raspberrypi:/home/pi# mosquitto
1558560975: mosquitto version 1.5.4 starting
1558560975: Using default config.
1558560975: Opening ipv4 listen socket on port 1883.
1558560975: Opening ipv6 listen socket on port 1883.
1558560978: New connection from aaaa::212:4b00:60d:6205 on port 1883.
1558560978: New connection from aaaa::212:4b00:60d:6230 on port 1883.
1558560978: New client connected from aaaa::212:4b00:60d:6230 as d:00124b0d6230 (c1, k1920).
1558560978: New client connected from aaaa::212:4b00:60d:6205 as d:00124b0d6205 (c1, k1920).
```

**Figura 3.26.** Conexión de las motas con el servidor privado

**c. Conexión MQTT entre la aplicación móvil y el servidor público**

La aplicación móvil al ser abierta se conecta al servidor público y además se suscribe a un tópico, de manera que se imprimen dos mensajes (Conectado y Suscrito) cuando estas acciones son exitosas. Esto se muestra en la Figura 3.27.


**Figura 3.27.** Conexión de la aplicación móvil al servidor público

**d. Conexión MQTT entre las motas y un cliente en Raspberry Pi**

Las motas deben conectarse con un cliente MQTT en la Raspberry Pi; esto se puede comprobar al observar que efectivamente la información que ha sido transmitida desde las motas, ha sido recibida por un cliente en la Raspberry Pi, cliente que imprime todos los mensajes que le lleguen en el mismo tópico que las motas han publicado. Esta prueba de conexión se aprecia en la Figura 3.28.

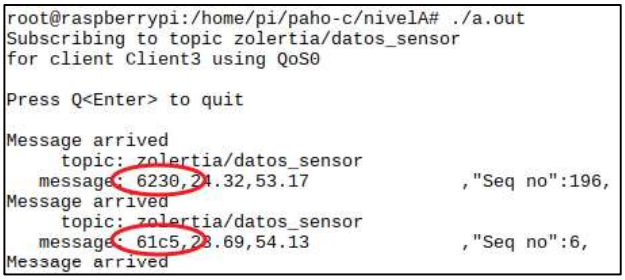



**Figura 3.28.** Conexión MQTT entre las motas y un cliente en la Raspberry Pi

En esta prueba se observa que se han conectado dos motas porque los primeros cuatro números del mensaje identifican a los últimos dígitos de la dirección MAC de las motas.

### e. Conexión MQTT entre un cliente en la Raspberry Pi y el servidor público

Dos de los clientes creados en la Raspberry Pi deben conectarse al servidor público Mosquitto, mediante la dirección *iot.eclipse.org*, por lo que al momento de ejecutar cada *script* se debería imprimir una salida similar a la que se muestra en la Figura 3.29. cuando esta conexión sea exitosa.

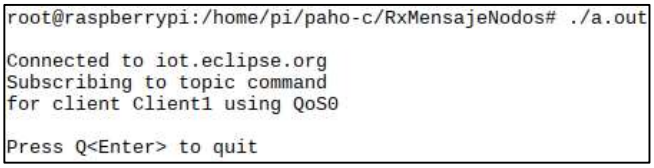



**Figura 3.29.** Conexión MQTT entre un cliente en la Raspberry Pi y el servidor público

### f. Conexión MQTT entre la aplicación móvil y un cliente en la Raspberry Pi

Un cliente en la Raspberry Pi y otro en la aplicación móvil deben mantener una comunicación MQTT a través del *broker* público Mosquitto. La forma de comprobar que existe una conexión exitosa entre ambos es observar a la salida del cliente en la Raspberry Pi, mensajes que hayan sido enviados desde la aplicación móvil, mensajes que llegan desde el tópico *command*. Esta conexión se puede observar en la Figura 3.30.


**Figura 3.30.** Conexión entre la aplicación móvil y un cliente en la Raspberry Pi

### 3.2.2. PRUEBAS DE FUNCIONAMIENTO

Para algunas de las pruebas de funcionamiento global de todo el proyecto se han creado escenarios de pruebas.

**a. Recepción de mensajes de la red de nodos en la aplicación móvil**

El primer escenario se describe en la Figura 3.31. donde los mensajes con información de temperatura y humedad son transmitidos desde los nodos sensores en dirección a la aplicación móvil, pasando por el *router* de borde, los clientes MQTT en la Raspberry Pi, el *broker* privado y el *broker* público.

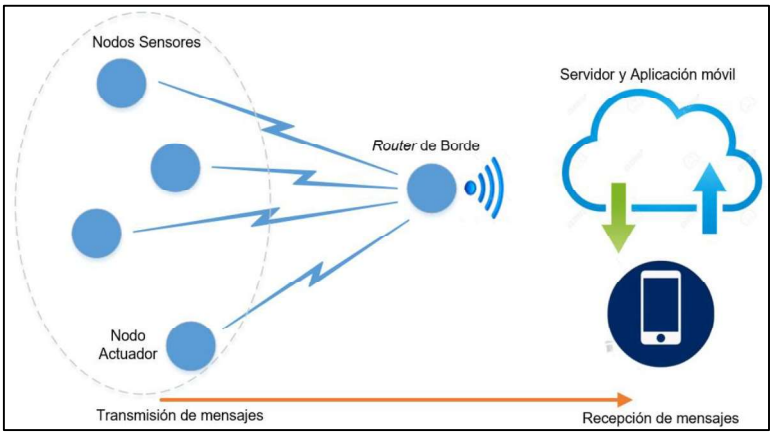



**Figura 3.31.** Escenario de prueba 1

Los mensajes que se transmiten desde la red de nodos de sensores son periódicos, de modo que la aplicación móvil recibirá varios mensajes de distintos nodos que serán listados en la interfaz móvil, de acuerdo al tiempo en el que son recibidos, tal como se muestra en la Figura 3.32.


**Figura 3.32.** Mensajes recibidos desde la red de nodos sensores

Estos mensajes son generados en las motas tomando los datos de los sensores de temperatura y humedad que luego son publicados en el tópico *zolertia/datos_sensor*, para posteriormente ser recibidos por un cliente MQTT y publicados por otro cliente MQTT en la Raspberry Pi, de modo que llega hasta la aplicación móvil mediante el *broker* público y el tópico *datos*.

Otro tipo de mensaje que la aplicación recibe es el que corresponde a la respuesta de la solicitud del usuario de conocer el nivel de agua en el pozo, mensaje que es recibido cuando el usuario elige presionar la opción *NIVEL DE AGUA*. Este mensaje se muestra dentro de la fila de mensajes que ha recibido de la red de nodos sensores, tal como se puede apreciar en la Figura 3.33. Este mensaje es transmitido por el nodo actuador que consta de una mota conectada a sensores de nivel de agua ubicados en el pozo.

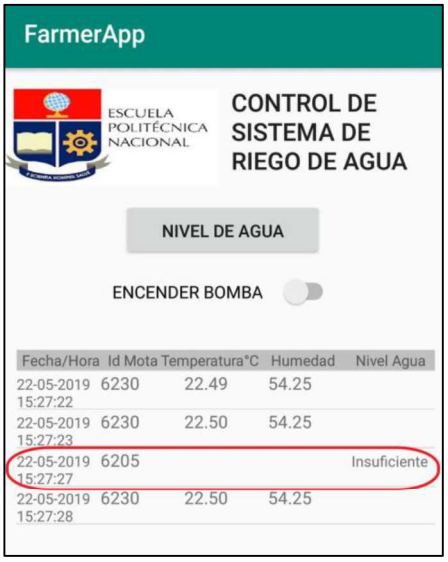



**Figura 3.33.** Mensaje recibido desde el nodo actuador

**b. Recepción de mensajes de la aplicación móvil en un nodo**

El siguiente escenario de pruebas se puede observar en la Figura 3.34., donde los mensajes son generados y transmitidos desde la aplicación móvil hacia el nodo actuador.

Los mensajes son publicados al *broker* público en el tópico *datos* y luego atraviesan dos clientes MQTT que se encuentran en la Raspberry Pi, para finalmente llegar hasta el nodo actuador que recibe los datos en el tópico *zolertia/cmd/bomb*.

El usuario final al presionar el botón *NIVEL DE AGUA* genera y publica un mensaje en el tópico ya mencionado, mensaje que luego es recibido por un cliente MQTT y publicado por otro en la Raspberry Pi y que finalmente llega al nodo actuador.

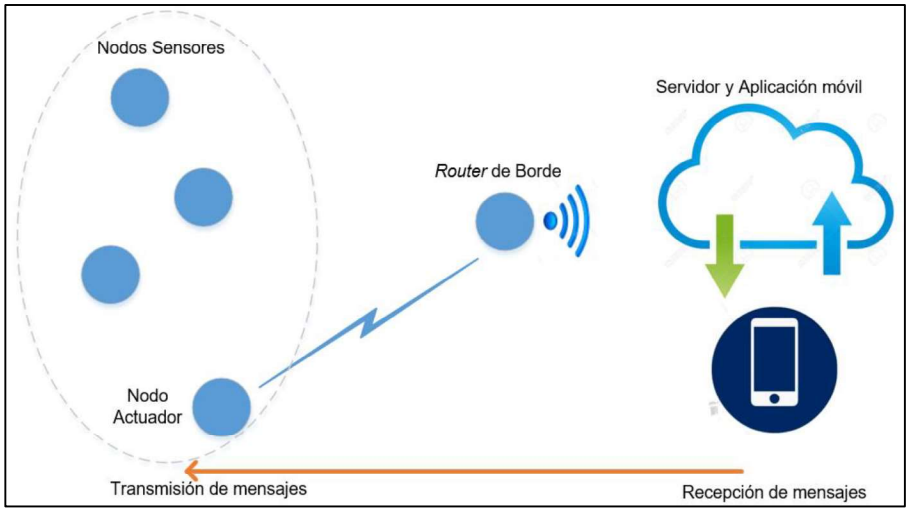



**Figura 3.34.** Escenario de prueba 2

La salida de la mota que trabaja como nodo actuador muestra los mensajes que se observan en la Figura 3.35. Esta mota primero indica que tiene una conexión MQTT, luego que se suscribe al tópico correspondiente para recibir mensajes publicados y luego que se ha recibido un mensaje, cuyo tamaño es de un bit y que solicita un informe del nivel de agua en el pozo.

```
Application has a MQTT connection
 Subscribing to zolertia/cmd/bomb
 Application is subscribed to topic successfully

Application received a publish on topic 'zolertia/cmd/bomb'. Payload size is 1 bytes. Content:
Pub Handler: topic='zolertia/cmd/bomb' (len=17), chunk_len=1
Informe ...
```

**Figura 3.35.** Mensaje recibido desde la aplicación móvil

Existen tres tipos de mensajes que pueden ser recibidos en el nodo actuador, los cuales son: mensaje para pedir informe del nivel de agua, mensaje para encender la bomba de regadío y mensaje para apagar la bomba de regadío.

### c. Accionamiento de la bomba de agua desde la aplicación móvil

El último escenario de pruebas es donde se prueba que, el usuario final mediante la aplicación móvil es capaz de comandar el encendido o apagado de la bomba de regadío.

En este escenario la información viajará de un extremo al otro, atravesando los *broker*s tanto público como privado y los clientes MQTT intermedios de la Raspberry Pi hasta llegar al nodo actuador, el cual accionará la bomba de agua con la ayuda de un convertidor de voltaje DC-DC MT3608 y un relé de estado sólido SSR-100 DA. Este escenario se puede apreciar en la Figura 3.36.

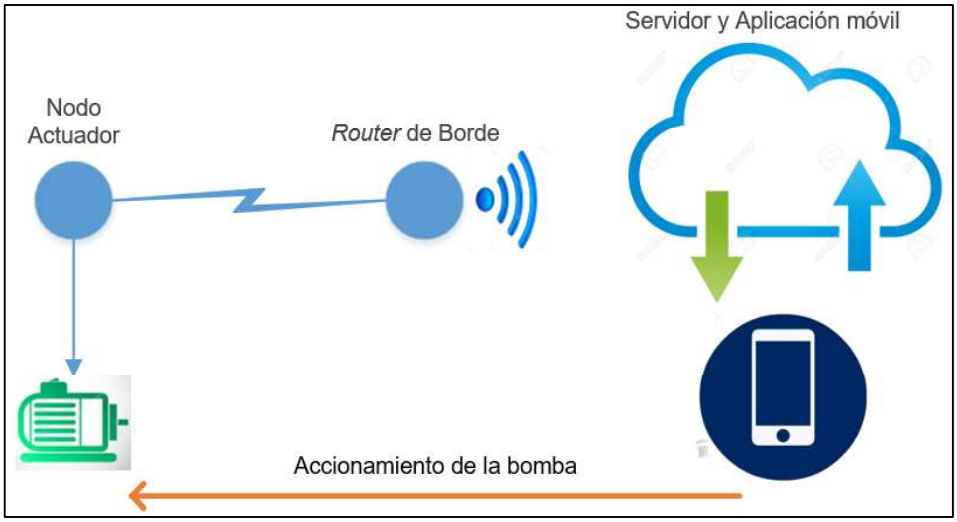



**Figura 3.36.** Escenario de prueba 3

El usuario final desde la aplicación presiona el botón Encender Bomba como se observa en la Figura 3.37., y en ese mismo instante se genera un mensaje que se publica en el tópico *command*.


**Figura 3.37.** Usuario presiona ENCENDER BOMBA

Luego de la transmisión de este mensaje, la mota suscrita al tópico *zolertia/cmd/bomb* recibe el mensaje, cuyo tamaño es de 1 bit, como se observa en la Figura 3.38., esta salida del nodo actuador también indica que el mensaje que se ha recibido es para encender la bomba de regadío.

```
Application received a publish on topic 'zolertia/cmd/bomb'. Payload size is 1 bytes. Content:
Pub Handler: topic='zolertia/cmd/bomb' (len=17), chunk_len=1
Turning bomb on!
```

**Figura 3.38.** Enciende la bomba de agua

Ahora, para apagar la bomba de regadío, el usuario desactiva el *switch* en la aplicación móvil, luego de lo cual también se genera un mensaje que es publicado inmediatamente y luego de viajar por dos *broker*s y dos clientes MQTT llega al nodo actuador, el cual lo interpreta y apaga la bomba de agua, como se observa en la Figura 3.39.

```
Application received a publish on topic 'zolertia/cmd/bomb'. Payload size is 1 bytes. Content:
Pub Handler: topic='zolertia/cmd/bomb' (len=17), chunk_len=1
Turning bomb off!
```

**Figura 3.39.** Apagado de la bomba de agua

Cuando la bomba se ha encendido, el agua será direccionada mediante tubos hasta el lugar de cultivo, y el proceso de riego de agua se efectuará, como se observa en la Figura 3.40.

**Figura 3.40.** Proceso de riego de agua

## 3.3. ANÁLISIS DE RESULTADOS

Se han realizado pruebas de funcionamiento del sistema, las mismas se pueden observar en el Anexo B, de las cuales se ha llegado a deducir lo siguiente:

- Los valores de temperatura y humedad que se han adquirido en distintos tiempos, considerando los estados del suelo son los que se muestran en la Tabla 3.3. Los datos son tomados de un cultivo de brócoli en etapa de desarrollo, tomando en cuenta que la textura del suelo es franco.

**Tabla 3.3.** Valores de Humedad y Temperatura

| Estado del suelo | Humedad | Temperatura |
|---|---|---|
| Punto de marchitez, necesita riego de agua | 40 % o menor | 30 °C en adelante |
| Saturación, no necesita riego de agua | 80 % en adelante | 20 °C en adelante |
| Capacidad de campo, no necesita riego | 60 % en adelante | 20 °C en adelante |

- El tiempo que tarda el proceso de riego completo del área de cultivo con un sembrío de brócoli depende de dos factores, como son la etapa del cultivo y el nivel de humedad. Esto se puede ver en la Tabla 3.4.

**Tabla 3.4.** Tiempos de riego según Etapa y Nivel de humedad

| Etapa | Nivel de humedad | Tiempo |
|---|---|---|
| Plantación | 30 % | 3 horas y media |
| Crecimiento | 40 % | 2 horas y media |
| Desarrollo | 40 % | 2 horas |

- La bomba de regadío generalmente se enciende dos veces al día, debido a que el agua en el pozo demora alrededor de 4 horas para volver al nivel Suficiente. Sin embargo, también es posible proceder con el riego de agua cuando el nivel es Insuficiente, en este caso el tiempo de riego será menor comparado con un riego realizado cuando el nivel de agua es Suficiente.
- De la Tabla B.3. del Anexo B, se extraen las Figuras 3.41 y 3.42, en las que se muestran los valores de humedad y temperatura tomadas por dos motas ubicadas en distintas partes de un mismo cultivo.

  Los parámetros de temperatura y humedad serán diferentes al realizarse el proceso de riego de agua, debido a que los sensores se encuentran ubicados en distintos lugares en el área de cultivo.

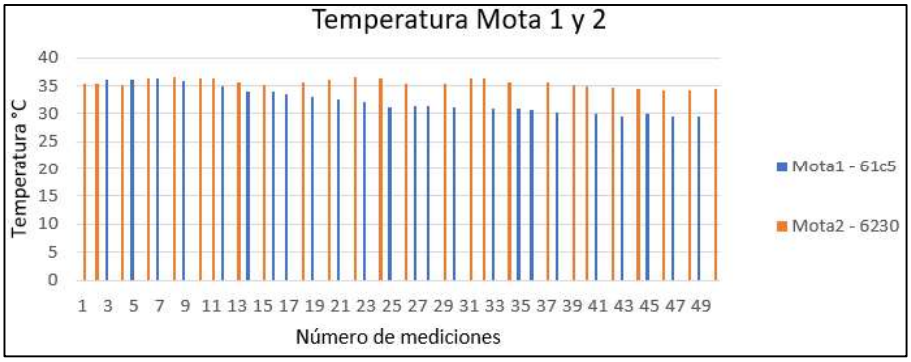



**Figura 3.41.** Resultado de pruebas de Temperatura

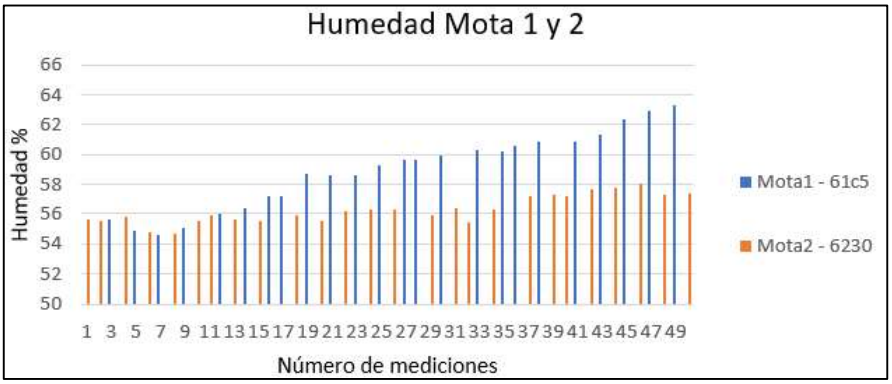



**Figura 3.42.** Resultado de pruebas de Humedad

Los valores de humedad y temperatura son similares al principio debido a que el proceso de riego aun no inicia; luego de haber iniciado el riego, los valores tanto de humedad como temperatura cambian en el sector donde se ubica la mota 61c5, porque el agua ya ha llegado a dicho sector, sin embargo, en el sector donde se encuentra la mota 6230 no existen cambios relevantes porque el agua aún no ha llegado hasta ese lugar. Los resultados de humedad y temperatura serán similares cuando el agua llegue hasta donde éstos se encuentran. De modo que el usuario sabrá que el riego de agua se ha realizado en todo el cultivo, cuando los sensores proporcionen valores similares, tomando en cuenta que la humedad y temperatura hayan llegado a valores comprendidos entre los estados, capacidad de campo y saturación.

- La cantidad de nodos sensores ubicados en el área de cultivo es limitado debido a la disponibilidad; sin embargo, el número de sensores que se deben colocar en un cultivo depende de la topografía y del área del terreno [40]. Tomando en cuenta esta información, en este cultivo son necesarios alrededor de 6 sensores.

  Al aumentar los sensores, el usuario tendrá un mayor conocimiento del estado de todo el cultivo.

- De acuerdo a las Tablas B.1 y B.2 del Anexo B, los niveles de agua en el pozo tomadas durante varias horas, no cambian rápidamente debido a que solo existen tres niveles que se indican gracias a dos sensores de nivel discretos. Sin embargo, debido a que el nivel máximo de agua en el pozo es de aproximadamente 4 metros y cuya forma circular tiene un radio de 2 metros, la información que brindan los sensores de nivel al usuario no es tan minuciosa.

### 3.3.1. PRESUPUESTO DEL PROTOTIPO Y COMPARACIÓN

El presupuesto del proyecto abarca la parte de software y hardware empleados en la implementación del proyecto completo. En la Tabla 3.5 se describen los dispositivos utilizados, sus costos y el costo total de la parte de hardware.

**Tabla 3.5.** Costo del hardware del proyecto

| Cant. | Descripción | V. Unit. | TOTAL |
|---|---|---|---|
| 3 | Sensores de humedad y temperatura | $ 12.00 | $ 36.00 |
| 1 | Kit Zolertia RE-Mote | $ 500.00 | $ 500.00 |
| 1 | Raspberry Pi | $ 50.00 | $ 50.00 |
| 1 | Equipo de automatización | $ 20.00 | $ 20.00 |
| **TOTAL (incluye IVA)** | | | $ 600.00 |

El costo del desarrollo de software de todo el proyecto es de $ 480.00, tomando en consideración que las horas de trabajo totales fueron de 60 y el costo por hora de trabajo es de $ 8.00.

Es así que, en total, el costo de la implementación del sistema de riego de agua automatizado y control remoto es $ 1 080.00. El costo del desarrollo es alto porque el trabajo es un prototipo, sin embargo, en las siguientes implementaciones el costo será más económico. La implementación de este proyecto en comparación con otros sistemas de riego automatizado que algunas empresas ofertan en el país es más económico y completo.

Entre las empresas investigadas que ofrecen sistemas de automatización de riego de agua están:

- **NETAFIM:** Dentro de las opciones que brinda esta empresa, se encuentra la sección de Equipos de automatización [41], en donde uno de los equipos para automatizar el riego de agua cuesta alrededor de $ 1 000.00. Este costo debería

ser sumado al costo de los demás equipos necesarios para el sistema de riego de agua y también al costo de la instalación del sistema.

- **RIEGO ECUADOR:** Una de las secciones que presenta esta empresa en su página oficial, es Automatización [42], que presenta los equipos que están a disposición para automatizar el riego, donde el costo del equipo a utilizar en un sistema de riego de agua sin utilizar sensores está alrededor de los $ 500.00. A este precio es preciso añadir el costo de instalación y mantenimiento, de modo que su precio irá aumentando acorde también a las dimensiones del terreno.

# 4. CONCLUSIONES Y RECOMENDACIONES

## 4.1. CONCLUSIONES

- El uso de la plataforma Zolertia RE-Mote en el desarrollo de esta aplicación IoT ha presentado varias ventajas; una de ellas es que provee una respuesta en procesamiento rápida, ya que a pesar de que la información debe pasar por varios puntos de conexión hasta llegar a la aplicación móvil, el usuario final puede acceder a los datos tan pronto como son transmitidos desde la plataforma, brindando también una menor latencia en la comunicación. Otra de las grandes ventajas es que gracias a que provee comunicación entre nodos a grandes distancias, se puede utilizar una menor cantidad de nodos para el despliegue en un área grande. Además, el consumo de energía de esta plataforma es bajo, lo cual ha permitido el uso de baterías como fuente de alimentación para su funcionamiento.

- El proyecto ha sido desarrollado tomando como base el enfoque del Internet de las Cosas; esto es, toma las propiedades físicas de los objetos del mundo real, mediante los sensores, y a través de dispositivos electrónicos, las motas, se conectan a Internet para posteriormente interactuar con los objetos, esto es el control de la bomba de regadío para dar lugar al proceso de riego de agua.

- Parte del desarrollo del software de este proyecto se ha simplificado debido a que se cuenta con un *broker* público disponible en la Nube, tecnología conocida como Cloud Computing. Esta es la razón por la cual, es vital una conexión a Internet estable.

- El despliegue de este prototipo en otros terrenos se podrá realizar mediante un análisis previo de las características del área de cultivo, como el tipo de suelo, la topografía que posee, entre otras.

- Para esta aplicación, el uso de los dos *brokers* tanto el público como el privado es indispensable, porque gracias al privado se tiene salida al Internet mientras que con el público se brinda el control remoto del sistema.

- El diseño, desarrollo y creación del aplicativo móvil se podría realizar mediante otros entornos de programación distintos a Android Studio, a fin de facilitar su implementación. Algunos de los entornos que se podrían utilizar son Basic4Android, App Inventor, Appcelerator Titanium, etc.

- Gracias al uso de la metodología Kanban en este proyecto, se han organizado adecuadamente las diferentes tareas que al ser desarrolladas según la prioridad que presenta cada una, se ha logrado el buen término del proyecto. Sin embargo, existen otras metodologías que podrían utilizarse en el desarrollo de proyectos de este tipo, metodologías como SCRUM, XP, RAD, entre otras.

- Las diferentes herramientas provistas por Eclipse Paho en distintos lenguajes de programación han facilitado la creación de los clientes MQTT y su conexión, mediante la fuente de código disponible en sus repositorios, además de la flexibilidad para escoger el utilizar una u otra.

- Existe una amplia diversidad de sensores comerciales que están a disposición de un desarrollador, sin embargo, es importante escoger los sensores que más se adecúen al escenario de implementación, basándose en las características que presentan.

- En el presente proyecto se ha aprovechado de las grandes ventajas que provee el sistema operativo Contiki, gracias a los archivos ejemplo usados como base tanto para el estudio como para la implementación de programas en las motas, además de la facilidad de compilación y ejecución de software.

## 4.2. RECOMENDACIONES

- En un futuro proyecto que puede tomar como base este prototipo, se recomienda automatizar el proceso de riego, tomando como base los parámetros de humedad y temperatura, de modo que, cuando en un cultivo se alcancen valores determinados de humedad y temperatura, instantáneamente se encienda la bomba de regadío.

- Debido a que el alcance de las antenas es limitado, y puede ser extendido dependiendo de la antena que se utilice y de la potencia de transmisión con que se trabaje en la plataforma Zolertia RE-Mote, se recomienda tomar en consideración ambas vías para un despliegue mayor de la red de nodos de sensores.

- La comunicación MQTT ha sido vital para la implementación del prototipo, donde una de las entidades importantes es el *broker* porque gestiona el envío de mensajes, por este motivo se recomienda que de los *brokers* públicos disponibles en la red, se escoja aquel cuya disponibilidad sea alta y estable.

- La corriente de arranque que usa la bomba generalmente es mucho mayor a la corriente nominal, por lo cual, se recomienda que el relé de estado sólido a emplear para controlar remotamente su accionamiento, soporte una corriente mayor a la corriente nominal de la bomba de agua.
- La información de nivel de agua obtenida gracias a los sensores de nivel discretos podría ser más explícita, por lo que se recomienda utilizar más sensores discretos ubicados en niveles más cercanos o también se pueden utilizar otros tipos de sensores de nivel de líquido que sean más explícitos en sus lecturas.
- El proceso de riego de agua en los cultivos es diferente dependiendo del tipo de cultivo, es por esto que, se recomienda analizar el tipo de cultivo para determinar los valores de humedad y temperatura y vincularlos con su necesidad de riego de agua.
- Se recomienda diseñar en un futuro un mecanismo de alimentación para las plataformas Zolertia RE-Mote a través de paneles solares, debido a que éstas deben estar alimentadas ininterrumpidamente y, su despliegue es realizado en un campo de cultivo.
- La actividad del sistema diseñado es continua por lo que, como medida de seguridad se recomienda utilizar un sistema de respaldo de energía que se active cuando ocurra alguna interrupción en el aspecto energético.
- Debido a que la implementación de este proyecto se ha realizado para la plataforma Android, se recomienda desarrollarlo también para la plataforma MAC, de modo que su uso se extienda a una mayor cantidad de clientes.
- El voltaje que ofrece la plataforma Zolertia RE-Mote puede no ser suficiente para el correcto funcionamiento de un actuador, es por esta razón que se recomienda utilizar un convertidor de voltaje que aumente el voltaje de la mota hasta el valor necesario para activar un actuador.
- Debido a la gran diversidad de herramientas disponibles en diferentes lenguajes de programación que provee el proyecto Eclipse Paho, se recomienda utilizar aquellas que sean más conocidas o sean más fáciles de utilizar.

# 5. REFERENCIAS BIBLIOGRÁFICAS


[1] MAGAP, *La Política Agropecuaria Ecuatoriana*. 2015.

[2] V. Tsiatsis, S. Karnouskos, J. Höller, D. Boyle, and C. Mulligan, "Why the Internet of Things?," in *Internet of Things*, Elsevier, 2019, pp. 3–7.

[3] "Digital Transformation - Reports - World Economic Forum," *World Economic Forum*, 2018. [Online]. Available: http://reports.weforum.org/digital-transformation/the-internet-of-things-and-connected-devices-making-the-world-smarter/. [Accessed: 08-Jan-2019].

[4] "¿Realidad virtual o Internet de las cosas? |." [Online]. Available: https://es.ihodl.com/investment/2016-10-27/realidad-virtual-o-internet-de-las-cosas/. [Accessed: 17-May-2019].

[5] "Internet of Things IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps EUROPEAN RESEARCH CLUSTER ON THE INTERNET OF THINGS," 2015.

[6] A. Tzounis, N. Katsoulas, T. Bartzanas, and C. Kittas, "Internet of Things in agriculture, recent advances and future challenges," *Biosyst. Eng.*, vol. 164, pp. 31–48, Dec. 2017.

[7] FAO, "La agricultura mundial en la perspectiva del año 2050," 2009.

[8] F. Edwards Murphy, E. Popovici, P. Whelan, and M. Magno, "Development of an heterogeneous wireless sensor network for instrumentation and analysis of beehives," in *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, 2015, pp. 346–351.

[9] D. Navani, S. Jain, and M. S. Nehra, "The Internet of Things (IoT): A Study of Architectural Elements," in *2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, 2017, pp. 473–478.

[10] "IEEE 802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks," *standards.ieee.org*, 2019. [Online]. Available: https://standards.ieee.org/content/ieee-standards/en/standard/802_15_4-2015.html. [Accessed: 14-Jan-2019].

[11] A. Tzounis, N. Katsoulas, T. Bartzanas, and C. Kittas, "Internet of Things in

agriculture, recent advances and future challenges," *Biosyst. Eng.*, vol. 164, pp. 31–48, 2017.

[12] R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch," Nov. 2016.

[13] MQTT.org, "Frequently Asked Questions | MQTT." [Online]. Available: http://mqtt.org/faq. [Accessed: 17-May-2019].

[14] OASIS, "About Us | OASIS," 2019. [Online]. Available: https://www.oasis-open.org/org. [Accessed: 17-May-2019].

[15] "IoT based Real time data acquisition using MQTT Protocol," 2018.

[16] S. Cope, "Understanding the MQTT Protocol Packet Structure," 2019. [Online]. Available: http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/. [Accessed: 17-May-2019].

[17] IBM Eurotech, "MQTT V3.1 Protocol Specification." [Online]. Available: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html. [Accessed: 17-May-2019].

[18] "Eclipse Paho | projects.eclipse.org," *eclipse.org*, 2018. [Online]. Available: https://projects.eclipse.org/projects/technology.paho. [Accessed: 26-Dec-2018].

[19] "Eclipse Paho - MQTT and MQTT-SN software," 2019. [Online]. Available: https://www.eclipse.org/paho/. [Accessed: 17-May-2019].

[20] "MQTT C Client," *eclipse.org*, 2018. [Online]. Available: https://www.eclipse.org/paho/clients/c/. [Accessed: 25-Dec-2018].

[21] "Eclipse Paho Android Service," *Eclipse.org*, 2018. [Online]. Available: https://www.eclipse.org/paho/clients/android/. [Accessed: 25-Dec-2018].

[22] "RE-MOTE SUITE - Zolertia," *Zolertia*, 2018. [Online]. Available: https://zolertia.io/product/re-mote-suite/. [Accessed: 17-Jan-2019].

[23] Zolertia, "Zolertia RE-Mote Revision B," no. September, pp. 1–5, 2016.

[24] "The Contiki Community," *Contiki*, 2018. [Online]. Available: http://www.contiki-os.org/community.html. [Accessed: 17-Jan-2019].

[25] Y. Bin Zikria, M. K. Afzal, F. Ishmanov, S. W. Kim, and H. Yu, "A survey on routing protocols supported by the Contiki Internet of things operating system," *Futur.*

*Gener. Comput. Syst.*, vol. 82, pp. 200–219, May 2018.

[26] A. Liñán, C. Alvaro, V. A. Bagula, M. Zennaro, and E. Pietrosemoli, “Internet of Things in 5 Days,” no. March, pp. 1–177, 2015.

[27] “Raspberry Pi 3 Model B - Raspberry Pi,” *www.raspberrypi.org*, 2019. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 05-Feb-2019].

[28] Consorcio de Gobiernos Autónomos Provinciales del Ecuador, “HABLEMOS DE RIEGO CON LOS AGRICULTORES,” Quito, 2016.

[29] “El Suelo,” *Organización de las Naciones Unidas para la Alimentación*, 2018. [Online]. Available: http://www.fao.org/docrep/006/w1309s/w1309s04.htm. [Accessed: 22-Jan-2019].

[30] “Los Tipos de Riego y sus ventajas,” *Sistema Agrícola - México*, 2018. [Online]. Available: http://sistemaagricola.com.mx/blog/tipos-de-riego-en-la-agricultura-y-ventajas/. [Accessed: 22-Jan-2019].

[31] *Plan Nacional de Riego y Drenaje 2011-2026*. Ecuador, 2011.

[32] L. Germán Corona Ramírez, G. S. Abarca Jiménez, and J. Mares Carreño, *Sensores y actuadores.* Larousse - Grupo Editorial Patria, 2014.

[33] “Módulo De Relé De Estado Sólido | DHgate.Com,” 2019. [Online]. Available: https://es.dhgate.com/product/wholesale-1pcs-24v-380v-100a-250v-ssr-100/394942989.html. [Accessed: 20-May-2019].

[34] T. Sensor, “Datasheet SHT25 Humidity and Temperature Sensor Fully calibrated with 1 . 8 % RH accuracy Digital output , I 2 C interface Low power consumption Excellent long term stability DFN type package – reflow solderable D0AC4,” *Response*, no. December, pp. 1–12, 2010.

[35] “SHT25 Humidity and Temperature Sensor,” 2019. [Online]. Available: https://shop.controleverything.com/products/humidity-and-temperature-sensor-1-8-rh-0-2-c-1. [Accessed: 20-May-2019].

[36] “Sensor de nivel de agua horizontal— ElectroCrea,” 2019. [Online]. Available: https://electrocrea.com/products/sensor-de-nivel-de-agua-horizontal. [Accessed: 20-May-2019].

[37] “Sensores de Agua, Flujo y Nivel - ABC Proyectos Electrónicos,” 2019. [Online].

Available: https://www.abcelectronica.net/productos/sensores/flujo-agua/. [Accessed: 20-May-2019].

[38] “Conoce Android Studio | Android Developers,” *android.com*, 2018. [Online]. Available: https://developer.android.com/studio/intro/?hl=es-419. [Accessed: 26-Dec-2018].

[39] Rong Long, “Bomba de agua agrícola 5HP W125275,” 2019. [Online]. Available: https://ronglong.com.ec/bombas-agua/w125275-detail. [Accessed: 24-May-2019].

[40] C. Schugurensky and F. Capraro, “Control Automático de Riego Agrícola con Sensores Capacitivos de Humedad de Suelo. Aplicaciones en Vid y Olivo.”

[41] “NETAFIM,” 2019. [Online]. Available: http://netshop.netafim.ec/. [Accessed: 08-May-2019].

[42] “Riego Ecuador,” 2019. [Online]. Available: https://www.riegoecuador.com/index.php/categorias/automatizacion. [Accessed: 08-May-2019].

# ANEXOS

ANEXO A. Modelo de encuesta para levantamiento de información

ANEXO B. Pruebas de funcionamiento del prototipo

ANEXO C. Código del proyecto

# ANEXO A

## Modelo de encuesta para levantamiento de información

**Esta entrevista tiene como único fin ser parte del desarrollo de un Proyecto de Titulación en la Escuela Politécnica Nacional. Por favor lea atentamente las preguntas y responda con toda sinceridad.**

Encierre en un círculo su respuesta o complete.

1. ¿Qué parámetros o qué condiciones toma en cuenta para realizar el riego de agua?
    a) Temperatura del ambiente
    b) Humedad del suelo
    c) Lluvia
    d) Etapa del desarrollo de la planta
    e) Época del año
    f) Otros:

    _____
2. ¿Cuál es el tipo de riego de agua que más utiliza?
    a) Riego por inundación o gravedad
    b) Riego por goteo
    c) Riego por aspersión
    d) Otros:

    _____
3. ¿Ud. ¿Tiene pozo para el almacenamiento de agua de regadío?
    Si ☐
    No ☐
4. ¿El espacio de tiempo transcurrido entre dos riegos de agua depende de la necesidad del cultivo?
    Si ☐
    No ☐
5. ¿Cuál es la disponibilidad de agua que tiene por día?
    Entre 1 y 2 horas
    Entre 2 y 4 horas
    Otros: _____
6. ¿Tiene instalado algún sistema de riego automatizado?

    Si ☐

No ☐

7. ¿Considera Ud. que es necesario controlar remotamente el encendido y apagado de la bomba de agua, teniendo como datos la temperatura y humedad del suelo?

   Si ☐

   No ☐

8. ¿Qué tecnologías le interesaría que abarque un sistema de riego automatizado?

   a) Control mediante el teléfono celular
   b) Que se pueda visualizar información del cultivo (temperatura, humedad, otros) en el momento que está funcionando el sistema de riego
   c) Control a distancia del accionamiento de la bomba de regadío, de acuerdo a las necesidades
   d) Otros: _____

9. ¿Le interesaría disponer una instalación de un sistema de riego automatizado? Si su respuesta es afirmativa continúe con la pregunta 10.

   Si ☐

   No ☐

10. ¿Cuánto estaría dispuesto a pagar por tener un sistema de riego automatizado?

    a) $ 100
    b) $ 150
    c) $ 200
    d) No estaría dispuesto a pagar nada

# ANEXO B

## Pruebas de funcionamiento del prototipo

Las pruebas de funcionamiento de la Tabla B.1. se tomaron en un día. Las pruebas fueron realizadas aproximadamente cada 10 minutos.

**Tabla B.1.** Pruebas de funcionamiento día 1

| Núm. Prueba | ID Mota | Comando | Recepción en Cliente1 | Publicación Cliente 4 | Mensaje en Aplicación | Bomba |
|---|---|---|---|---|---|---|
| 1 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 2 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 3 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 4 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 5 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 6 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 7 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 8 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 9 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 10 | 6205 | 1 | Si | Si | Suficiente | Encendida |
| 11 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 12 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 13 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 14 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 15 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 16 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 17 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 18 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 19 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 20 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 21 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 22 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 23 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 24 | 6205 | 2 | Si | Si | Escaso | Encendida |
| 25 | 6205 | 2 | Si | Si | Escaso | Encendida |
| 26 | 6205 | 0 | Si | Si | Escaso | Apagada |

| | | | | | | |
|---|---|---|---|---|---|---|
| 27 | 6205 | 2 | Si | Si | Escaso | Apagada |
| 28 | 6205 | 2 | Si | Si | Escaso | Apagada |
| 29 | 6205 | 2 | Si | Si | Escaso | Apagada |

Las pruebas de funcionamiento de la Tabla B.2. se realizaron en el día 2. Las pruebas se realizaron aproximadamente cada 15 minutos

**Tabla B.2.** Pruebas de funcionamiento día 2

| **Núm. Prueba** | **ID Mota** | **Comando** | **Recepción en Cliente1** | **Publicación Cliente 4** | **Mensaje en Aplicación** | **Bomba** |
|---|---|---|---|---|---|---|
| 1 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 2 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 3 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 4 | 6205 | 1 | Si | Si | Insuficiente | Encendida |
| 5 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 6 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 7 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 8 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 9 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 10 | 6205 | 2 | Si | Si | Escaso | Encendida |
| 11 | 6205 | 2 | Si | Si | Escaso | Encendida |
| 12 | 6205 | 0 | Si | Si | Escaso | Apagada |
| 13 | 6205 | 2 | Si | Si | Escaso | Apagada |
| 14 | 6205 | 2 | Si | Si | Escaso | Apagada |
| 15 | 6205 | 2 | Si | Si | Escaso | Apagada |
| 16 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 17 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 18 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 19 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 20 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 21 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 22 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 23 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 24 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 25 | 6205 | 2 | Si | Si | Insuficiente | Apagada |

| | | | | | | |
|---|---|---|---|---|---|---|
| 26 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 27 | 6205 | 2 | Si | Si | Insuficiente | Apagada |
| 28 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 29 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 30 | 6205 | 2 | Si | Si | Suficiente | Apagada |
| 31 | 6205 | 1 | Si | Si | Suficiente | Encendida |
| 32 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 33 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 34 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 35 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 36 | 6205 | 2 | Si | Si | Suficiente | Encendida |
| 37 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 38 | 6205 | 2 | Si | Si | Insuficiente | Encendida |
| 39 | 6205 | 0 | Si | Si | Insuficiente | Apagada |

En la Tabla B.3. se muestran los valores de humedad y temperatura adquiridos en un día de pruebas. Las mediciones son periódicas.

**Tabla B.3.** Mediciones de Temperatura y Humedad mediante proceso de riego

| **# Medición** | **ID Mota** | **Temperatura °C** | **Humedad %** | **Bomba** |
|---|---|---|---|---|
| 1 | 6230 | 35.33 | 55.62 | Apagada |
| 2 | 6230 | 35.34 | 55.57 | Apagada |
| 3 | 61c5 | 36.02 | 55.67 | Apagada |
| 4 | 6230 | 35.03 | 55.80 | Apagada |
| 5 | 61c5 | 36.14 | 54.89 | Apagada |
| 6 | 6230 | 36.23 | 54.75 | Apagada |
| 7 | 61c5 | 36.33 | 54.60 | Encendida |
| 8 | 6230 | 36.52 | 54.71 | Encendida |
| 9 | 61c5 | 35.85 | 55.08 | Encendida |
| 10 | 6230 | 36.34 | 55.50 | Encendida |
| 11 | 6230 | 36.35 | 55.89 | Encendida |
| 12 | 61c5 | 34.98 | 55.97 | Encendida |
| 13 | 6230 | 35.67 | 55.63 | Encendida |
| 14 | 61c5 | 34.05 | 56.36 | Encendida |
| 15 | 6230 | 35.21 | 55.57 | Encendida |

| | | | | |
|---|---|---|---|---|
| 16 | 61c5 | 34.02 | 57.24 | Encendida |
| 17 | 61c5 | 33.50 | 57.21 | Encendida |
| 18 | 6230 | 35.57 | 55.95 | Encendida |
| 19 | 61c5 | 33.12 | 58.68 | Encendida |
| 20 | 6230 | 36.05 | 55.54 | Encendida |
| 21 | 61c5 | 32.52 | 58.65 | Encendida |
| 22 | 6230 | 36.50 | 56.21 | Encendida |
| 23 | 61c5 | 32.15 | 58.59 | Encendida |
| 24 | 6230 | 36.40 | 56.24 | Encendida |
| 25 | 61c5 | 31.24 | 59.26 | Encendida |
| 26 | 6230 | 35.47 | 56.32 | Encendida |
| 27 | 61c5 | 31.45 | 59.65 | Encendida |
| 28 | 61c5 | 31.45 | 59.69 | Encendida |
| 29 | 6230 | 35.46 | 55.95 | Encendida |
| 30 | 61c5 | 31.06 | 59.98 | Encendida |
| 31 | 6230 | 36.32 | 56.37 | Encendida |
| 32 | 6230 | 36.32 | 55.45 | Encendida |
| 33 | 61c5 | 30.86 | 60.32 | Encendida |
| 34 | 6230 | 35.50 | 56.32 | Encendida |
| 35 | 61c5 | 30.96 | 60.25 | Encendida |
| 36 | 61c5 | 30.78 | 60.58 | Encendida |
| 37 | 6230 | 35.60 | 57.23 | Encendida |
| 38 | 61c5 | 30.17 | 60.87 | Encendida |
| 39 | 6230 | 35.03 | 57.35 | Encendida |
| 40 | 6230 | 34.99 | 57.24 | Encendida |
| 41 | 61c5 | 30.01 | 60.84 | Encendida |
| 42 | 6230 | 34.68 | 57.67 | Encendida |
| 43 | 61c5 | 29.64 | 61.32 | Encendida |
| 44 | 6230 | 34.53 | 57.82 | Encendida |
| 45 | 61c5 | 29.99 | 62.35 | Encendida |
| 46 | 6230 | 34.32 | 58.06 | Encendida |
| 47 | 61c5 | 29.63 | 62.95 | Encendida |
| 48 | 6230 | 34.26 | 57.32 | Encendida |
| 49 | 61c5 | 29.47 | 63.35 | Encendida |
| 50 | 6230 | 34.50 | 57.39 | Encendida |

## ORDEN DE EMPASTADO