UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE INGENIERIA DE SISTEMAS

Trabajo de Titulación previo a la obtención del Título de Ingeniera de Sistemas

PROYECTO TÉCNICO:

"DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UNA ESTACIÓN METEOROLÓGICA BASADA EN UNA RED JERÁRQUICA DE SENSORES, SOFTWARE LIBRE Y SISTEMAS EMBEBIDOS PARA LA EMPRESA ELECAUSTRO EN LA MINICENTRAL GUALACEO UTILIZANDO COMUNICACIÓN MQTT Y MODBUS"

Autora:

Sonia Isabel Palaguachi Encalada

Tutor:

Ing. Erwin Jairo Sacoto Cabrera, MSc

Cuenca – Ecuador

2018

CESIÓN DE DERECHOS DE AUTOR

Yo, Sonia Isabel Palaguachi Encalada con documento de identificación No.0103882411, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud que soy autora del trabajo de titulación "DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UNA ESTACIÓN METEOROLÓGICA BASADA EN UNA RED JERÁRQUICA DE SENSORES, SOFTWARE LIBRE Y SISTEMAS EMBEBIDOS PARA LA EMPRESA ELECAUSTRO EN LA MINICENTRAL GUALACEO UTILIZANDO COMUNICACIÓN MQTT Y MODBUS", mismo que ha sido desarrollado para optar por el título de Ingeniera de Sistemas, en la Universidad Politécnica Salesiana quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

Aplicando a los determinados en la ley de Propiedad Intelectual, en mi condición de autora me reservo los derechos morales de la obra antes citada. En concordancia, suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, agosto de 2018

Sonia Isabel Palaguachi Encalada.

C.I 0103882411

CERTIFICACIÓN

Yo, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación "DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UNA ESTACIÓN METEOROLÓGICA BASADA EN UNA RED JERÁRQUICA DE SENSORES, SOFTWARE LIBRE Y SISTEMAS EMBEBIDOS PARA LA EMPRESA ELECAUSTRO EN LA MINICENTRAL GUALACEO UTILIZANDO COMUNICACIÓN MQTT Y MODBUS", realizado por Sonia Isabel Palaguachi Encalada, obteniendo el Proyecto Técnico que cumple con todos los requisitos estipulados por la universidad.

Cuenca, agosto de 2018

TUTOR DEL TRABAJO DE TITULACIÓN

DECLARATORIA DE RESPONSABILIDAD

Yo, Sonia Isabel Palaguachi Encalada con número de cédula 0103882411, autora del trabajo de titulación: "DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UNA ESTACIÓN METEOROLÓGICA BASADA EN UNA RED JERÁRQUICA DE SENSORES, SOFTWARE LIBRE Y SISTEMAS EMBEBIDOS PARA LA EMPRESA ELECAUSTRO EN LA MINICENTRAL GUALACEO UTILIZANDO COMUNICACIÓN MQTT Y MODBUS", certifico que el total contenido del Proyecto Técnico es de mi exclusiva responsabilidad y autoría.

Cuenca, agosto de 2018

Sonia Isabel Palaguachi Encalada.

C.I. 0103882411

AGRADECIMIENTO.

Gracias a la Universidad Politécnica Salesiana, abrirme las puertas y darme la oportunidad de poder estudiar una carrera, a mis docentes por haberme brindado sus conocimientos y apoyo para culminación de la misma.

A mi director de Tesis Magister. Jairo Sacoto por haberme brindado la oportunidad de recurrir a sus conocimientos, por toda la paciencia y comprensión que me ha tenido durante la realización de este trabajo de titulación.

A ELECAUSTRO S.A por haberme aceptado realizar la tesis en esta prestigiosa Empresa.

DEDICATORIA.

Este trabajo de titulación se la dedico a mi amado esposo Marcelo Leonardo Gomezcoello Salinas, por su paciencia, sacrificio y esfuerzo, por creer en mi capacidad, y ayuda para culminar mi carrera, y aunque habido momentos difíciles siempre ha estado junto a mí. A mis hijos por ser mi motivación para seguirme superando. A mis padres y hermanos por su apoyo incondicional, y por consejos brindados.

Sonia Isabel Palaguachi Encalada.

PALABLAS CLAVE

Meteorología: Disciplina que se ocupa del estudio de los fenómenos atmosféricos.

Arduino: Plataforma de hardware libre, basada en una placa con un microcontrolador.

Raspberry pi: Es un pequeño ordenador con una enorme flexibilidad y simplicidad.

Sistemas Embebidos: Diseño computacional realizado para funciones dedicadas en un sistema que funciona tiempo real.

Sensor: Dispositivo capaz de convertir magnitudes físicas o químicas, en magnitudes eléctricas.

MODBUS: Protocolo de comunicaciones que permite el control de una red de dispositivos.

MQTT: Message Queue Telemetry Transport, protocolo IoT que se especializa a la entrega y recepción de mensajes. Es decir, con la publicación y subscripción de mensajes.

INDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente proyecto consiste en el diseño e implementación de una estación meteorológica basada en el uso de sistemas embebidos tales como Raspberry Pi y Arduino Yun, conectados inalámbricamente mediante la comunicación MQTT y MODBUS, con lo cual se monitorea en tiempo real las diferentes variables climáticas, con la opción de consultas de los históricos.

La estación esta conforma por un Arduino Yun, se encarga de monitorear las variables climáticas mediante sus entradas analógicas, digitales y de comunicación I2C, receptando los diferentes valores de los sensores de humedad, temperatura, velocidad del viento (anemómetro), dirección del viento (veleta), pluviómetro e irradiación solar.

Por otra parte, para cumplir las funciones de un servidor se incorpora un Raspberry Pi con su sistema operativo Raspbian, en el cual, mediante programación realizada en Java, que contiene librerías propias del MQTT, almacena las variables en una base de datos MySql.

La base de datos se usa como enlace entre los sensores y el servidor web, este servidor fue ejecutado en la Raspberry Pi, mediante Apache Tomcat, que es usado como un servidor proxy, el cual permite acceder a los recursos, enrutando todo el tráfico (consultas en la WEB) al servidor, por la necesidad de que la aplicación y la página WEB deben ser mostradas fuera de la red privada.

La generación de página WEB se las realizo mediante el programa Web Ratio, las cuales se actualizan en tiempo real de forma periódica y son mostrados mediante tabla y graficas en la página WEB emeteo.ddns.net. Mediante un servidor DNS dinámicas, de acceso libre, este nos permite asociar nuestra dirección IP a un nombre de dominio para la página WEB.

Finalmente, para poder establecer que los datos de la estación meteorológica transmitan en protocolo MODBUS, se utilizó la herramienta grafica Node-Red, que está instalada dentro del sistema operativo Raspbian, la que se encarga de convertir la comunicación MQTT a MODBUS. Para comprobar el adecuado funcionamiento de esta conversion se utilizó un simulador de llamado ModbusSlave, así como también instaló un SCADA de prueba denominado SCADALaquis, mostrando de una manera gráfica los datos de la estación, permitiendo así mismo consultas de los históricos de los datos.

ABSTRACT

This project consists in the design and implementation of a meteorological station based on the use of embedded systems like the Raspberry Pi and Arduino Yun, connected wirelessly through the communication of MQTT and MODBUS, which checks in real time the different climatic variations, with the option of historical inquiries.

The station is made up of an Arduino Yun, which is in charge of monitoring the climatic variation through its analog, digital and I2C communication inputs, getting the different values of humidity sensors, temperature, wind speed (anemometer), wind direction, rain gauge and solar irradiation.

On the other hand, to fulfill the functions of a server a Raspberry Pi with its Raspbian operating system is incorporated, where by programming in Java, which contains libraries of the MQTT, stores the variables in a database MySql.

The database is used as a link between the sensors and the WEB server. This server was executed on the Raspberry Pi, using Apache Tomcat, which is used as a proxy server. This fact allows access to resources, routing all traffic (searching on the WEB) to the server, due to the need that the application and the Web page should be shown outside the private network.

The Web page generation was done through the WEB Ratio program, which is updated in real time on a regular basis and they are shown by tables and graphs on the WEB page: emeteo.ddns.net. Through a dynamic DNS server, with free access, this allows us to associate our IP address with a domain name for the WEB page.

Finally, in order to establish that the data from the meteorological station transmits in MODBUS, the Nod-Red graphic tool was used, which is installed within the Raspbian operating system. This is responsible for converting the MQTT to MODBUS. To verify the correct function of this, a ModbusSlave simulator was used, as well as a test SCADA called SCADALaquis, showing in a graphical way the data of the station, also allowing the requirement of the historical data.

1. INTRODUCCIÓN

1.1 Antecedentes

Empresa Electro Generadora del Austro (ELECAUSTRO S.A) dentro de sus necesidades de información para la toma de datos tanto hidrológicos como meteorológicos, en sus complejos hidroeléctricos en convenio con la Empresas ETAPA EP tiene instaladas estaciones meteorológicas que miden parámetros como son: lluvia, temperatura ambiental, humedad relativa , presión atmosférica , velocidad del viento, etc., los datos obtenidos se usan para planificar la programación de energía y despacho de agua basados en los registros históricos que se obtienen.

ELECAUSTRO S.A dentro de su parque de generación ha ido incrementando las centrales hidroeléctricas, pero en la zona de la recientemente construida Minicentral de Guacaleo aún no se ha implementado la estación meteorológica necesaria para la medición de datos, por lo que en el presente trabajo se plantea la construcción de una estación para instalarla en la dicha zona, como un plan piloto.

1.2 Justificación

A pesar que actualmente se puede comprar estaciones meteorológicas en el mercado, las mismas son importadas y tienen un costo elevado, aproximadamente 20.000 USD; por lo que se propone desarrollar una estación meteorológica apegada a los estándares de calidad y telecomunicaciones que tiene la Empresa, para este tipo de equipos, con un costo menor al que se puede adquirir en el mercado nacional.

Mediante la implementación de la estación meteorológica se busca tener acceso a la información de los sensores instalados desde cualquier parte del mundo a través de una página WEB, lo cual a más de servir a ELECAUSTRO S.A permitirá proporcionar acceso a la información meteorológica a entidades como Secretaria del Agua (SENAGUA), Gobierno Autónomo Descentralizado Municipal del Cantón de Guacaleo, etc., para que puedan utilizar en la planificación y alerta temprana de inundaciones y otros eventos naturales.

Para cumplir este objetivo se utilizará un dispositivo Arduino en el cual se conectarán el conjunto de sensores que realizarán las lecturas de las variables meteorológicas, para luego ser enviados estos datos a un dispositivo Raspberry Pi, que tendrá la función tanto de un servidor Web como de una base de datos, el que conectándose mediante protocolo de comunicación industrial tendrá la capacidad de enviar información al Programador Lógico Controlado (PLC) principal de la Minicentral y por ende al Sistema Informático de Supervisión, Control y

Adquisición de Datos (SCADA) de la Empresa ELECAUSTRO S.A. Para el diseño se considerará un modelo de red jerárquico de sensores.

1.3 Alcance

Diseñar, desarrollar e implementar una estación meteorológica apegada a los estándares de calidad que tiene la Empresa ELECAUSTRO S.A., que permita monitorear los datos en línea a través de la WEB y generar reportes automáticos a las áreas involucradas de la Empresa, considerando un modelo para redes jerárquicas de sensores, utilizando sistemas embebidos y protocolos MQTT y MODBUS

1.4 Objetivos

1.4.1 Objetivo General

Diseñar, desarrollar e implementar una estación meteorológica apegada a los estándares de calidad que tiene ELECAUSTRO, que permita monitorear los datos en línea a través de la WEB y generar reportes automáticos a las áreas involucradas de la Empresa, considerando un modelo para redes jerárquicas de sensores, utilizando sistemas embebidos y protocolos MQTT y MODBUS.

1.4.2 Objetivos Específicos

- Investigar, estudiar y analizar las mediciones y predicciones meteorológicas de los distintos sensores de medición de los diferentes parámetros: lluvia, temperatura ambiente, humedad, presión, velocidad y dirección del viento.
- Diseñar el sistema de medición de datos meteorológicos, considerando un modelo de red jerárquico de sensores.
- Diseñar una estación meteorológica que cumplan los estándares de calidad de la empresa ELECAUSTRO S.A.
- Desarrollo e implementación en los protocolos MQTT y MODBUS para la comunicación.
- Diseñar y desarrollar una base de datos en la que se recopilarán los datos obtenidos con los diversos instrumentos de medición meteorológica para luego ser presentadas por la Web.
- Diseñar y ejecutar un plan de experimentación que permita analizar y validar los datos obtenidos de los distintos sensores.

1.5 Planteamiento Del Problema

El conocer las variaciones climáticas es de mucha ayuda para los seres humanos, ya que podremos saber cuándo realizar ciertas tareas sin que estas sean afectadas por el estado del tiempo, así de esta manera podemos tomar las debidas precauciones o suspenderlas si fuera lo más apropiado.

Si bien existe el Instituto Nacional de Meteorología e Hidrología (INAMHI) que es la entidad encargada del monitoreo del clima, esta nos ofrece informes climatológicos a nivel general, pero algunas empresas, industrias, etc., necesitan de un informe especifico, creando la necesidad de construir sus propias estaciones meteorológicas, para la realización de tareas en las mismas.

Empresa Electro Generadora del Austro (ELECAUSTRO) dentro de sus necesidades de información para la toma de datos tanto hidrológicos como meteorológicos, en sus complejos hidroeléctricos tiene instaladas estaciones meteorológicas que miden parámetros como son: lluvia, temperatura ambiental, humedad relativa , presión atmosférica, velocidad del viento, etc., los datos obtenidos se usan para planificar la programación de energía y despacho de agua basados en los registros históricos que se obtienen.

La Empresa ELECAUSTRO S.A. dentro de su parque de generación ha incrementado las centrales hidroeléctricas, pero en la zona de la recientemente construida Minicentral de Gualaceo aún no se ha implementado la estación meteorológica necesaria para la medición y obtención de datos.

El presente proyecto Técnico está orientado a buscar una forma eficiente y a bajo costo de medir las variables meteorológicas como: presión atmosférica, cantidad de lluvia, humedad, temperatura ambiental, irradiación solar, además de almacenar los datos eficientemente y poder visualizarlos de forma gráfica tanto en la Web como en una aplicación Android.

1.6 Recopilación De Información Básica

1.6.1 Equipos de control y sistema SCADA

ELECAUSTRO dispone de tres centros de control localizados en las centrales de generación de Saymirín, El Descanso y Ocaña, que cuentan con arquitecturas de control idénticas.

El centro de control de El Descanso es desde donde se monitorea y controla la Minicentral Gualaceo, así también se encuentra el data center donde se almacena los datos provenientes de la Minicentral de Generación en tiempo real, adquiridos por un PLC SAYTEL Telvent-Schneider, que es el Programador Lógico Controlado (PLC) encargado de recibir las señales de campo, digitales y analógicas, las cuales pueden ser cableadas directamente o a través de protocolos de comunicación destinados para el efecto.

Dispone del software SCADA OASYS DNA de Telvent-Schneider, que se encuentra instalado en dos servidores que disponen de en Configuración redundante (Hot Standby).

La filosofía de control de todas las centrales y localidades de ELECAUSTRO es distribuida, modular, escalable y abierta, donde el protocolo de comunicación a nivel de control y supervisión es el IEC 60870-5-104 y el protocolo de comunicación de protecciones es el IEC 61850 [1].

Además, para equipos periféricos auxiliares utiliza protocolo de comunicación modbus para comunicar la información al PLC maestro de cada central y este se interconecta con el SCADA a través de protocolo de comunicación IEC 60870-5-104 [1].

1.6.2 Red de comunicación.

La Empresa ELECAUSTRO S.A. para el intercambio de información entre las centrales de generación y los centros de control posee dos redes de comunicación.

La primera red es de control, la cual envía los datos del sistema de control de las centrales de generación hacia el centro de control por medio de fibra óptica en protocolo IEC -104.

La red de control establece comunicación con los PLC de control de las unidades de generación, así como de las RTU de control de los tanques de presión y bocatomas. Los PLC indicados adquieren datos ya sea directamente cableado a través de sus entradas analógicas y digitales o por medio de comunicación a través de protocolo MODBUS, DNP3 o IEC 61850.

La segunda es la red corporativa, la cual sirve para la comunicación telefónica y reparto de servicio de internet. Esta red brinda servicio de internet por dos proveedores, el principal es TELCONET y el segundo respaldo es ETAPA. Esta red permitiría realizar la comunicación IOT de los diferentes sensores de la estación meteorológica.

1.6.3 Almacenamiento de datos.

Para el almacenamiento de datos de las centrales de la Empresa ELECAUSTRO S.A se utiliza el sistema SCADA OASYS DNA de Telvent-Schneider, en la misma se encuentra actualmente guardados los datos de las estaciones meteorológicas que normalmente se procesan por protocolo MODBUS.

Los datos se guardan en dos servidores que disponen de en Configuración redundante (Hot - Standby) en cada uno de los centros de control, según corresponda, realizando copias periódicas de respaldo de toda la información disponible en la base de datos.

Actualmente los datos que se guardan en el sistema SCADA son los de lluvia de las estaciones meteorológicas ubicadas en el complejo Hidroeléctrico del Rio Machangara, las mismas que son procesados para la programación de energía de las centrales Hidroeléctricas

Saucay y Saymirín, así como la programación de almacenamiento de agua de las represas Labrado y Chanlud.

Para el caso de entrega de información al público, ELECUSTRO tiene una página WEB www.elecaustro.gob.ec diseñada en JOOMLA, Esta página WEB permitiría colocar un enlace para que el usuario que necesite ingresar en la estación meteorológica.

2. ESTADO DEL ARTE ESTACIONES METEOROLÓGICAS

2.1 Estaciones Meteorológicas

Estación meteorológica se define como una instalación destinada a medir y registrar regularmente diversas variables del clima donde se ubica, estos datos se utilizan tanto para la elaboración de predicciones meteorológicas a partir de modelos numéricos como para estudios climáticos.

2.1.1 Clasificación de estaciones Meteorológicas

Según [2] lo establecido por la Organización Meteorológica Mundial (OMM), las estaciones meteorológicas se las puede clasificar de las siguientes formas:

SEGÚN SU FINALIDAD	CLASIFICACION
Sinóptica	Climatológica Agrícolas Especiales Aeronáuticas Satélites
De acuerdo a la magnitud de las observaciones	Principales Ordinarias Auxiliares o adicionales
Por el nivel de observación:	Superficie Altitud
Según el lugar de observación	Terrestre Aéreas Marítimas

Tabla 1. Clasificación de Estaciones Meteorológicas según OMM
Fuente: www.senamhi.gob.pe/main_down.php?ub=mmt&id=cap2

Pero de hoy en día las estaciones meteorológicas podemos también clasificarlas en dos principales grupos las realizadas para nuestros domicilios llamadas Domesticas y las estaciones propiamente dichas profesionales.

Estaciones meteorológicas domesticas nos permite conocer datos como la temperatura y humedad y con base a estos criterios se podrán realizar algunas tareas como, por ejemplo: el riego, estas son de una estructura muy sencilla y están a costos muy bajos.

Estaciones profesionales tienen ya integrados varios sensores como un anemómetro, dirección del viento, presión, estas necesitan de personas que tengan un conocimiento medio a cerca de programación de estos dispositivos y sus costos pueden llegar a ser muy elevados.

En nuestro país son muy escasas las estaciones meteorológicas conocidas como se describir en [3] el Ministerio del Ambiente (MAE), a través del Proyecto de Adaptación al Impacto del Retroceso Acelerado de Glaciares en los Andes Tropicales (PRAA), ha desarrollado dos Estaciones Meteorológicas Automáticas para el Monitoreo de Glaciares y Páramos. Este proyecto cumple la función de unir distintas estaciones meteorológicas, teniendo en total 12 sensores climáticos (viento, nubosidad, humedad, entre otros), la comunicación de las mismas se las realiza mediante los protocolos TCP/IP y GPRS también se utilizó un routers NanoStation 2 y un sistema de repetición punto a punto, los datos obtenidos son almacenados y enviados a las distintas estaciones para su posterior análisis.

En [4], describe una estación experimental con la cual se realiza el estudio de las condiciones climatológicas tomando diferentes datos como: presión, humedad, precipitaciones y temperatura. En el cual se implementó una estación para medidas pluviométricas y la comunicación se la realiza mediante redes de telefonía móvil (GSM) y radio enlaces hacia otras estaciones.

2.1.2 Sensores en las estaciones Meteorológicas.

De la literatura especializada y de las estaciones meteorológicas en el mercado, se puede encontrar las mismas con los siguientes sensores recomendados:

- Temperatura
- Humedad
- Velocidad del viento
- Dirección del viento
- Presión Barométrica
- Precipitación
- Opcional: Medición de radiación solar

2.1.3 Reporte de datos de las estaciones Meteorológicas

Generalmente las estaciones meteorológicas han ido evolucionando su transmisión de datos de lo que inicialmente capturaban la información en papel milimetrado y a través de una persona las lecturas visuales, hoy en día podemos encontrar la transmisión por diferentes Formas:

WEB:

Se encuentran estaciones meteorológicas que vienen con programas ejecutables para Windows o MAC y conectarse a la página WEB por medio de la cual envía datos en línea siempre y cuando tenga una conexión a internet. Las páginas WEB que se pueden encontrar para este efecto son:

www.wunderground.com

www.rainwise.com

Para ingresar los datos a la WEB dependiendo del servicio que se requiere, puede ser gratis y pagado para un costo anual para entrega de datos con mayor detalle**.**

MODBUS

Se encuentran estaciones meteorológicas profesionales integrables al sistema SCADA vía RS232 para luego transmitir la información en vivo a un PLC o sistema SCADA, su costo es alto.

GPRS o RADIO

Se encuentran estaciones meteorológicas que incorporan módems de recepción vía GPSR o radio a distancias lejanas y con línea de vista para el caso de radio, y con GPRS es necesario la red de celular para el envío de información. Así mismo tiene un costo alto.

DESCARGA EN SITIO

Finalmente, existen tipos de estaciones meteorológicas que, si no disponen de comunicaciones, tiene la opción de descargar en sitio los datos grabados en su datalogger, las cuales su autonomía de grabación de datos está por arriba de los 3 meses, la más básica. Este tipo de estaciones no tiene un costo alto.

2.2 MQTT (Message Queue Telemetry Transport)

MQTT (Message Queue Telemetry Transport) en [5] nos indica que nació en 1999, por Dr. Andy Stanford-Clark, trabajador de IBM, y Arlen Nipper, de Arcom. Este ha sido usado en diversos proyectos alrededor del mundo, entre los últimos usos que se le ha dado a este es de datos enviados por sensores, sistemas de bases de datos distribuidas y aplicaciones móviles un claro ejemplo son: Facebook, Messenger, Amazon.

OASIS (Organization for the Advancement of Structured Information Standards), dio a conocer la primera especificación del estándar MQTT (Message Queue Telemetry Transport) 3.1.1 [6].

En [7] nos indica que es un recurso de mensajería simple y ligera, teniendo en cuenta que nos garantiza una interoperabilidad parcial entre publicadores y subscriptores. MQTT tiene la capacidad de intercambiar mensajes entre diferentes aplicaciones de MQTT, con la condición que los dos extremos de la comunicación deben conocer el formato del cuerpo. Este protocolo necesita un encargado de enviar los mensajes a los distintos destinos, este encargado se le suele llamar bróker.

Dentro del funcionamiento de MQTT, podemos diferenciar varios conceptos como los que nombraremos a continuación:

Los clientes pueden:

- Publicar mensajes de aplicación que puedan interesar a otros clientes.
- Suscribirse a mensajes de aplicación que les puedan interesar.
- Revocar suscripciones para dejar de recibir ciertos mensajes de aplicación.
- Desconectarse del bróker.

El bróker se encarga de:

- Aceptar conexiones de clientes.
- Aceptar mensajes de aplicación enviados por clientes.
- Procesar mensajes de suscripción y revocación de parte de clientes.
- Notificar los mensajes de aplicación que cumplan los criterios de suscripción de los clientes.

La arquitectura MQTT en [8] indica que está basada en un modelo cliente / servidor donde cada sensor es un cliente que debe conectar a un servidor (bróker) a través de TCP y además está orientada a mensajes, por lo que cada uno de ellos es un paquete de datos opaco para el bróker. Cada mensaje se publica a una dirección (topic) y los clientes pueden suscribirse a múltiples topics (recibiendo todos aquellos mensajes que se publiquen en los topics suscritos suscritos).

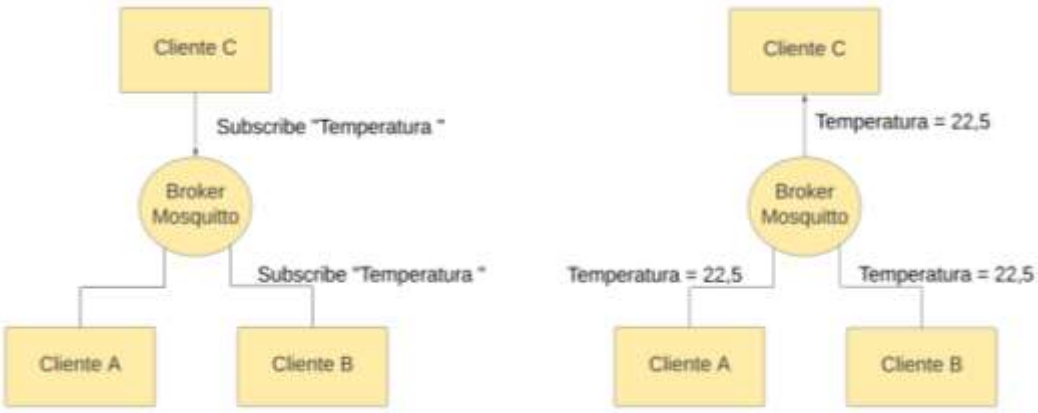


Figura 1 . MQTT Ejemplo de arquitectura
Fuente: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

De esta forma MQTT permite la comunicación:

- uno a uno
- uno a muchos; y,

- muchos a uno, de un modo fácil y sencillo.

2.3 MODBUS

En [9] inicialmente se definía a Modbus como una especificación de tramas, mensajes y funciones para la comunicación con los PLCs Modicon. Modbus, pero de hoy en día puede implementarse sobre una comunicación serie, la comunicación de este protocolo se realiza por medio de tramas binarias o ASCII con un proceso interrogación-respuesta.

Desde su inicio este estaba integrado en los PLCs de Modicon, en 1979, esta ha dado como resultado un estándar para la unión serie entre dispositivos de manejo industrial. Como medio físico utiliza el par trenzado o fibra óptica. En la actualidad Modbus es soportado por el grupo de automatización Schneider (Telemechanique, Modicon)

En [10] nos da a conocer las competencias de una aplicación Modbus el modelo de interconexiones de sistemas abiertos (OSI), encontraremos que el protocolo está ubicado en dos niveles de los siete de OSI, en el Nivel de aplicación (aplication layer), donde trabajarían los programas que utilizan los dispositivos e interpretan los datos de dichos instrumentos. (Ej: Windmill, Hyperterminal, etc.).

Es muy importante especificar que el intercambio de tramas básicas del protocolo Modbus se hace en la capa de enlace como se muestra en la Figura 2.

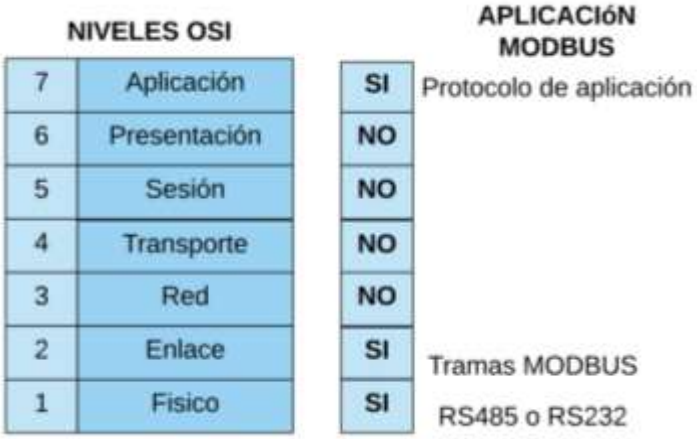


Figura 2 . Niveles OSI en los que se aplica MODBUS.
Fuente: https://www.eeymuc.co/31-protocolo-modbus/

Este se maneja mediante tramas, las cuales permiten consultar o actualizar el valor de dichos registros. La trama es recibida por todos los dispositivos de la red, pero es únicamente procesada si la trama está dirigida al identificador del sensor que la recibe, enviando una respuesta modificando tan solo el PDU.

Figura 3. Comunicación de MODBUS
Fuente: Autor

2.4 ARDUINO YUN

Arduino en [11] es una plataforma de electrónica que es muy flexible para la creación de proyectos basados en software y hardware libre, sencillo de utilizar.

Figura 4. Arduino Yun
Fuente: http://alsw.net/tienda/arduino/arduino_original/arduino-yun/

La principal característica Arduino Yún es su capacidad de conectarse a Internet, ya sea a través del puerto Ethernet que viene integrado a la placa o bien por medio de una red Wi - Fi. A simple vista el Arduino Yún es bastante parecido a los modelos Arduino Leonardo y Uno.

Conectividad:

Esta placa contiene una red ethernet 10/100 Mbps y otra Wifi (IEEE 802.11 b/g/n, 2,4GHz) que puede utilizarse tanto como cliente o como punto de acceso.

Esta nos brinda muchas utilidades, vamos a destacar que tiene un zócalo para memoria MicroSD esta nos permite almacenar datos como páginas WEB, valores adquiridos por los diferentes sensores o algún otro parámetro que este proyecto necesite, dándole muchas más funcionalidades a la placa.

El software consiste en la programación de un microcontrolador en la placa mediante un lenguaje propio de Arduino el mismo que está basado el Wiring (plataforma de programación para prototipos con electrónica), y el entorno está basado en Processing (entorno de desarrollo y programación basado en Java) [12].

El Arduino adquiere la información de los distintos componentes, sensores, etc., mediante pines de entrada. Además, tiene la capacidad por medio de los pines de salida dar señales de mando pudiendo controlar luces, motores y otros actuadores.

Hay que recalcar que en [13] para que Arduino funcione no necesita estar conectado a un computador. Como anteriormente se mencionó Arduino se programa mediante su lenguaje propio, pero es posible utilizar otros lenguajes de programación tales como: Ruby, Python, Java, C#, C++, C, etc., debido a que Arduino usa una transmisión serial soportada por la mayoría del lenguaje de programación.

2.5 RASPBERRY PI

Raspberry Pi en [14] nos indica que es una placa computadora económica desarrollada en Reino Unido por la fundación Raspberry Pi, la misma es una organización sin ánimo de lucro, y su principal objetivo es el incrementar el estudio de las ciencias de computación en los niveles secundarios y universitarios.

Su principal objetivo es el posibilitar realizar pequeños proyectos y el poder aprendizaje de un lenguaje de programación de forma sencilla, teniendo en cuenta que nuestros computadores que tenemos en nuestros hogares solo nos sirve para tareas informáticas o de ocio.

En [15] no dice que esta tecnología fue diseñada con el fin de ser lo más barato posible y poder llegar al mayor número de personas. Esta idea empezó en el año 2006, pero como organización empezó a trabajar desde 2008, y la comercialización comenzó a principio del año 2012.

Si hablamos de hardware, en [16] encontramos dos modelos de Raspberry; el modelo A que es el inicial y el modelo B, este el con que se comercializa de hoy en día. El modelo B tiene una placa de 8,5 cm por 5,3 cm. Tiene un chip integrado Broadcom BCM2835, un procesador ARM11 con varias frecuencias que se las puede incrementar hasta 1 GHz, contiene un procesador gráfico Video Core IV, una memoria RAM de 512MB.

La placa Raspberry cuenta una salida de audio y video a través de un conector HDMI lo que nos permite conectar la tarjeta un monitor, como a una televisión desde luego que cuente con este tipo de conexión.

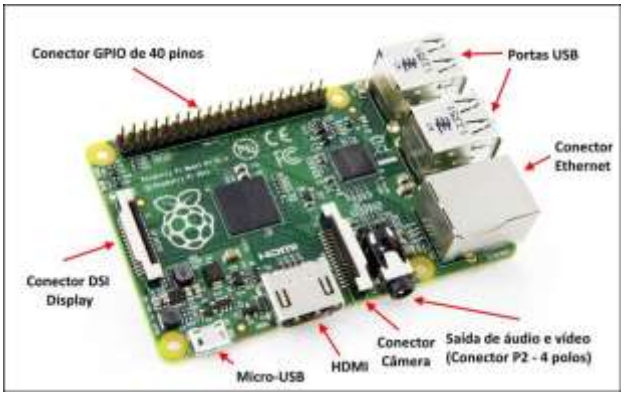


Figura 5. Componentes Raspberry Pi modelo B.
Fuente: http://mikroe.es/raspberry-pi-3-modelo-b/

Raspberry no tiene un software específico instalado por ello, este deja que se escoja el mismo, pudiendo escoger entre Raspbian, distintas distribuciones de Linux, Windows IoT, etc. Este software se instala en una memoria micro SD, este proceso lo que se puede realizar de dos formas denominadas:

1. Berry Boot. - Programa que se encarga del trabajo de instalación desde la propia Raspberry Pi. Una vez copiados los archivos en la tarjeta, se coloca en el Raspberry y este permite elegir el sistema operativo a instalar, descargándolo desde internet [16].
2. Noobs. - Esta es una aplicación que nos facilita la instalación de Linux. Esta aplicación no necesita el acceso a internet, el proceso sería el de descargar Noobs y descomprimirlo en la tarjeta SD de al menos 4 GB de capacidad. Se instalará Raspbian como sistema operativo [16].

Como último a Raspberry le podemos añadir algunos dispositivos como el módulo de cámara. Este sensor cuenta con 5 megapíxeles y podría grabar video a 1080p h.264 a 30 fotogramas por segundo. La cámara se conecta gracias al puerto CSI a través de un cable plano flexible.

3. DISEÑO DE LA ESTACIÓN METEOROLÓGICA

3.1 Consideraciones.

Previo al inicio del desarrollo del diseño de la estación meteorológica, se identificaron aspectos necesarios a ser considerados para el desarrollo de la misma, entre ellos tenemos:

3.1.1 Consideraciones físicas

- Dentro de las consideraciones físicas se encuentra determinado el desarrollo del equipamiento de la estación meteorológica, tiene que ser apta para intemperie y tener una protección el tableo IP 67 y con una resistencia adecuada.
- Otro aspecto a considerar es la utilización de bases inoxidables para la implantación de sensores que pueda soportar la rigurosidad del clima a la intemperie.
- Se considera que no todos los sitios donde potencialmente podríamos instalar las estaciones meteorológicas no se dispondrá energía eléctrica.
- Se deberá poder tener la capacidad de conectar y desconectar los sensores de una manera fácil rápida y ágil, que brinde las comodidades para poder reemplazar sensores o poder incrementar mediciones.
- Debe considerarse la portabilidad del equipo y su fácil instalación.

3.1.2 Consideraciones eléctricas.

- Se consideró que la energía que consume es pequeña al estar diseñada con un Arduino y sensores de poco consumo, por tanto, se prevé un respaldo suficiente a través de un sistema solar de energía.
- Se considera que las variables a medir del clima son estándares, según la recomendación de la literatura especializada.
- Los protocolos con los que pueden integrarse al sistema SCADA es MODBUS, IEC 61850, DNP3 e IEC 60870-5-104; de la cual se determina Modbus para la implementación.
- Se debe tener un sistema de corriente continua todo el sistema de alimentación, a través en la opción 1, de un panel solar y en la opción 2, de un adaptador a 12 V
- Se considera alimentar los sensores a 5V y 3.3 V desde la misma fuente que tiene el Arduino, por su limitado consumo de energía.

3.1.3 Consideraciones de programación.

- Se utilizará software libre para la implementación de la estación meteorológica
- La programación Arduino deberá ser compatible con la base de datos.

- El reporte de la estación meteorológica deberá tener ser amigable en la interpretación de resultados.

3.2 Definición de Sensores a utilizar

La definición de sensores a incluir en la estación meteorológica, se ha tomado en función de la recomendación de la literatura especializada a monitorear en estos temas.

3.2.1 Sensor de temperatura y humedad

Para la medición de los fenómenos físicos de temperatura y humedad se va a definido utilizar el sensor DHT22 [17]; este sensor tiene la capacidad de medir la temperatura y humedad en el mismo dispositivo y la transmisión de datos se lo realiza en una sola señal y como valor agregado al proyecto se tiene que ubicar una librería directa para comunicarse con Arduino.

Su principio de funcionamiento en [18] nos indica que se basa en un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos.

El DHT22 está conformado por 4 pines; el primer pin de la izquierda a la fuente de alimentación 3 -5V, el segundo pin a tu pin de entrada de datos, el tercer pin no se utiliza y el cuarto (último) pin a tierra.

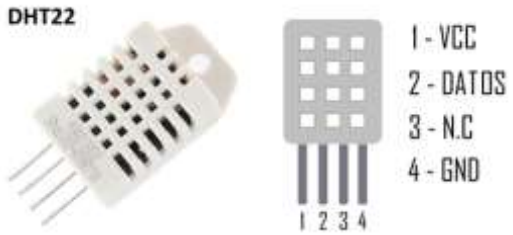


Figura 6. Pines sensor DHT22
Fuente: http://www.techmake.com/00358.html

3.2.1.1 Protocolo de comunicación:

El DHT22 (o AM3202) es un senosr que esta conformado por un microcontrolador que se encarga de realizar la comunicación con los demas dispositivos, este utiliza un bus simple de comunicación, es un bus de una sola línea para la transmisión y recepción de datos.

Características:

- Puede estar localizado a 25 metros del controlador sin tener pérdida de datos por la distancia del cable.

- Se encuentran calibrados de fábrica, tienen su calibración almacenada en memoria, este coeficiente de calibración se utiliza en las mediciones realizadas por el dispositivo.
- Consumo de corriente reducido.
- Fácil de reemplazar.

3.2.1.2 Especificaciones técnicas del DHT22

- Alimentación: 3.3Vdc ≤ Vcc ≤ 6Vdc.
- Rango de medición de temperatura: -40°C a 80 °C.
- Precisión de medición de temperatura: <±0.5 °C.
- Resolución Temperatura: 0.1°C.
- Rango de medición de humedad: De 0 a 100% RH.
- Precisión de medición de humedad: 2% RH.
- Resolución Humedad: 0.1%RH.
- Tiempo de censado: 2s.

Figura 7. Sensor DHT22 y su protector
Fuente: https://es.aliexpress.com/item/Misol-spare-part-for-weather-station-Transmitter-thermo-hygro-sensor-433Mhz/32791676037.html

3.2.2 Sensor de viento (Anemómetro)

El anemómetro es considerado un equipo meteorológico, que se utiliza para la predicción del clima, específicamente la velocidad del viento que incide sobre él; si este instrumento es colocado fijamente en la tierra, medirá la velocidad del viento en ese instante, pero en el caso contrario puede servir para la medición de la velocidad de movimiento relativo del objeto con respecto al viento en calma.

En [18] nos indica que el anemómetro más común para la medición es el de cazoletas que está conformado por un eje vertical y tres copas que captura el viento, donde el número de revoluciones por segundo son registradas de forma eléctrica, para luego poder ser procesadas y hacer el cálculo de la velocidad del viento en el emplazamiento.

Para la medición de la velocidad del viento se va a utilizar un sensor analógico. El funcionamiento de este sensor se describe como un mini generador de corriente continua que al girar genera tensión en un rango de 0 – 5 Voltios VCC, teniendo como rango de medición de 0 – 30 m/s, respetivamente.

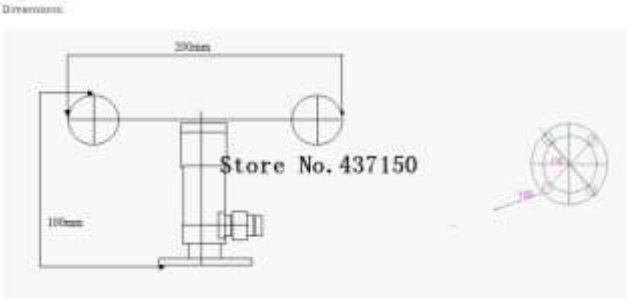


Figura 8. Diagrama físico del anemómetro
Fuente: https://es.aliexpress.com/w/wholesale-analog-anemometer.html

Características:

Modelo	JL-FS2	
Nombre	UniversalWind speed sensor	
Cable y salida	Waterproof Aviation Plug-in	
Sensor	Señal de salida	20mA / voltage- 0-5V
	Estilo de sensor	Tres cucharas
	Velocidad inicial del viento	0.4-0.8m / s
	Resolución	0.1m / s
	Rango efectivo de medición de la velocidad del viento	0-30m /s
Parámetros Generales	Porcentaje de error	3%
	Distancia de transmisión	Mayor de 1000m
	Medio de transmisión	Transmisión por cable
	Alambrado	Tres cables
	Temperatura de trabajo	-40 a 80
	Tensión de alimentación	DC12-24V puede ser común
	Fuente de Poder	VoltageTypeMAX0.3W; Tipo actual MAX0.7W
	Peso	<1 kg Cable estándar de 3 m

Tabla 2. Características técnicas Anemómetro
Fuente www. https://es.aliexpress.com/store/product/factory-direct-wind-sensor-4-20mA-signal-output-wind-speed-transmitter-anemometer/437150_32279079084.html

Figura 9. Anemómetro
Fuente www. https://es.aliexpress.com/store/product/factory-direct-wind-sensor-4-20mA-signal-output-wind-speed-transmitter-anemometer/437150_32279079084.html

3.2.3 Sensor dirección de viento(Veleta)

La veleta se la puede describir como un dispositivo giratorio que consta de una placa, la cual gira de forma libre y un señalador que indica la dirección del viento, como principio de funcionamiento presenta el sensor tipo analógico, una resistencia variable interna, que en cada posición entrega un determinado valor de tensión, el cual se replica en la rosa de los vientos entregando la dirección correspondiente, y que es programada en el Arduino.

Características:

Cables:

- Rojo: potencia + 12-24v
- Negro: poder
- Amarillo: señal de salida 0-5v
- Azul: señal de salida 4-20ma

Parámetros técnicos:

- Tensión de alimentación: DC9-24V DC12-24V
- Salida de señal: Voltaje: 0-5V
- Estilo del sensor: Estilo de la cola
- Resolución: 0.1m / s
- Rango de medición: 0-360 Orientación completa 16
- Error del sistema: 3%
- Distancia de transmisión: Más de 1000m
- Cableado: Voltaje: Tres cables de corriente: Tres hilos, dos hilos digitales:
- Temperatura de funcionamiento: -20 º C ~ 80 º C
- Consumo de energía: 0,3 W

Figura 10. Veleta
Fuente: https://es.aliexpress.com/wholesale?catId=0&initiative_id=AS_ 20180131181350&SearchText=wind+speed+sensor+direction

3.2.4 Sensor de lluvia (pluviómetro

El sensor de lluvia o pluviómetro, en [19] nos dice que este es un Instrumento que sirve para medir la cantidad de agua precipitada de un tiempo y lugar determinado. La unidad de media es en milímetros (mm) de lluvia en 1 metro cuadrado.

Una precipitación de 5mm indica que, si toda el agua de la lluvia se acumulará en un terreno plano sin escurrirse ni evaporarse, la altura de la capa de agua seria de 5mm. Los milímetros (mm) son equivalentes a los litros por metros cuadrados.

El registro de la precipitación se lo puede realizar mediante volumen, altura de la lámina de precipitación así también a base del peso de la misma. Con los registros de las precipitaciones es posible conocer la distribución temporal de las lluvias.

3.2.4.1 Pluviómetro balancín

Este es un instrumento que mide las precipitaciones mediante un mecanismo de cubos calibrados de iguales proporciones que da a conocer el volumen de precipitación obtenido mediante una acción de intercambio. El intercambio se lo realiza el momento en que uno de los cubos se llena de lluvia y el otro está listo para continuar el proceso de medición.

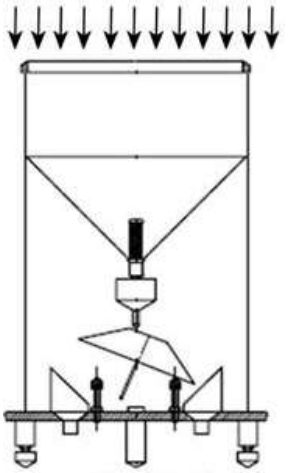


Figura 11. Funcionamiento de pluviómetro
Fuente: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1680-03382013000200007

El mecanismo de balancín de las cubetas es generado cuando el peso de la lluvia caída es almacenado en uno de los dos cubos provocando un desbalance respecto al otro cubo. Esto causa que cambié rápidamente de cubo y se vacié el agua contenida en el primer cubo.

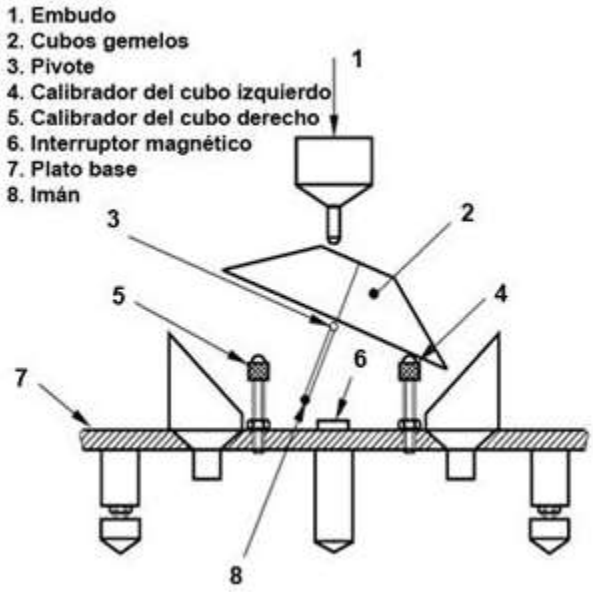


Figura 12. Partes pluviómetro balancín
Fuente: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1680-03382013000200007

Figura 13. Pluviómetro MS-WH-SP-RG
Fuente: https://es.aliexpress.com/item/Free-Shipping-Spare-part-for-weather-station-to-measure-the-rain-volume-for-rain-meter-for/32793155455.html

3.2.5 Sensor de presión BMP180

El sensor BMP180 está diseñado para leer la presión atmosférica, definiendo esta es la fuerza que el aire ejerce sobre la superficie terrestre. La presión atmosférica se debe a la columna de aire que está en una determinada área que se extiende desde este punto hasta el límite superior de la atmosfera, debido a esto si medidos la presión atmosférica en los puntos más altos la presión será baja ya que hay menos cantidad de aire sobre nosotros.

La presión atmosférica también varía debido a las condiciones climáticas en especial con la temperatura, pues esta cambia la densidad del aire, por tanto, cambia el peso [20].

El sensor BMP180 está diseñado para conectarse directamente a un microcontrolador de un dispositivo móvil a través del bus I2C.

Las conexiones de este sensor se determinan de la siguiente manera:

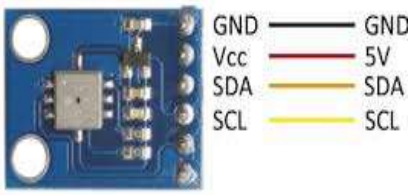


Figura 14. Pines Sensor BMP180
Fuente: https://www.luisllamas.es/medir-presion-del-aire-y-altitud-con-arduino-y-barometro-bmp180/

Características

- Interfaz Digital de dos cables (I2C)
- Amplio rango de medición de presión barométrica (desde - hasta)
- Ultra-bajo consumo de energía
- Bajo ruido
- Completamente calibrado

- Medición de temperatura incluida
- Ultra plano y pequeño tamaño
- Alimentación: 1.8V – 3.6V
- Rango de medición: 300 – 1100hPa
- Velocidad del protocolo máxima: 3.4 MHz.

Figura 15. Sensor BMP180
Fuente: https://es.aliexpress.com/item/Free-Shipping-Barometer-BMP180-Module-Replace-BMP085-Digital-Barometric-Pressure-Sensor-Board-Module-for-Arduino-Raspberry/32734971588.html

3.2.6 Sensor de UV ML8511

El Sensor ML8511, es un sensor de luz ultravioleta, trabaja por medio de una señal analógica en relación con la cantidad de luz UV que este detecta. Este es muy útil para la creación de dispositivos para la prevención de explosión a los rayos ultravioleta las cuales causan quemaduras solares, como también la detección de índice UV como se refiere a las condiciones climáticas.

Este sensor detecta 280-390nm luz con mayor eficacia. Este se clasifica como parte de la UVB (ardientes rayos) espectro y la mayoría de la uva (rayos de bronceado) espectro. Se da salida a una tensión analógica que es linealmente relacionado a la medida intensidad UV (mW/cm2). Si el microcontrolador puede hacer una conversión de la señal analógica a digital a continuación, puede detectar el nivel de UV.

Características:

- Sensor ML8511
- Espectro de detección: 289~390nm (1 voltio a 2.5 voltios)
- Salida linealmente relacionada con la intensidad de luz UV (mW/cm2)
- Voltaje de alimentación: 3.3~5V.

3.3 Definición de Software Libre a utilizar

Con el fin de que no se tenga problemas legales por la programación de SOTWARE propietarios y con licencia, se utilizará software libre para la programación de base de datos y sistemas operativo en los equipos a utilizar en la estación meteorológica.

3.3.1 MQTT (Message Queue Telemetry Transport)

MQTT (Message Queue Telemetry Transport) es un protocolo basado en binario donde los elementos de control son bytes binarios y no cadenas de texto, usa un comando y formato de confirmación. Las comunicaciones son más ligeras y rápidas para el diseño de internet de las cosas. Su última versión del protocolo es la MQTT V3.1, esta especificación establece que su propósito es "ser un protocolo de mensajes de publicación/subscripción basado en un bróker ligero, diseñado para ser abierto, simple, ligero y fácil de implementar".

MQTT es perfecto para su uso en los dispositivos integrados porque:

- Es asíncrono con diferentes niveles múltiples de calidad del servicio, lo que resulta ser importante en los casos donde las conexiones de Internet son poco confiables.
- Envía mensajes cortos y estrechos que se vuelven adecuados para las situaciones de un bajo ancho de banda.
- No se requiere de demasiado software para implementar a un cliente, lo cual lo vuelve fantástico para los dispositivos como Arduino con una memoria limitada. [21].
- El formato de paquete o mensaje MQTT consta de un encabezado fijo de 2 bytes (siempre presente) + encabezado variable (no siempre presente) + carga útil (no siempre presente).

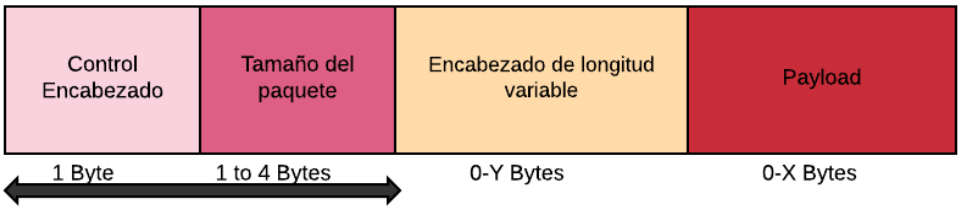


Figura 16. Formato del paquete MQTT
Fuente: http://www.steves-internet-guide.com/MQTT-protocol-messages-overview/

MQTT requieren el uso de este campo para transportar información de control adicional. Este puede ser similar, pero no el mismo para todos los tipos de mensaje, cabe recalcar que este campo no siempre está presente en un mensaje MQTT

Ejemplo

Paquete MQTT = control + longitud + nombre del protocolo + nivel de protocolo + Conectar banderas + keep alive+ payload

Lo interesante del MQTT es que el campo ID de cliente se envíe como la primera parte de la carga y no como parte del encabezado. Tiene un campo de 2 bytes de longitud como prefijo [22].

Payload = ID del cliente = longitud del ID del cliente + ID del cliente

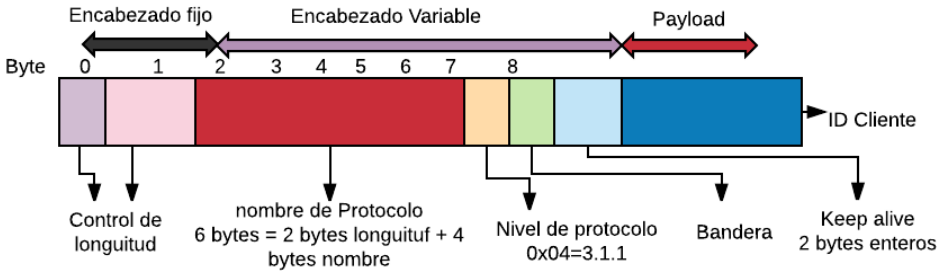


Figura 17. Estructura de conexión de Mensajes MQTT
Fuente: http://www.steves-internet-guide.com/MQTT-protocol-messages-overview/

La comunicación con el protocolo MQTT se basa en el cliente (Sensor) pública utilizando un nombre de locación o mejor conocido como Topics, los nodos que deseen recibirlo (Dispositivos) deben suscribirse a los Topics que estos publiquen. El servidor o Broker MQTT sirve de puente de comunicación entre la plataforma y el sensor para brindar seguridad en el acceso a la información mediante encriptación y autenticación, almacena los históricos para futuros accesos y generación de reportes.

Topología Publish/Subscribe

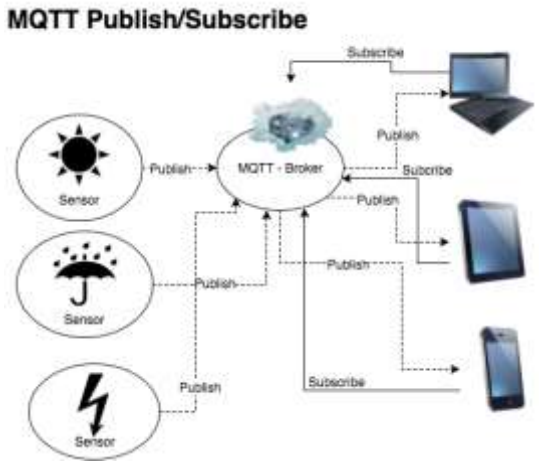


Figura 18. Diagrama Publicar/Subscribir
Fuente: http://noterau.blogspot.com/2016/03/mqtt-sensores-ambientales.html

3.3.1.1 Bróker

Es un software que es implementado en el protocolo MQTT se encarga de establecer la comunicación a nivel de aplicación, entre los distintos clientes. Es el encargado de recibir los mensajes, filtrarlos y enrutarlos a los clientes subscritos según su tópic.

Características

- Aceptar conexiones de clientes.
- Aceptar mensajes de aplicación enviados por clientes.
- Procesar mensajes de suscripción y cancelación de parte de clientes.
- Notificar los mensajes de aplicación que cumplan los criterios de suscripción de los clientes.

El Broker es el elemento encargado de gestionar la red y transmitir los mensajes, sin este no tendría una comunicación eficiente, es de fácil implementación e instalación y su funcionamiento es instantáneo.

Mosquitto

Broker Mosquitto este es un proyecto de código abierto desarrollado por Eclipse1. La configuración por defecto de mosquitto opera en el puerto 1883, su descarga se realiza desde la página Web oficial https://mosquitto.org/download/, es importante saber que Mosquitto de forma nativa solo funciona a través de sockets, esto implica que sólo se puede conectar al Broker si está conectado a la misma red.

3.3.1.2 Calidad de Servicio (QoS)

El MQTT tiene integrada un QoS (Quality of Service, calidad de servicio). El publisher tiene la posibilidad de integrar la calidad de servicio de su mensaje, para lo cual hay tres niveles:

- Un mensaje de QoS nivel 0 se entregará como mucho una vez. Eso significa que el mensaje se envía sin garantías de recepción.
- Un mensaje de QoS nivel 1 se entregará al menos una vez. El cliente lo transmitirá varias veces si es necesario, hasta que el bróker le confirme que lo ha enviado a la red.
- Un mensaje de QoS nivel 2 será obligatoriamente guardado por el emisor, que lo transmitirá siempre que el receptor no confirme su envío a la red. La principal diferencia radica en que el emisor utiliza una fase de reconocimiento más sofisticada con el bróker para evitar la duplicación de los mensajes (más lento, pero más seguro) [23].

3.3.1.3 Tópicos MQTT

Es una etiqueta con la cual se diferencian los mensajes de aplicación, para que luego estos sean publicados en la red y sean recibidos solo aquello clientes que están suscritos a esta etiqueta. La comunicación se basa en unos "topics" (temas), que el cliente que publica el mensaje crea y los nodos que deseen recibirlo deben subscribirse a él. La comunicación puede ser de uno a uno, o de uno a muchos. Un "topic" se representa mediante una cadena y tiene una estructura jerárquica [24].

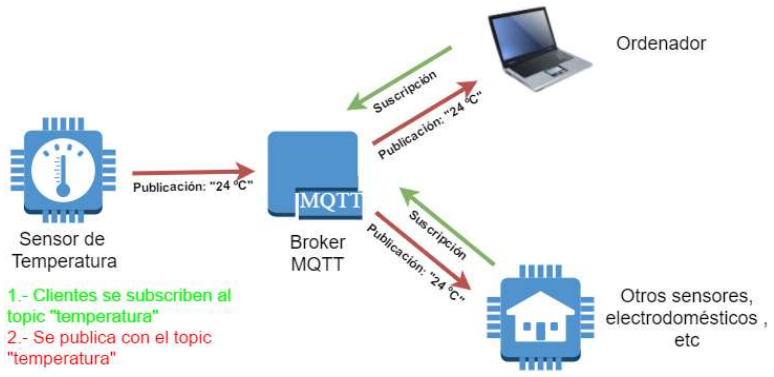


Figura 19. Arquitectura de envío de mensajes MQTT
Fuente: http://www.ermesh.com/aprender-protocolo-mqtt-parte-1/

3.4 Implementaciones en el proyecto

3.4.1 Arquitectura de Software

En esta sección se da a conocer los componentes de software que fueron utilizados para construir la aplicación que nos permitirá lo siguiente:

- Recoger datos que provienen de los diferentes sensores mediante el hardware libre tanto Arduino como Raspberry Pi.
- Implementar un servidor WEB en Raspberry Pi, de forma que se pueda acceder a ella a través de Internet para la respectiva visualización de los datos, en formato WEB.
- Implementar un módulo de reportes para dar a conocer los datos meteorológicos dependiendo de la fecha.
- Implementar una aplicación móvil, para la verificación de datos dependiendo de una hora y fecha.

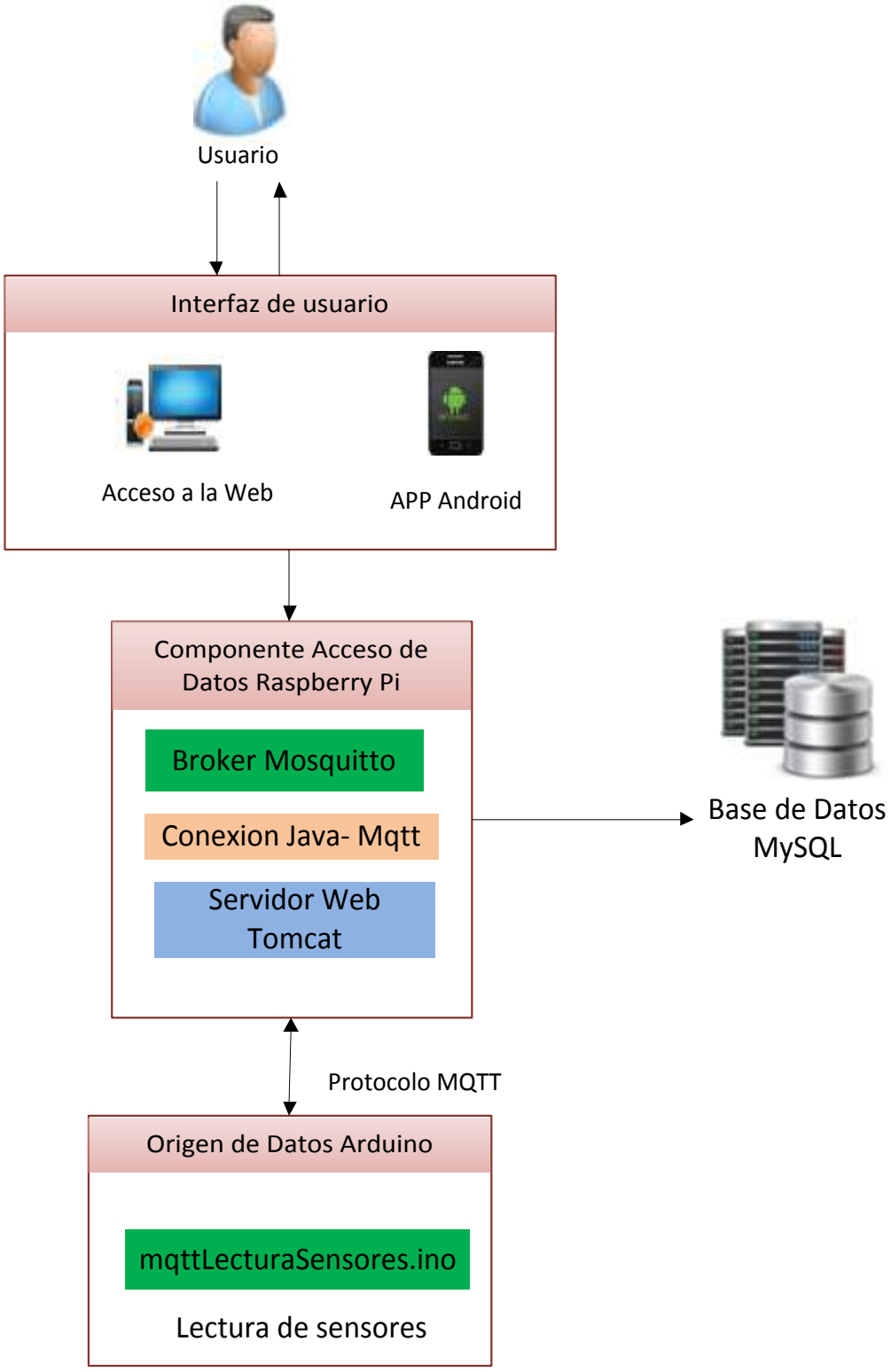


Figura 20. Arquitectura de Software General
Fuente: Autor

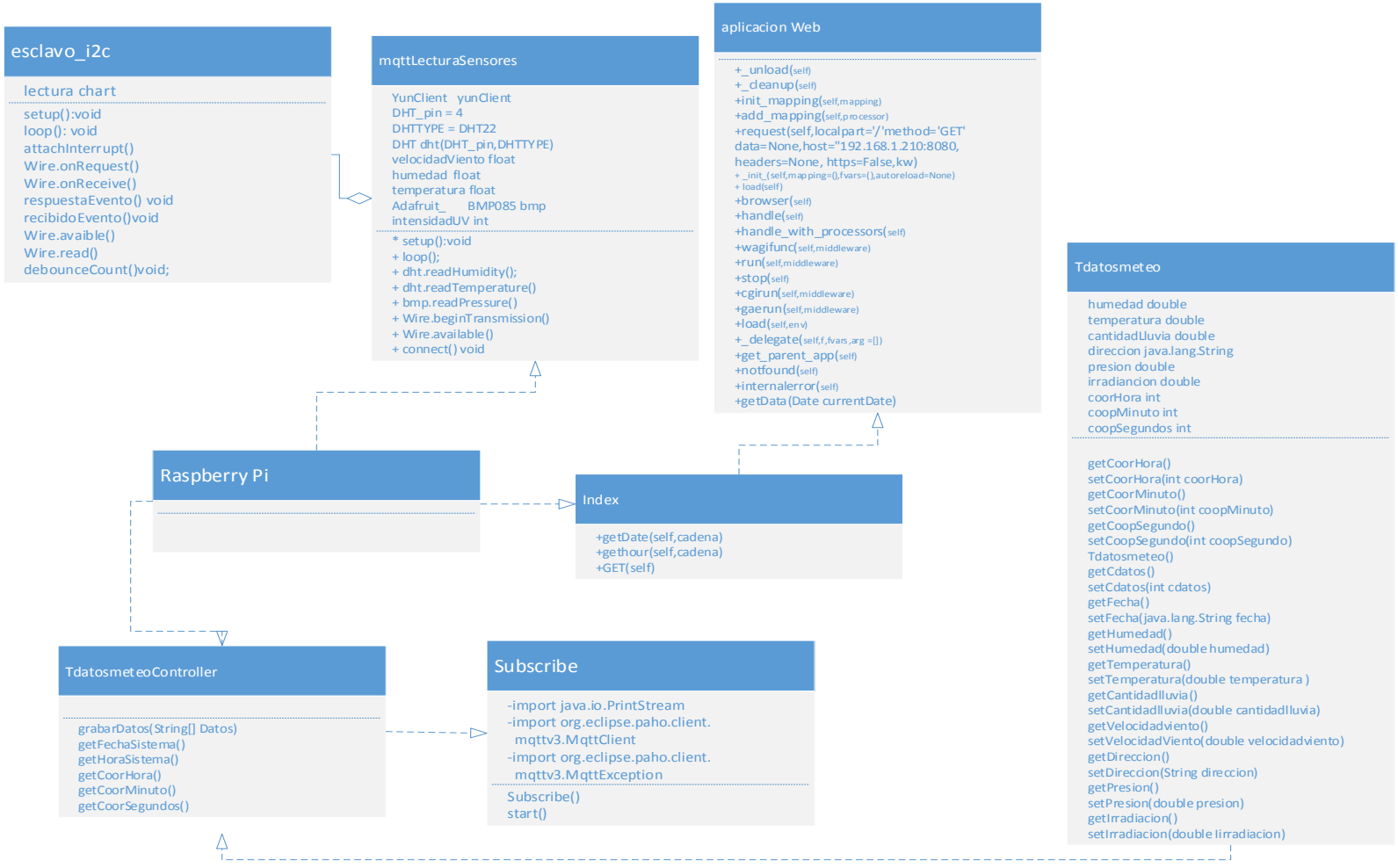


Figura 21. Diagrama de clases
Fuente: Autor

Una vez determinada la arquitectura a nivel de software, se analiza cada una de las partes que compone la misma, con el objetivo de conocer las funciones de los distintos componentes.

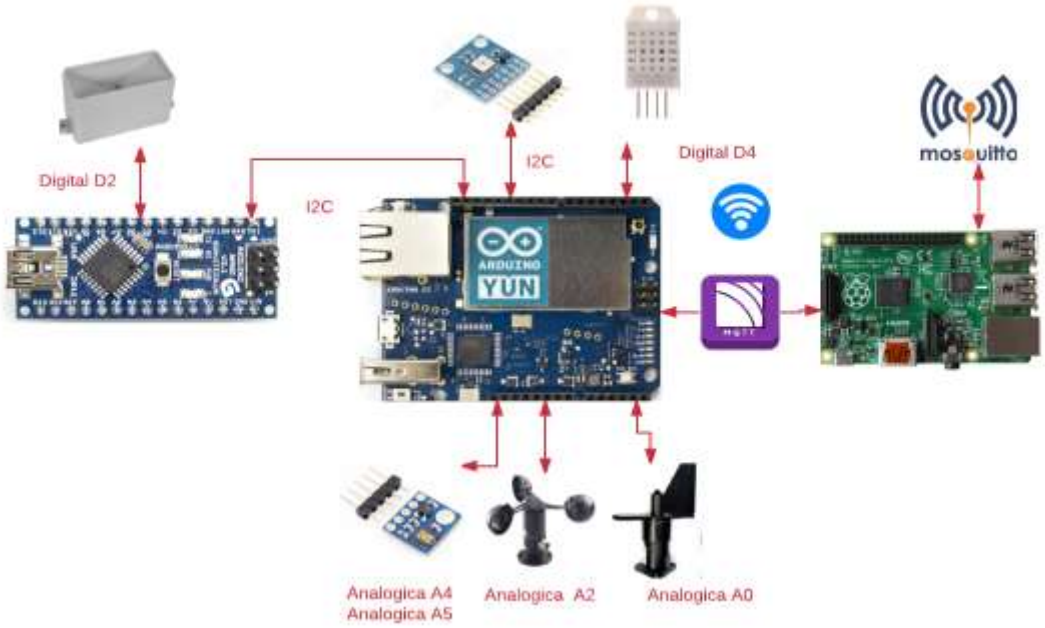


Figura 22. Lectura de datos de los sensores
Fuente: Autor

3.4.2 MQTT Lectura de Sensores con Arduino

Como primer elemento de software tenemos la adquisición y lectura de los datos que provienen de los diferentes sensores.

Para lo cual se utiliza el Entorno de Desarrollo Integrado (IDE) Arduino, ya que este dispositivo cuenta con su propio entorno de programación. Los programas que se realicen es IDE mejor llamados sketches (proyecto con extensión .ino), la estructura básica de este lenguaje es bastante simple, constan de la siguiente estructura:

1. Definición de librerías, constantes y /o variables globales, si estas fueran necesarias.
2. Función Setup, esta parte se encarga de inicializar los elementos y variables, de recoger la configuración, esta se ejecuta una solo vez.
3. Función loop () contiene el programa principal, código que se ejecutara continuamente sobre la placa.

En esta función se ha realizado el programa, al cual se lo ha llamado "MQTTLecturaSensores.ino", el cual contiene la lectura de los diferentes sensores, por sus distintos puertos asignados, además contiene la conexión del Arduino Yún con el protocolo MQTT mediante conexión inalámbrica WIFI para enviar los datos hacia el bróker "Mosquitto" que se encuentra en el Raspberry.

3.4.2.1 Software protocolo MQTT

El protocolo MQTT para Arduino Yun se lo realiza mediante las librerías:

- Yun Client.h
- Bridge.h
- Yun Server.h

Yun Client.h dentro de esta clase creamos a la clase clientes Yun , estas no se llaman directamente, al contrario se invoca cada vez que utiliza una función que depende de esta.

Posterior se crea una instancia de un servidor que va a permitir que el dispositivo Yun escuche a los clientes conectados, Yun Client Yún Client.

En las siguientes líneas se va explicar cada uno de los componentes para la conexión MQTT Arduino-Moquitto:

- **MQTT: Client<IPStack, Countdown> client = MQTT: Client<IPStack, Countdown>(ipstack):** se crea un objeto cliente pasándole una instancia de clase de red
- **void connect ():** esta clase contiene los parámetros para la conexión del Arduino Yún con el Raspberry Pi mediante protocolo MQTT, para aquello se debe ajustar algunos parámetros . Se listan los principales:
    - hostname []: dirección IP del servidor MQTT
    - port: número de puerto por el cual se comunica el protocolo MQTT (1883)
    - rc: código de retorno se una para verificar que la conexión entre servir y cliente fue establecida.
    - MQTTPacket_connectData data = MQTTPacket_connectData_initializer;
    - data. MQTTVersion = 3.1 versión de protocolo por defecto
    - data. clientID.cstring = (char*) "arduino-Yún ": nombre del cliente

3.4.2.2 Métodos adquisición Datos de sensores mediante Arduino

Las funciones y métodos que se usaron para la adquisición de datos de los distintos sensores, son:

- **dht. readHumidity () y dht. readTemperature ().** Funciones que permiten leer los datos provenientes del sensor de humedad y temperatura (DHT22).
    - *readHumidity ()* **devuelve** un número real en el rango de -40 a 80 con precisión de ±0.5. Esta se presenta en grados centígrados
    - *readTemperature ()* devuelve un número real en el rango de 0 y 100 con precisión de 2%. Esta se presenta en %.
    - Para el funcionamiento de esta función se implementa la librería para Arduino *#include <DHT.h.>*.
- **readPressure ().** Función que permite leer la presión atmosférica proveniente del sensor BMP180.
    - *readPressure ()* devuelve un mero real en el rango de 30.000 y 100.000, representada en la unidad de medida Pascales.

  Para el correcto funcionamiento de esta función, se ha implementado la librería #include <Adafruit_BMP085.h>.

  Hay que tener en cuenta que el sensor de presión atmosférica, se comunica mediante el protocolo nativo del Arduino I2C, el cual utiliza la librería <Wire.h>.

- **Intensidad UV.** Este sensor detecta luz con una longitud de onda entre 280-390nm, dentro de este rango se cubre el espectro UV-B como al UV-A. La salida analógica está relacionada linealmente con la intensidad UV (mW/cm2). De este conseguimos una señal analógica lo cual conectado al arduino lo podemos cambiar a un valor de entre 0- 3.3V DC
- **promedioLecturaAnalog ().** Toma 8 valores de los cuales saca promedio para estabilizar un voltaje y este nos dé una medida más exacta.
- **mapfloat (voltajeSalida, 0.99, 2.4, 0.0, 15.0).** La función map es sencilla y bastante útil, nos permite convertir un rango de variación en otro. Esta se aplicó para sacar los niveles de Radiación UV de entre 0 - 15, de un rango de voltaje 0.99 Voltios a 2.4 Voltios.

  Para el cambio de un valor analógico un rango de voltaje de 0-3.3 voltios, se aplica la siguiente ecuación $voltajeSalida = uvLevel * 0.00488281$

La placa Arduino contiene un convertidor analógico a la digital 8 canales lo cual significa que mapeará voltajes de entrada entre 0 y 5 voltios en valores enteros entre 0 y 1023.

- **Velocidad del viento.** Para la obtención de la velocidad del viento, este es tomado de una entrada analógica, se lo realizado de forma similar a la toma de la radicación, se utilizó la función promedioLecturaAnalog **().**

- , Este rango se cambia a voltaje de 0- 5V, cuyo valor según la tabla del fabricante se multiplica x seis (6), obtenido la velocidad del viento en m/s
- **Dirección Viento.** La veleta se comporta como una resistencia variable y mediante un divisor de tensión, se podrá obtener la tensión de los extremos de la veleta. Con el fin de obtener las distintas direcciones se tomarán los valores del divisor de tensión, teniendo en cuenta que la resistencia es de 10 kΩ y la se lo alimenta con una tensión de 5 voltios.

```
float volt= m*0.0048;//(5/1024);
      if (volt>=0&& volt<0.5){
 sprintf(direccion, "%s"," 'NORTE' ");
                }
  else if (volt>=0.6 && volt<1.1){
   sprintf(direccion, "%s"," 'NE' ");
                }

  else if (volt>=1.2 && volt<1.7){
  sprintf(direccion, "%s"," 'ESTE '");
                }

  else if (volt>=1.8 && volt<2.3) {
   sprintf(direccion, "%s"," 'SE' ");
                }

  else if (volt>=2.4 && volt<2.9){
   sprintf(direccion, "%s"," 'SUR' ");
                }

   else if (volt>=3 && volt<3.5){
    sprintf(direccion, "%s"," 'SO' ");
                }

  else if (volt>=3.6 && volt<4.1){
 sprintf(direccion, "%s"," 'OESTE' ");

                }
```

Figura 23. Direcciones del viento
Fuente: Autor

- **Pluviómetro**. Para obtener la medición de la lluvia se usaron interrupciones con cada pulso de lluvia, para esto y con el fin de no crear conflicto con el Arduino YÚN principal, se utilizó una comunicación *Maestro/Esclavo,* mediante la inclusión de un Arduino nano como esclavo**.**
    - **Clase esclavo_I2C**. esta contiene la conexión del esclavo, para enviar la señal por el puerto D2 del Arduino nano. Utiliza las siguientes funciones:
    - Se inicializa utilizando la librería Wire.h.
    - Wire.begin(1). Añadir al bus esclavo, la dirección es obligatoria.
    - Wire.onRequest(recibidoEvento). Activa el evento de lectura del sensor y ejecuta la función.

- void respuestaEvento (). Evento de petición se activa cuando un maestro pide que le enviemos los datos del sensor.[1]
- void recibidoEvento (). Evento de recepción se activa cuando el maestro confirma que todos los datos se han envía de forma correcta.
- **Maestro**. Esta está dentro de la clase MQTTLecturaSensor, para su funcionamiento se utiliza las siguientes líneas de código.
- **Wire.beginTransmission(1).** Comienza la entrega del mensaje al dispositivo esclavo I2C con una dirección antes dada en este caso será 1.
- **Wire.endTransmission**(). Finaliza la entrega a un dispositivo esclavo que comenzó con *beginTransmission* (). Si esta entrega fue exitosa, *endTransmission* () envía un mensaje de detención, Liberando el bus I2C. Si fue erróneo endTransmission () envía un mensaje de reinicio después de la entrega.
- **Wire.requestFrom(1, 1)** Función usada por el maestro para solicitar bytes desde el esclavo. Cabe recalcar que si estos bytes se pierden por algún motivo se los puede recuperar con las funciones available () y read ().

3.4.3 BASE DE DATOS

La base de datos donde se almacenarán los datos recogidos por el Arduino YÚN desde los distintos sensores, se encuentra instalada en la Raspberry Pi. A continuación, se presenta la estructura de la base de Datos creada en Mysql, siendo un sistema de gestión de base de datos de código abierto muy utilizado por los programadores.

```
CREATE TABLE bdmeteo.tdatosmeteo

(
cdatos          int NOT NULL PRIMARY KEY AUTO_INCREMENT ,
fecha           date COMMENT 'fecha lectura',
hora            time COMMENT 'hora de lectura',
humedad         float signed DEFAULT '0' ,
temperatura     float signed DEFAULT '0',
cantidadlluvia  float signed DEFAULT '0',
velocidadviento float signed DEFAULT '0',
direccion       varchar(20),
presion         float signed DEFAULT '0',
irradiacion     float signed DEFAULT ,
coorMinutos      int DEFAULT '0',
```

[1] Arduio nano placa pequeña, basada en el ATmega328. Esta versión fue pensada para usarla en protoboard, sus pines facilitan la conexión de los componentes si necesidad de muchos cables El Nano fue diseñado la empresa Gravitech.

```
coorSegundos    int DEFAULT '0'
);
```

Una vez creada la tabla que contendrá los diferentes valores, los campos se han definido como tipos float, por la naturaleza de datos, otros como entero y un valor como cadena de texto o *string*

3.4.4 IMPLEMENTACIÓN DE SOFTWARE JAVA

Otro de los componentes de software del proyecto es el lenguaje de programación JAVA, la descripción corresponde a las clases principales utilizadas para este proyecto:

Clase Suscriber (). Contiene los principales elementos para llevar a cabo la conexión del protocolo MQTT y JAVA, para lo cual necesita las siguientes librerías:

- **Import org.eclipse.paho.client.MQTTv3.MQTTClient**. Librería que contiene una interfaz de programación el cual nos da paso para realizar la comunicación a un servidor MQTT. En este proyecto nuestro bróker es "Mosquitto", estando instalado sobre el Raspberry Pi.
- **Import org.eclipse.paho.client.MQTTv3.MQTTException**. Librería que nos muestra error en caso de no existir conexión con el servidor.

Los métodos de usados en la clase Suscriber son:

- **Subscriber () void.** Contiene las propiedades de conectividad del topic de MQTT enviado por Arduino hacia el bróker "mosquitto". Para lo cual es necesita conocer:
  - String BROKER_URL, describe el nombre de DNS o el Puerto IP.
  - client username, describe el nombre de usuario o cliente al cual se va enviar la información tomada por los diferentes sensores.
  - **start ().** Recibe la respuesta desde el cliente. Recibe los datos provenientes de los sensores.
  - **Clase Conexión.** Clase donde se define la conexión hacia la base de datos mySql, para su posterior ingreso de datos hacia la misma.
  - **Clase ManagerConexion ().** Clase que contiene los diferentes métodos para la conexión e inserción de los datos en la base mySql.
  - **getConexion ().** Este método es un objeto tipo Connection, que se utiliza para crear las diferentes sentencias SQL, para luego ser enviadas a la base de datos y poder procesar los resultados.
  - **Clase Controller ().** Contiene los métodos para guardar los datos.

3.5 Definición de Envío de Datos a la WEB

Una vez realizada la programación de conexión, adquisición y registro de los datos de la estación meteorológica, mediante Java y la base de Datos Mysql, Se describirá el manejo de los datos para ser mostrados por medio de la WEB. Esta consiste en el montaje de un servidor WEB sobre un Raspberry pi.

El objetivo principal del enviar los datos a una página WEB es brindar al usuario final una forma amigable de visualizar los datos recogidos por los diferentes sensores, así como obtener los reportes necesarios en el tiempo.

3.5.1 ESTRUCTURA DE ENVÍO DE DATOS.

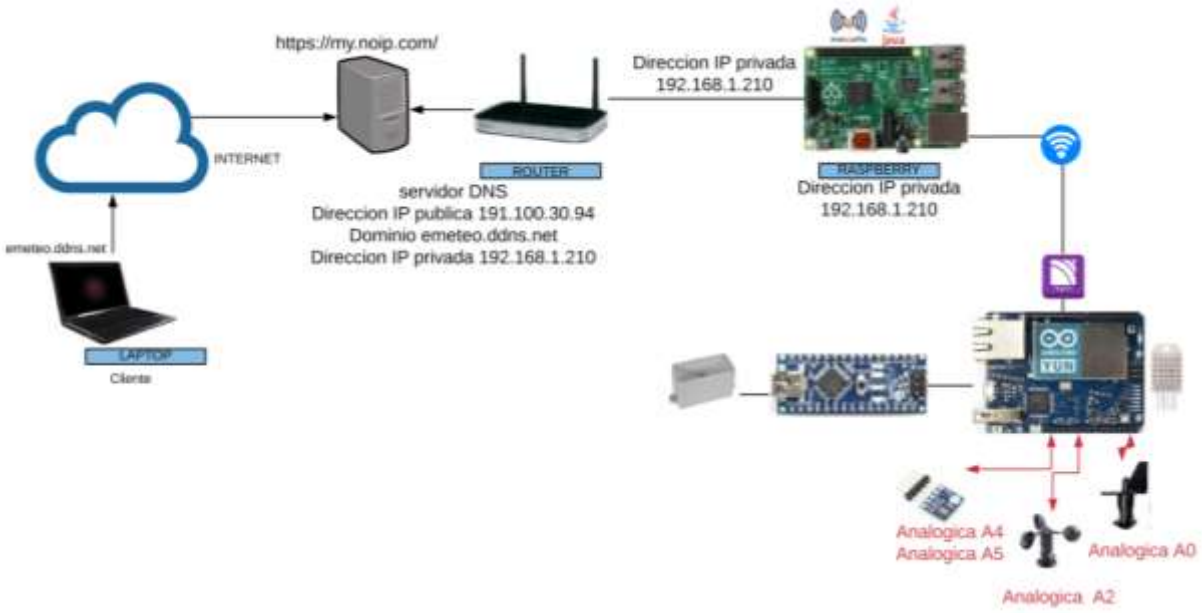


Figura 24. Diagrama Estructura WEB
Fuente: Autor

Para visualización de los datos de la Estación Meteorológica se ha basado en la estructura de la Figura. 24, la cual se determinó de la siguiente manera:

3.5.1.1 Montaje de servidor WEB.

Se realiza la configuración de las diferentes componentes para el correcto funcionamiento del servidor WEB.

Servidor Tomcat

Se ha escogido el Servidor Tomcat [25] este es un servidor WEB con soporte de servlets (clase de lenguaje de programación Java) y JSPs (JavaServer Page). Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor WEB Apache [26].

Tomcat puede funcionar de manera autónoma como motor de aplicaciones WEB desarrolladas con Java, pero habitualmente se usa en combinación con otros productos como el servidor Web Apache, para dar un mayor soporte a tecnologías y aumentar sus características [27].

Se debe tomar en consideración que la Raspberry Pi debe tener una dirección IP estática. La cual la podemos observar con comando "ifconfig", en este caso será la dirección asignada para el desarrollo es: IP 192.168.1.210.

3.5.2 APACHE –TOMCAT

Figura 25. Apache
Fuente: https://www.upguard.com/articles/15-ways-to-secure-apache-tomcat-8

Instalación Apache-Tomcat Raspberry pi

La instalación en la Raspberry Pi se la ejecuta Instalando los paquetes necesarios para el correcto funcionamiento de la misma.

```
apt-get install apache2 mysql-server  openjdk-6-jre
```

Seguidamente se procede a descargar Apache- Tomcat

```
wget https://archive.apache.org/dist/tomcat/tomcat-8/v8.5.16/bin/apache-tomcat-8.5.16.tar.gz
```

Se lo descomprime mediante:

```
tar xvzf apache-tomcat-8.5.16.tar.gz
```

Para que el servicio de Apache- Tomcat arranque desde el inicio del Raspberry se debe crear el script con el siguiente comando *sudo /etc/init.d/tomcat*; ejecutándose:

```
#!/bin/sh
# /etc/init.d/tomcat
# starts the Apache Tomcat service
### BEGIN INIT INFO
# Provides:          tomcat
# Required-Start:
```

```
# Required-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# X-Interactive:     true
# Short-Description: Start/stop tomcat application server
### END INIT INFO
export CATALINA_HOME="/home/pi/servidor/Tomcat/apache-tomcat-8.5.16"
case "$1" in
start)
  if [ -f $CATALINA_HOME/bin/startup.sh ];
  then
    echo $"Starting Tomcat"
    /bin/su pi $CATALINA_HOME/bin/startup.sh
  fi  ;;  stop)
  if [ -f $CATALINA_HOME/bin/shutdown.sh ];
  then
    echo $"Stopping Tomcat"
    /bin/su pi $CATALINA_HOME/bin/shutdown.sh
  fi ;; *)
  echo $"Usage: $0 {start|stop}"
  exit 1  ;;
esac
```

Una vez reiniciado el servicio de Apache-Tomcat, se procede con la instalación de la Base de datos, la cual se la instala con el siguiente comando:

```
sudo apt-get install mysql-server  mysql-client php5-mysql
```

En el proceso de instalación ira pidiendo diferentes parámetros de Configuración como la clave del administrador, contraseña, etc. Para que todas las configuraciones se ejecuten satisfactoriamente siempre se debe reiniciar el Raspberry.

Para comprobar que el Mysql este correctamente instalado se ejecuta el siguiente comando:

```
Mysql –u root -p
```

3.5.3 CREACIÓN DE LA BASE DE DATOS Y TABLA DE LA ESTACIÓN (METEO)

Para el almacenamiento de los datos recogidos por los distintos sensores, es necesario la creación de una base de datos la cual debe está conformada al menos por una tabla que contenga las columnas necesarias para el almacenamiento de los distintos valores.

En este caso la Base de Datos y tablas se las crea mediante comandos Mysql.

Para la creación de la tabla de datos se utiliza la siguiente estructura:

Nombre de campo	Tipo de dato	Descripción
cdatos	Int	Clave Primaria
fecha	Date	Fecha de la lectura del dato
hora	Time	Hora de la lectura del dato
humedad	Float	Cantidad de humedad
temperatura	Float	Cantidad de temperatura
cantidadlluvia	Float	Cantidad de lluvia
Velocidadviento	Float	Velocidad del viento
direccion	Varchar	Dirección Viento
presion	Float	Cantidad de presión atmosférica
irradiacion	Float	Cantidad de irradiación
coorHora	Int	Variable de la hora para gráficos
coorMinutos	Int	Variable de la minutos para gráficos
coorSegundos	Int	Variable de segundos para gráficos

Tabla 3. Datos de almacenamiento
Fuente:Autor

Los datos de temperatura, humedad, cantidad de lluvia, velocidad del viento, presión e irradiación son de tipo float, poseen parte decimal; el dato de dirección del viento es de tipo *varchar* ya que contiene texto como por ejemplo Norte, Sur, etc; *coorHora, coorMinuto y coorSegundo*, son de tipo entero, contendrá las horas, minutos y segundo por separados para las gráficas de los sensores.

El script utilizado para la creación de la tabla *tdatosmeteo* es el siguiente:

```
CREATE TABLE bdmeteo.tdatosmeteo
(
cdatos int NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT 'Clave primaria',
fecha date COMMENT 'fecha lectura',
hora time COMMENT 'hora de lectura',
humedad float signed DEFAULT '0' COMMENT 'cantidad humedad 0 a 100 se mide en porcentaje p(%)',
temperatura float signed DEFAULT '0' COMMENT 'cantidad temperatura 0 a 100 se mide grados centigrados gc',
cantidadlluvia float signed DEFAULT '0' COMMENT 'cantidad 0 a >100 se mide en milimetros mm',
velocidadviento float signed DEFAULT '0' COMMENT 'cantidad 0 a >100 se mide kilometros por hora kh',
direccion varchar(20) COMMENT 'dirección del viento NORTE, SUR, ESTE, OESTE',
```

```
presion         float signed DEFAULT '0' COMMENT 'valor 0 a >100 se mide libra-fuerza por pulgada cuadrada psi',
irradiacion     float signed DEFAULT '0' COMMENT 'valor 0 a >100',
coorHora        int DEFAULT '0',
coorMinutos     int DEFAULT '0',
coorSegundos    int DEFAULT '0'
);
```

3.5.4 CONEXIÓN BASE DE DATOS Y JAVA-MQTT

Para guardar las mediciones de los sensores en la base de datos y mantenerlos para su respectivo uso, se usó el lenguaje de programación Java con la plataforma de programación Eclipse.

Una de las ventajas de utilizar Java es que es de código abierto, tiene mucho soporte, es de fácil manejo y dispone de librerías para las distintas aplicaciones.

Para la conexión de la base de datos, primero se debe descargar un API (interfaz de programación de aplicaciones) de conexión para nuestra IDE, tomando en cuenta que esta es la interfaz permite realizar la conexión de un servidor de base de datos hacia nuestra aplicación Java, esta se denomina también JDBC(Java DataBase Connecticity).

Se realiza la importación de los siguientes paquetes:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

Estos paquetes tienen las siguientes funciones dentro de la clase “Conexión”:

- *java.sql. Connection*. - Realiza la conexión con la base de datos
- *java.sql.DriverManager*.- Gestor de la base de datos
- *java.sql. SQLException*.- Pproporciona información sobre un error de acceso a la base de datos.

Para llevar a cabo la una correcta conexión con la base de datos en [28] nos indica que se debe ejecutar los siguientes pasos:

1. Abrir la conexión y crear un puntero
2. Ejecutar una consulta
3. Hacer efectiva la escritura en el caso de insertar, actualizar o eliminar datos.
4. Cerrar la conexión y el puntero

5. En el paso uno (1) se debe pasar los parámetros para la conexión con la base, estos son el host de la base de datos, usuario, contraseña y el nombre de la base, todo este desarrollo se lo realizó en la Clase *Conexión.* Descrita en el Anexo 3

Para insertar los datos en la base de datos "*dbmeteo*", se la ha realizado en La clase, llamada *"ManagerConexion.java*", en la cual se encuentra la sentencia Mysql para la inserción de datos en la base. Código descrito en el Anexo 3.

En la clase *ManagerConexion.java* la sentencia *Insert*, la cual se debe ejecutarse varias veces durante una sesión y sobre una misma tabla siempre, por esta razón y para cuidar la integridad de los datos se ha utilizado la sentencia "*prepared statement*", este componente hace que la base datos se vuelva más eficiente al ejecutar un proceso o una sentencia, de forma que la base de datos la "precompila" y la guarda para que esta sea ejecutada inmediatamente.

Para que el "*prepared statement"* funcione de forma efectiva se ha creado una instancia de nombre "*preparedStmt*" con la sentencia SQL del INSERT, en la cual se ha reemplazado todos los valores de la base de datos por interrogantes, al crear la "*PreparedStatement*", se indica que sentencia SQL se va ejecutar y qué campos posee, ya que se tiene métodos específicos como son, *setInt* (), *setString* (), etc, identificando los valores de los campos y la sentencia a ejecutar

3.5.5 DISEÑO DE LA PÁGINA WEB

Para lograr la visualización de los datos de los distintos sensores, de forma sencilla y rápida, se va a crear una página WEB, en la cual se podrá mostrar los datos tanto de forma gráfica como también en una tabla, el usuario será capaz de sacar reportes mediante fechas puntuales a consultar.

Para este fin se ha utilizado la herramienta de desarrollo WEB "Webratio", esta es una herramienta de fácil manejo para aplicaciones WEB, ya que estas se las realiza de una forma visual ahorrando hasta el 80% de los tiempos y los costos en comparación con métodos tradicionales.

Figura 26. WebRatio

Fuente: https://www.webratio.com/site/content/es/home

El proyecto WEB esta construido bajo la siguiente lógica de modelado IFML (standard Interaction Flow Modeling Language) como evolución del WebML, este lenguaje visual creado por WebRatio y usado para la generación automática de aplicaciones WEB. Para la creacion de la pagina WEB la estacion meteorologica se ha ha tomado en cuenta 4 puntos fundamentales que son los siguientes:

- Modelado de Dominio Domain model
- Vista del Sitio Site view
- Vista de Servicio Service view
- Definiciones de módulos Module definitions

3.5.5.1 Modelado de Dominio (Domain model)

Describe la organización de datos de la aplicación, almacena la información obtenida del usuario, y la conexión a la base de datos, una vez conectado satisfactoriamente se pódra interactuar con las tablas del esquema, al cual se haya configurado. Esta puede estar representada ya se en UML (lenguaje unificado de modelado) o ER (entidad relación).

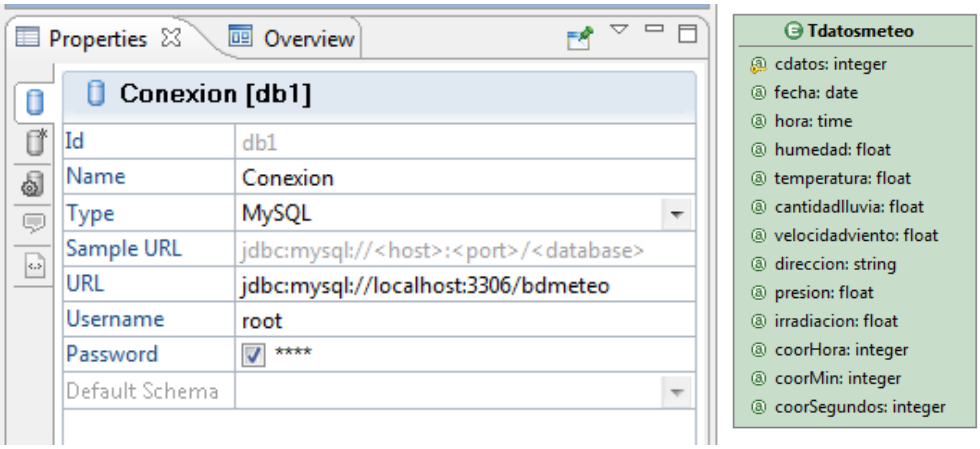


Figura 27. Conexión BD Tdatosmeteo WebRatio

Fuente: Autor

3.5.5.2 Site view (Public)

Se da a conocer la parte autónoma de un modelo de flujo de interacción, En este caso Página Estación contiene el diagrama de flujo diseñado para mostrar la sección datos meteorológicos y sus diferentes gráficas. Cuya implementación fue realizada através de la librería de jquery "*highcharts*" incorporado en el template de la página html.

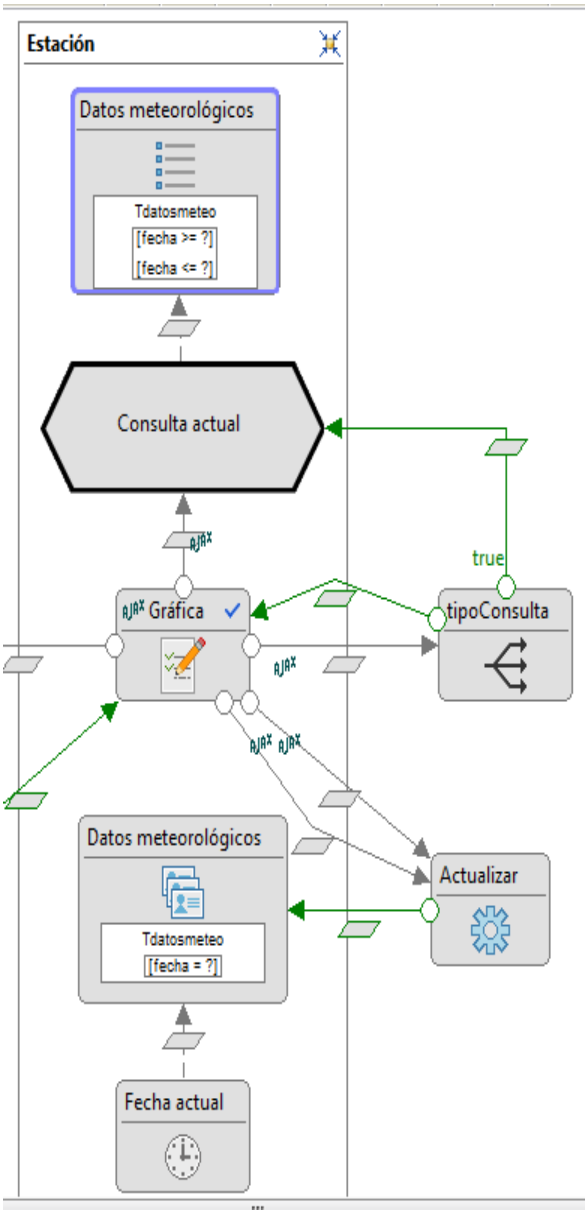


Figura 28. Diagrama de flujo de Pagina Web

Fuente: Autor

Figura 29. Pagina WEB Estacion Meteorologica

Fuente: Autor

Para realizar una vista del sitio , esta se divide en distintos componentes, lo cuales hemos utilizados en este proyecto

List

El cual tiene como parámetro de entrada el atributo fecha de la tabla *tdatosmeteode* (fecha desde, fecha hasta), visualiza los datos de la sección Datos meteorológicos. Si la fecha no se personaliza en la página por defecto muestra los 10 últimos registos de datos, en la tabla de la pagina WEB.

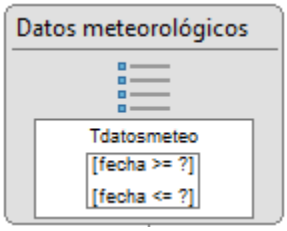


Figura 30. Vista del componentes para presentacion de Datos

Fuente: Autor

Multiple details

El cual tiene como parámetro de entrada el atributo fecha de la tabla *Tdatosmeteo*, esta nos ayuda para vizualiza las graficas de temperatura, cantidad de lluvia, radiación, velocidad de viento, humedad, presión. Por defecto se muestran los valores de la hora actual en el eje X, y los distintos valores monitoreados de los sensores en la eje Y.

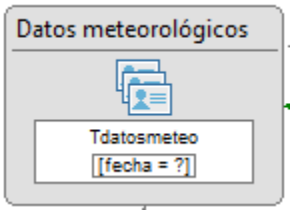


Figura 31. Plantilla de selección
Fuente: Autor

Form

Es utilizado para mostrar el formulario de ingreso de fechas (Fecha desde, fecha hasta) cuando se opta por la opción de personalizar la consulta, además activa el flujo de prodesamiento de datos por medio ajax (funcion de llamada que implementa acciones interactivas)para refrescar las gráficas o la tabla de datos meteorológicos, según corresponda la acción .

Figura 32. Form Grafica
Fuente: Autor

Time

Es utilizado para devolver la fecha actual cuando la página es cargada.

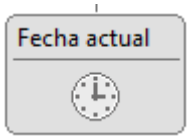


Figura 33. Fecha con la que se carga la pagina
Fuente: Autor

Consulta actual

Módulo de operación diseñado para devolver el rango de fechas al momento de personalizar una consulta. La página de reportes se imprimira en un formato pdf, despues de que se realice una consulta, cuya implementación se lo hace a traves de la librería *iText*, posee los mismos componentes explicados anteriormente.

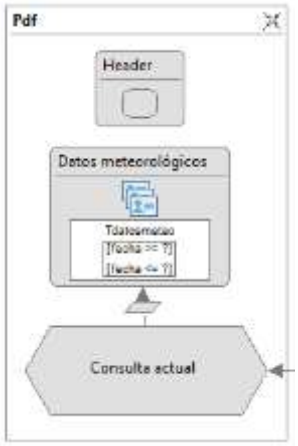


Figura 34. Consultas
Fuente: Autor

Service view (Servicios)

Se ha creado un servicio *datosTemperatura* para poder interactuar con la aplicación móvil en el cual consiste de un servicio REST (proveedor de servicio WEB) que tiene un método GET (envía datos mediante una dirección WEB) y que recibe como parámetros la fecha y la hora, ejecutando la consulta a la base de datos, devolviendo la respuesta en formato JSON (formato de texto ligero para envío de datos).

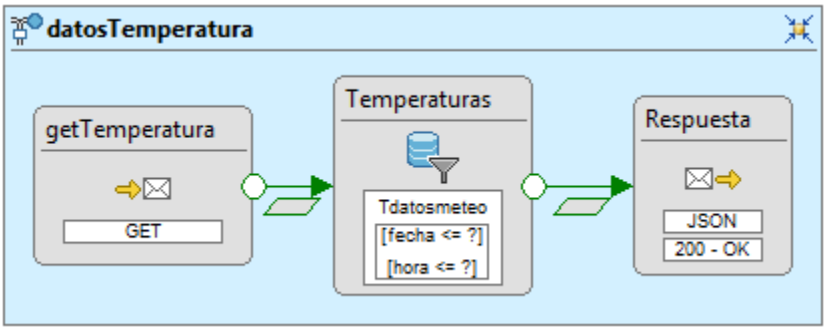


Figura 35. Interacción con la aplicación móvil
Fuente: Autor

Module definitions

Se detallan todos los módulos creados de acuerdo a la necesidad de la aplicación. Para este proyecto se lo uso para evaluar si requiere una consulta mayor a la presentada en pantalla, esta se las realiza por rango de fechas , en caso de haber una consulta, devolverá el reporte o caso contrario devolverá la fecha actual.

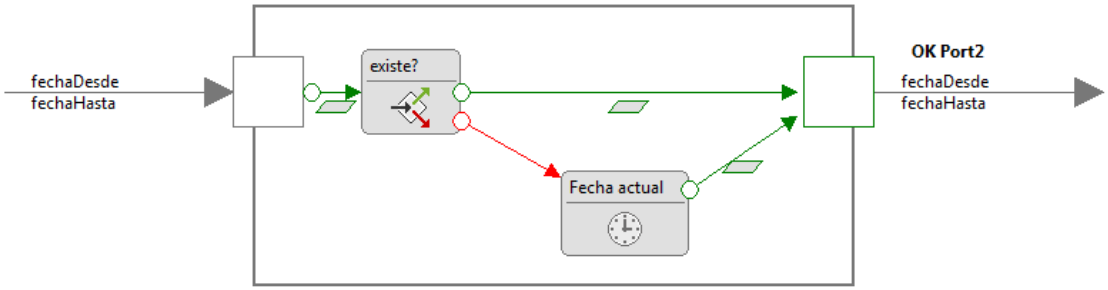


Figura 36. Estructura del modulo (Consulta actual)
Fuente: Autor

3.5.6 PUBLICACIÓN DE LA WEB

La página es publicada en la WEB, la cual esta pone a disposición los datos meteorológicos del proyecto, a cualquier usuario con acceso a Internet, al pasarlos de estar alojadas en el servidor Raspberry PI. Para esto se realizó lo siguiente:

Los archivos *WAR* (WEB application Archive) son utilizados para distribuir aplicaciones WEB, de un archivo ejecutable. [29]

Al haber sido diseñado y desarrollado la página WEB en **Web Ratio**, esta entrega un archivo. *JSP (JavaServer Page)* el cual se transformará en un archivo WAR, esta se colocará dentro de la carpeta *webapps* donde instalamos Apache Tomcat.

/home/pi/servidor/apache-tTomcat.8.5.16/webapps

Una vez situado el archivo este dentro de la carpeta *webapps*, se puede abrir la aplicación desde cualquier navegador WEB, poniendo la dirección (dirección IP del Raspberry), puerto de entrada al servidor y el nombre asignado al proyecto de la aplicación y el nombre del archivo que contiene la página WEB. Este proyecto en la fase de diseño está en la siguiente dirección

http://192.168.1.210:8080/Temperatura

3.5.7 CONFIGURAR APACHE PROXY

Es necesario que la página WEB se la pueda acceder desde cualquier sitio que cuente con una conexión a internet, para lo que se utiliza Apache como un servidor proxy, que básicamente va a permitir acceder a los recursos desde otros servidores a través de él mismo; esto lo realiza enrutando todo el tráfico y enviándolo al servidor correspondiente.

Para este proyecto es muy útil porque lo que se quiere lograr es el de exponer la página WEB, fuera de una red privada.

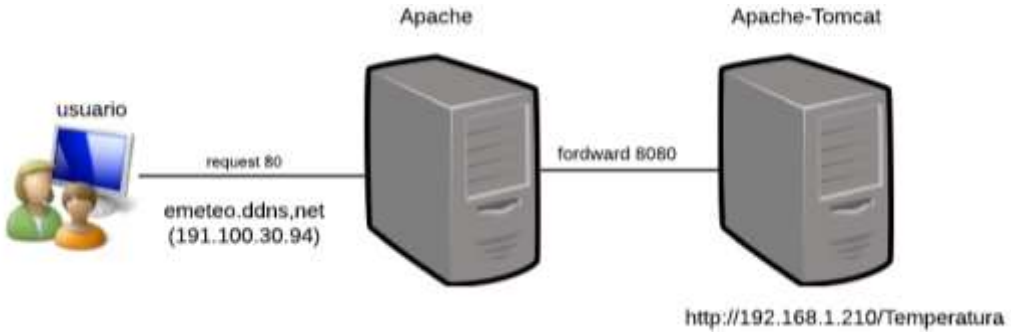


Figura 37. Configuración Apache Proxy
Fuente: Autor

Para la configuración de apache proxy se tiene que modificar el archivo "*000-default. conf*", que se encuentra dentro de la carpeta /*etc/apache2/sites-available*.

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumenRoot /var/www/html
    ProxyPass / Temperatura http://localhost:8080/Temperatura
    ProxyPassReverse  / Temperatura http://localhost:8080/Temperatura
```

```
    ProxyRequests Off
     ProxyPreserveHost On
     <location "/">
        Order deny,allow
        Allow from all
     </location>
  </VirtualHost>
```

Con la modificación de este archivo se ha logrado que al ingresar al http://localhost/ el proxy redireccione el tráfico y muestre el contenido, en este caso de /*Temperatura*

Hasta este punto la página WEB visualizará de forma local, desde el navegador, http://19.168.1.210/Temperatura. Para los usuarios es difícil recordar una dirección IP, por lo que se cambiará esta dirección por un nombre fácil de recordar.

3.5.8 CONFIGURACIÓN NO-IP

Lo más sencillo es recordar un nombre que una dirección IP, para ello se usara un servidor DNS para traducir estos nombres por la dirección IP, a lo que se denomina nombre de dominio. Un nombre de dominio, no es más que una cadena de caracteres utilizada para referirnos a una máquina [30], pero este tiene un costo elevado, tenemos más alternativos como los subdominios que son gratuitos y nos brindan el mismo servicio. Para este proyecto se ha escogido los servicios de NO-IP ya que es gratuito y sin publicidad.

Primero, se debe crear una cuenta en https://www.noip.com

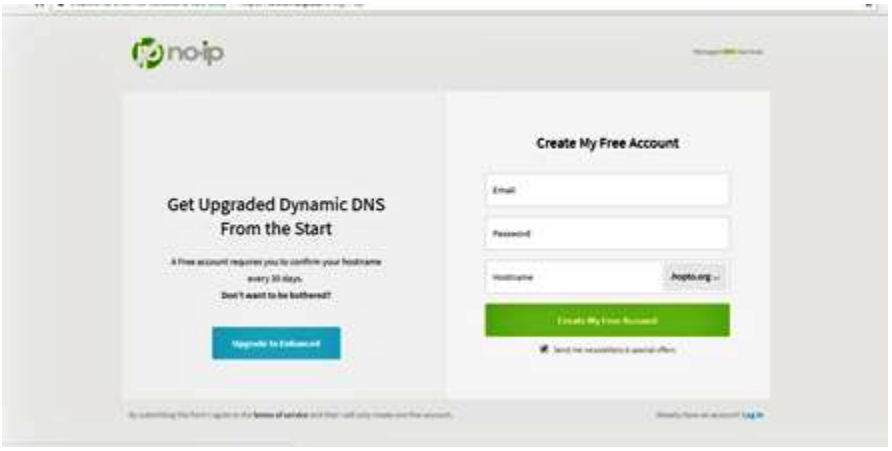


Figura 38. Creación de cuenta noip.com
Fuente: https://www.redeszone.net/redes/host-en-no-ip-manual-para-crear-un-dynamic-dns-con-no-ip/

Luego se crea un *hostname*, en donde se coloca el nombre con el cual se mostrará la página. Para el proyecto se ha elegido "emeteo", además se debe escoger un subdominio, el mismo que tiene muchas opciones, para este proyecto se ha escogido "ddns.net".

Figura 39. Añadir un Hostname no-ip
Fuente: Autor

Para configurarlo en la Raspberry PI, es necesario instalar el cliente de la siguiente manera:

Descargamos el software

```
wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
```

Descomprimirlo:

```
tar -zxvf noip-duc-linux.tar.gz
Instalarlo:
cd noip-2.1.9-1/
make
sudo make install
```

```
pi@raspberrypi ~ $ cd noip-2.1.9-1/
pi@raspberrypi ~/noip-2.1.9-1 $ make
gcc -Wall -g -Dlinux -DPREFIX=\"/usr/local\" noip2.c -o noip2
noip2.c: In function âdynamic_updateâ:
noip2.c:1595:6: warning: variable âiâ set but not used [-Wunused-but-set-variable]
noip2.c: In function âdomainsâ:
noip2.c:1826:13: warning: variable âxâ set but not used [-Wunused-but-set-variable]
noip2.c: In function âhostsâ:
noip2.c:1838:20: warning: variable âyâ set but not used [-Wunused-but-set-variable]
pi@raspberrypi ~/noip-2.1.9-1 $ sudo make install
if [ ! -d /usr/local/bin ]; then mkdir -p /usr/local/bin;fi
if [ ! -d /usr/local/etc ]; then mkdir -p /usr/local/etc;fi
cp noip2 /usr/local/bin/noip2
/usr/local/bin/noip2 -C -c /tmp/no-ip2.conf

Auto configuration for Linux client of no-ip.com.

Please enter the login/email string for no-ip.com
Please enter the password for user

Only one host                              is registered to this account.
It will be used.
Please enter an update interval:[30]  30
Do you wish to run something at successful update?[N] (y/N)  N

New configuration file '/tmp/no-ip2.conf' created.

mv /tmp/no-ip2.conf /usr/local/etc/no-ip2.conf
```

Figura 40. Configuración noip en la Raspberry Pi
Fuente: Autor

En el proceso de instalación pedirá una serie de requisitos como son el nombre con el que se creó la cuenta y la contraseña. Una vez configurado debe tener una consideración importante, que el puerto 80 del router este habilitado, para que la página WEB pueda ser mostrada fuera de la red de área local

3.6 Configuración Protocolo MODBUS

Para la Configuración de protocolo MODBUS dentro del proyecto se utilizó las siguientes herramientas de software:

3.6.3 NODE-RED

Esta es una herramienta de programación visual que muestra las relaciones y funciones, permitiendo al usuario programar sin tener que escribir ningún lenguaje de programación. Node-RED es un editor de flujo basado en el navegador donde se puede añadir o eliminar nodos y conectarlos entre sí con el fin de hacer que se comuniquen entre ellos. [31]

En el sistema operativo Raspbian este viene configurado. Se lo abre desde Inicio →Programación →Node-RED. Basta con abrir un explorador de internet y colocar en la barra de direcciones la dirección IP local y el puerto 1880 ejemplo: 192.168.1.210:1880

3.6.2 MODBUS SLAVE

Es un simulador de protocolos MODBUS ESCLAVO, sirve para comprobar la comunicación modbus del maestro hacia el ESLAVO; este simulador tiene la capacidad de captar hasta 32 dispositivos esclavos en 32 ventanas diferentes, para lo cual se inicia la programación y realiza la prueba antes de recibir su dispositivo esclavo del proveedor final [32]. La aplicación se la podrá descargar de forma gratuita desde http://www.modbustools.com/download.html

Para él envió de datos al SCADA el MODBUS SLAVE realiza la misma función de ESCLAVO el cual es interrogado por el sistema SCADA creando una circulación de información maestro-esclavo

3.6.3 SCADALAQUIS.

Es un programa para la adquisición de datos, supervisión de procesos, automatización y control industrial. El sistema SCADA se utiliza normalmente para la recepción de señales bajo protocolo de comunicación, transferidos desde un PLC o en este proyecto desde el sistema Arduino – Raspberry,

El SCADA tiene la capacidad de mostrar en forma gráfica al usuario los datos que recibe, la adquisición de datos la realiza en tiempo real e incorpora una estampa de tiempo a cada dato recibido e integra en una base de datos, en la cual se puede hacer consultas determinadas, con la capacidad de graficar los datos de reportes.

Para el caso del SCADALAQUIS utiliza protocolos Modbus en sus versiones MODBUS TCP IP, MODBUS SERIAL, MODBUS ASCCI, y tiene drives para manejar una serie de dispositivos entre los más conocidos PLC Allen Bradley, Micrologic, OPC, etc.

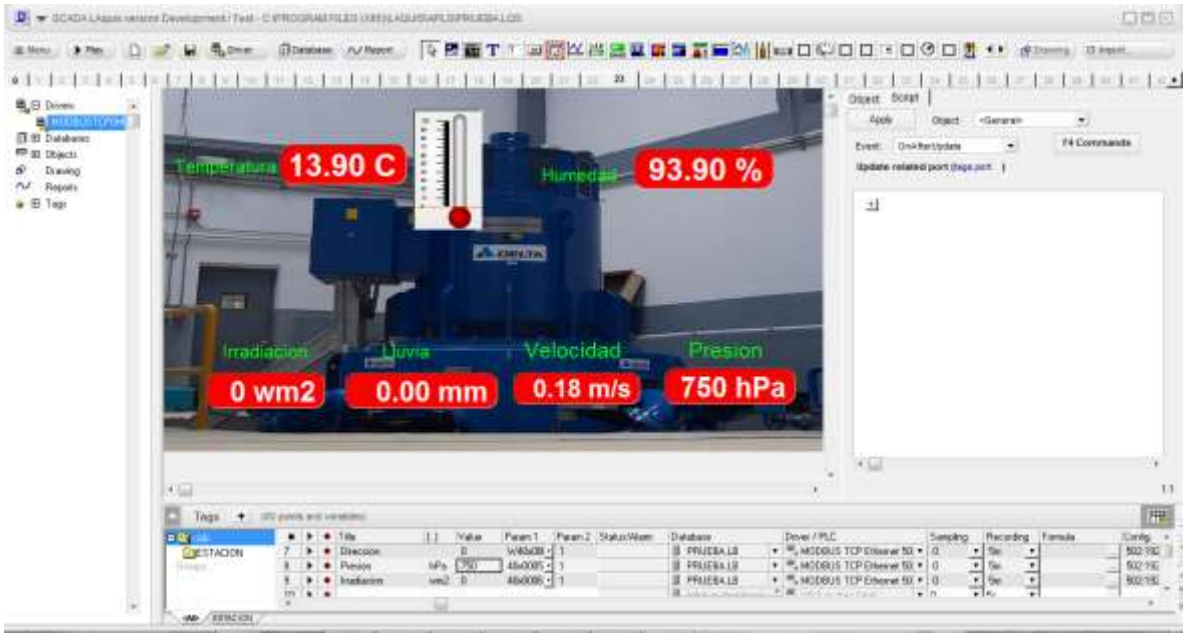


Figura 41. Pantalla Grafica SCADA Laquis
Fuente: Autor

3.6.4 ARQUITECTURA MQTT -NODE RED - MODBUS

Para la conversión de los datos desde el protocolo MQTT hacia el protocolo modbus se ha utilizado la siguiente arquitectura.

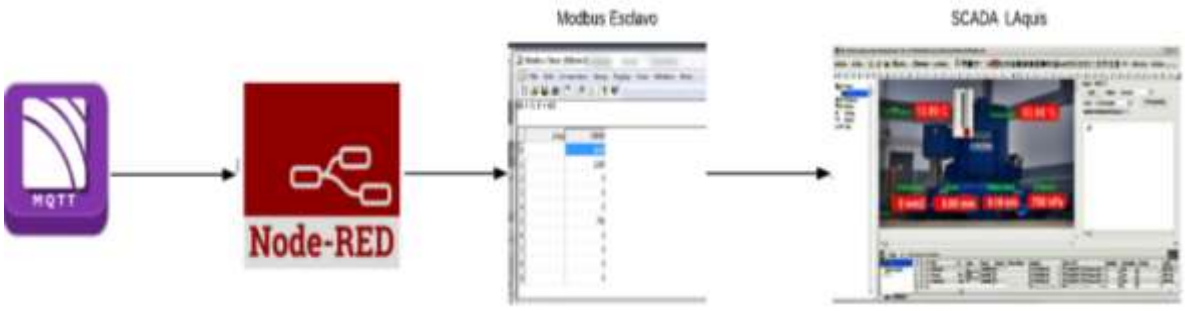


Figura 42. Arquitectura de Comunicación MQTT –MODBUS
Fuente: Autor

En el esquema se describe la recepción de los datos mediante el protocolo MQTT, desde el Arduino hacia el Raspberry PI, se realiza la lectura de los mismos mediante los nodos modbustcp del programa Node Red, el cual convierte al protocolo MODBUS TCP/IP como servidor, teniendo disponibilidad de datos para el Raspberry, el cual puede comunicarse hacia el simulador esclavo “ModbusSlave”; por otra parte, el Programa SCADA, interroga al

esclavo Modbus, recogiendo los datos y guardándolos en una Base de datos, para su publicación en tiempo real a través de las pantallas del SCADA.

DESARROLLO DE LA INTEGRACIÓN DE LA ESTACIÓN METEOROLÓGICA AL PROTOCOLO MODBUS

3.6.5 PROGRAMACIÓN NODE RED

En el programa Node Red, se ha configurado varios nodos como se muestra en la Figura 45.

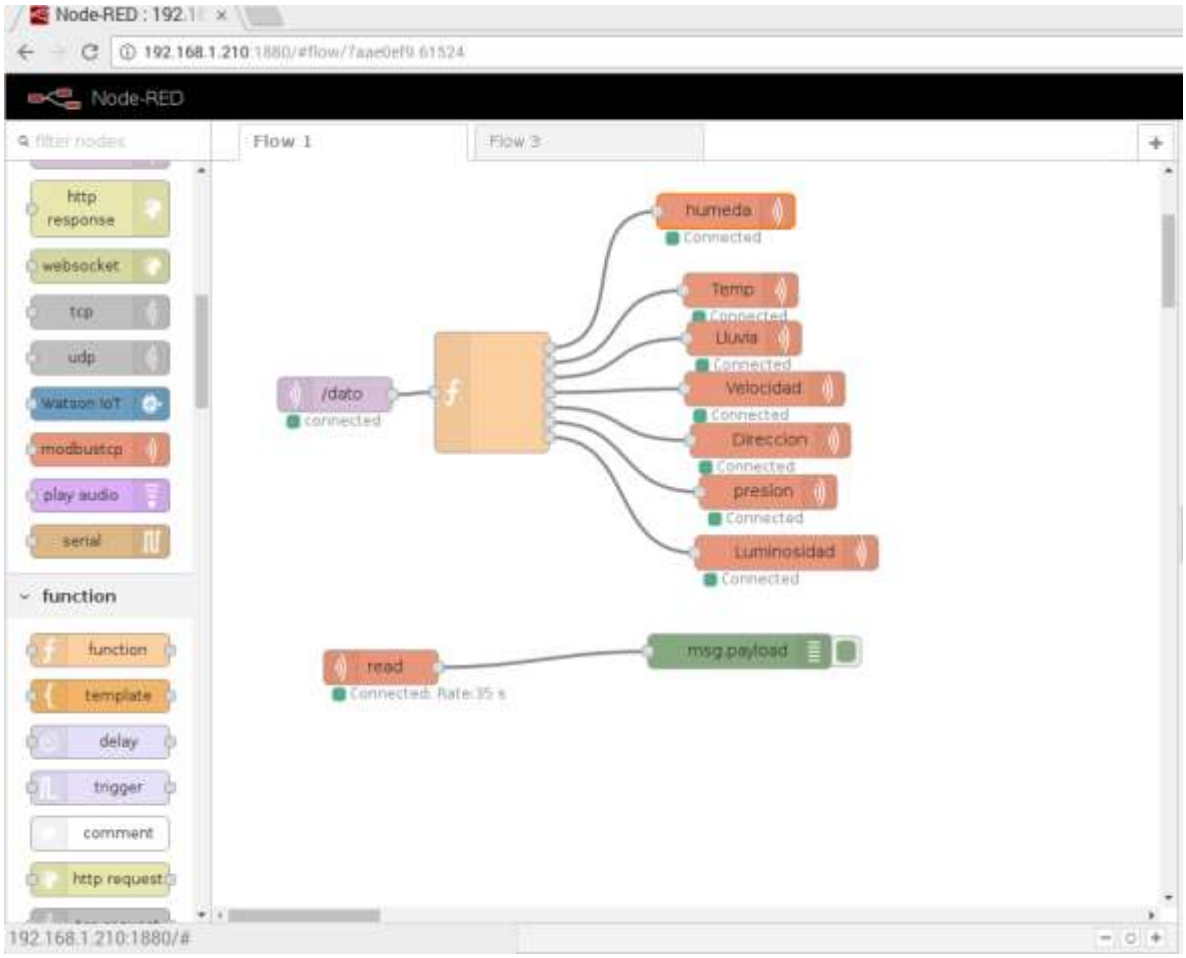


Figura 43. Configuración Node Red
Fuente: Autor

Nodo MQTT IN: Conecta el bróker MQTT y subscribe los mensajes del topic /dato

Figura 44. Nodo MQTT en Node-Red
Fuente: Autor

***Nodo Function*:** Realiza la división de los datos mediante un script, ya que desde el MQTT llegan en una cadena de texto separados con signo de ",". El *Nodo Function* constara de siete salidas, que representan cada uno de los datos obtenido mediante MQTT.

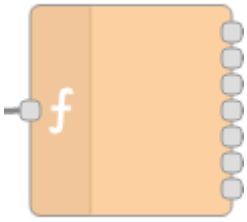


Figura 45. Nodo Funcion
Fuente: Autor

Nodo modbustcp-write o Modbus TCP cliente. Conecta al servidor Modbus TCP y escribe los mensajes a *coil* o *register*.

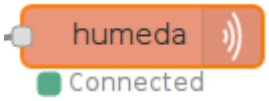


Figura 46. Nodo modbustcp escritura
Fuente: Autor

Nodo modbustcp-read: Conecta a un servidor Modbus TCP para leer valores de registro en un determinado *poll rate*.

Figura 47. Nodo modbustcp lectura
Fuente: Autor

3.6.6 CONFIGURACIÓN MODBUS SLAVE.

El Modbus Slave, su función es de escuchar y obtener los datos que son enviados por el Node Red.

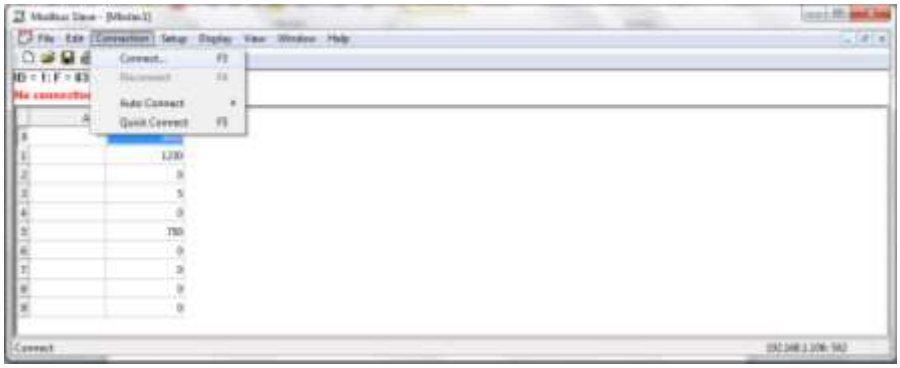


Figura 48. Pantalla de Conexión Modbus Slave
Fuente: Autor

Para esto se realiza una nueva conexión. Click en Connection →Connect , aparecerá la Figura N° 50 de la cual se escoge *conecction "Modbus TCP/IP*", además en el casillero "IP Address" se coloca la IP de la maquina en la cual está instalado el Software antes mencionado, y el puerto de comunicación por modbus 502.

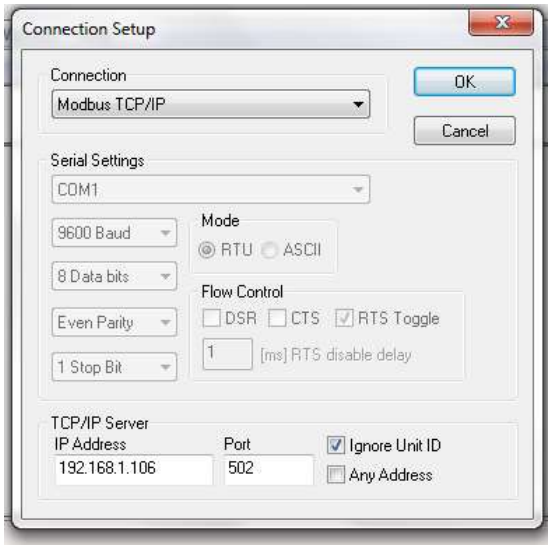


Figura 49. Configuración TCP/IP Modbus Slave
Fuente: Autor

3.6.7 CONFIGURACIÓN SCADA LAQUIS.

Para configurar SCADA Laquis, se debe fijar las variables, en el caso del presente trabajo serán: hum (Humedad), temp (Temperatura), velocidad (Velocidad del viento), lluvia, presión e Irradiación.

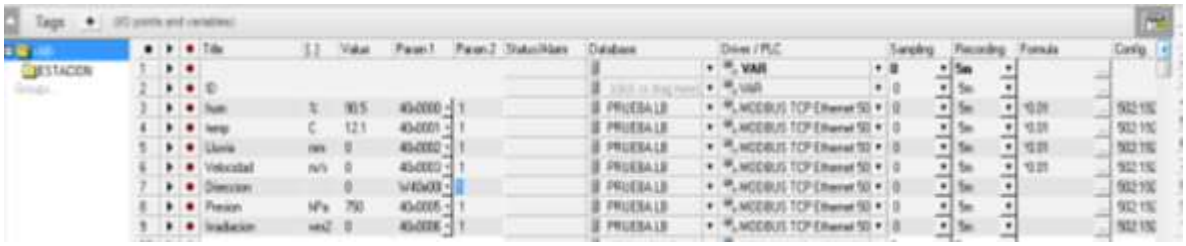


Figura 50. Parámetros a configurar SCADA Laquis

Fuente: Autor

La ventana de Configuración contiene las propiedades para definir que leerá y escribirá el SCADA. Algunos parámetros principales son: Nombre, titulo, Valor, Unidad, Base de datos, controlador PLC, Param1, Param 2. Uno de los parámetros importantes a configurar en la ventana *Driver PLC* es el protocolo de comunicación con el cual se conectará.

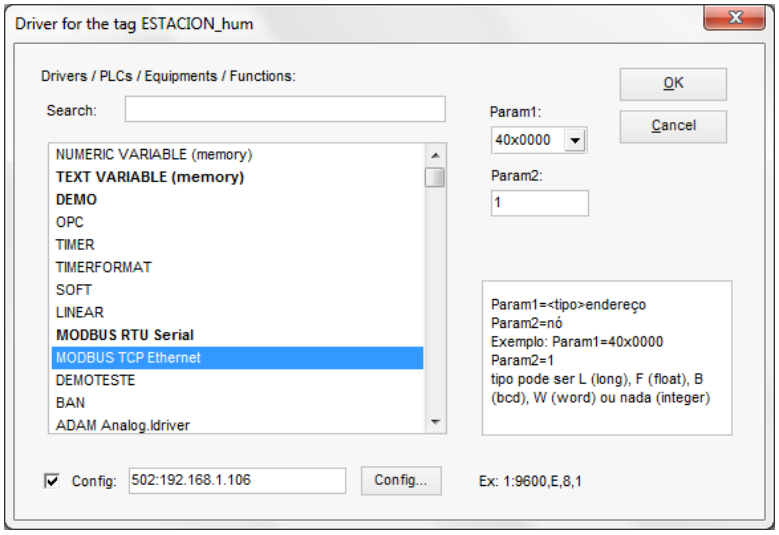


Figura 51. Configuración Driver MODBUS TCP/Ethernet en SCADA Laquis
Fuente: Autor

Para este proyecto se desarrollo en MODBUS TCP ETHERNET, debido a que se va a trabajar con una IP; colocando la direccion IP correspondiente y el puerto por cual comunica, en este caso el 502

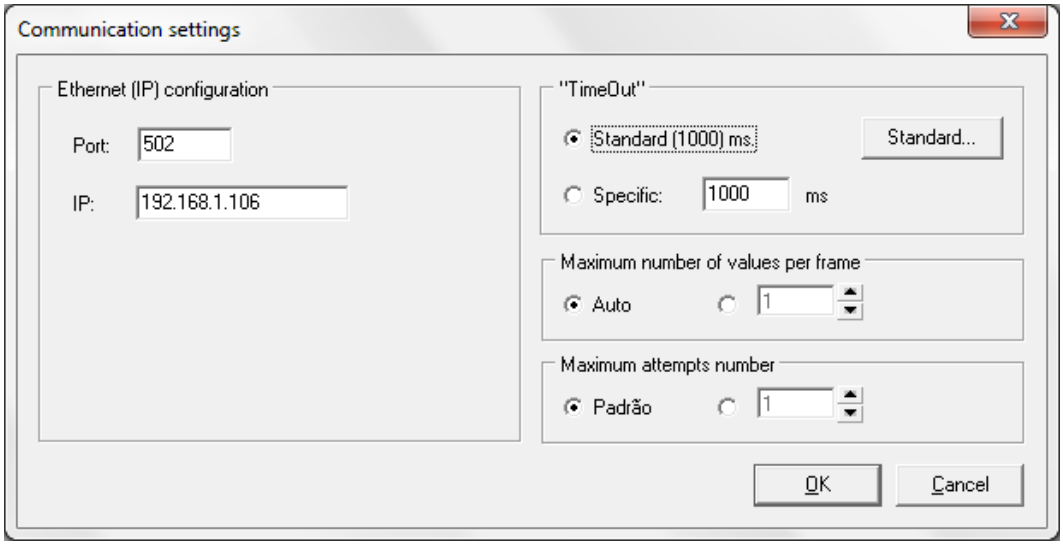


Figura 52. Configuración de la Comunicación TCP/Ethernet
Fuente: Autor

3.6.8 RESULTADOS

Una vez configurado el protocolo en El Raspberry Pi, en el MODBUS SLAVE y el SCADA LAQUIS, se pudo identificar que los datos recogidos por la estación meteorológicas en protocolo MQTT mostrados en la página WEB, son los mismos que se visualizaban en el SCADA LAQUIS y el MODBUS SLAVE, determinando el correcto funcionamiento de la transmisión de datos por el protocolo MODBUS TCP ETHERNET.

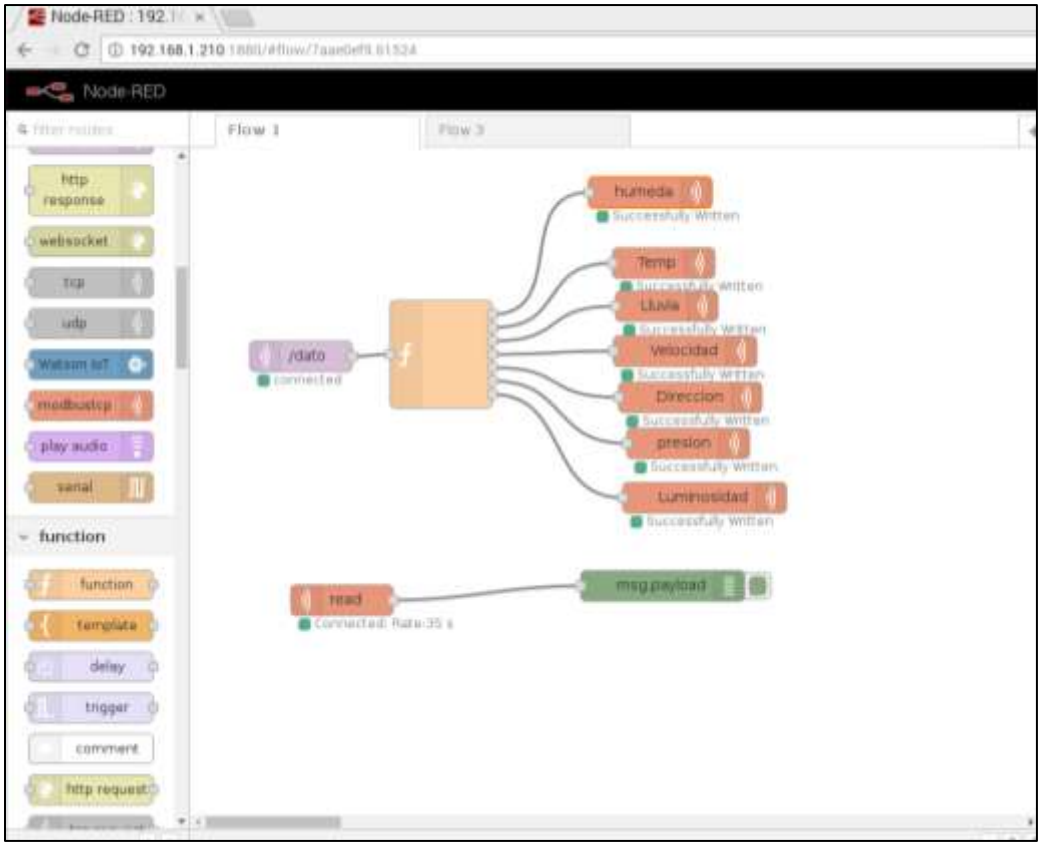


Figura 53. Node Red, Diagrama comunicación Modbus
Fuente: Autor

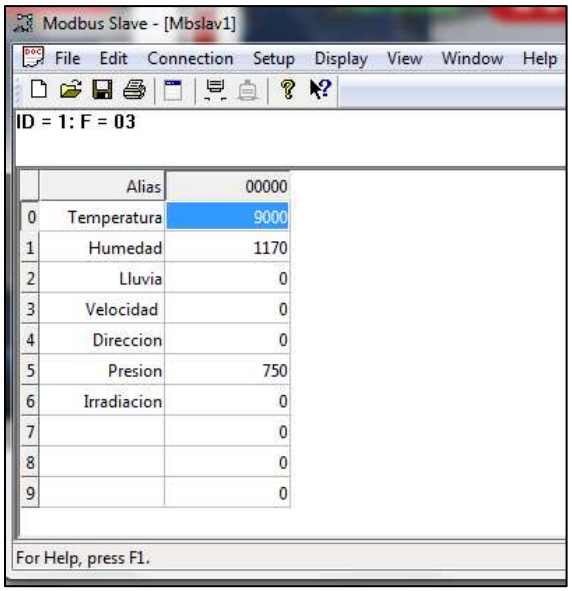


Figura 54. Adquisición de Datos con Modbus Slave
Fuente: Autor

Figura 55. Datos SCADA Laquis
Fuente: Autor

3.7 Aplicación móvil

La aplicación móvil de este proyecto se lo realizo en la herramienta *Android Studio* la cual proporciona una forma rápida de crear apps (aplicaciones) para toda clase de dispositivos Android.

Figura 56. Pantalla inicio Android Studio
Fuente: http://androidcel.com/tutorial/

La creación de la aplicación se lo realiza de una manera grafica en el programa *Android Studio*, definiendo cada una de los parametros a monitrear de la estación meteorologica, según se identifica en laFigura . N° 57

Figura 57. Configuración de la App Estación Meteorológica
Fuente: Autor

Una vez creada la parte gráfica de la aplicación, se escribe el código en JAVA para la ejecución de tareas de cada uno de los botones,asi como la programación de datos de salida de los parametros consultados.

Los botones de la aplicación direcionan laconsulta hacia la base de datos de la pagina WEB, la cual envia laconsulta hacia las ventanas de impresión de consulta de la aplicación.

En el anexo N° 4 , se muestra el código utilizado.

4. CONSTRUCCIÓN DE LA ESTACIÓN METEOROLÓGICA

4.1 Diagramas de construcción

El diagrama eléctrico ha sido diseñado en función de los niveles de voltajes que se utilizan los equipos principales de adquisición y procesamiento de datos, Arduino y Raspberry PI, así como los niveles de tensión de los sensores escogidos con sus necesidades electrónicas para adquisición de datos.

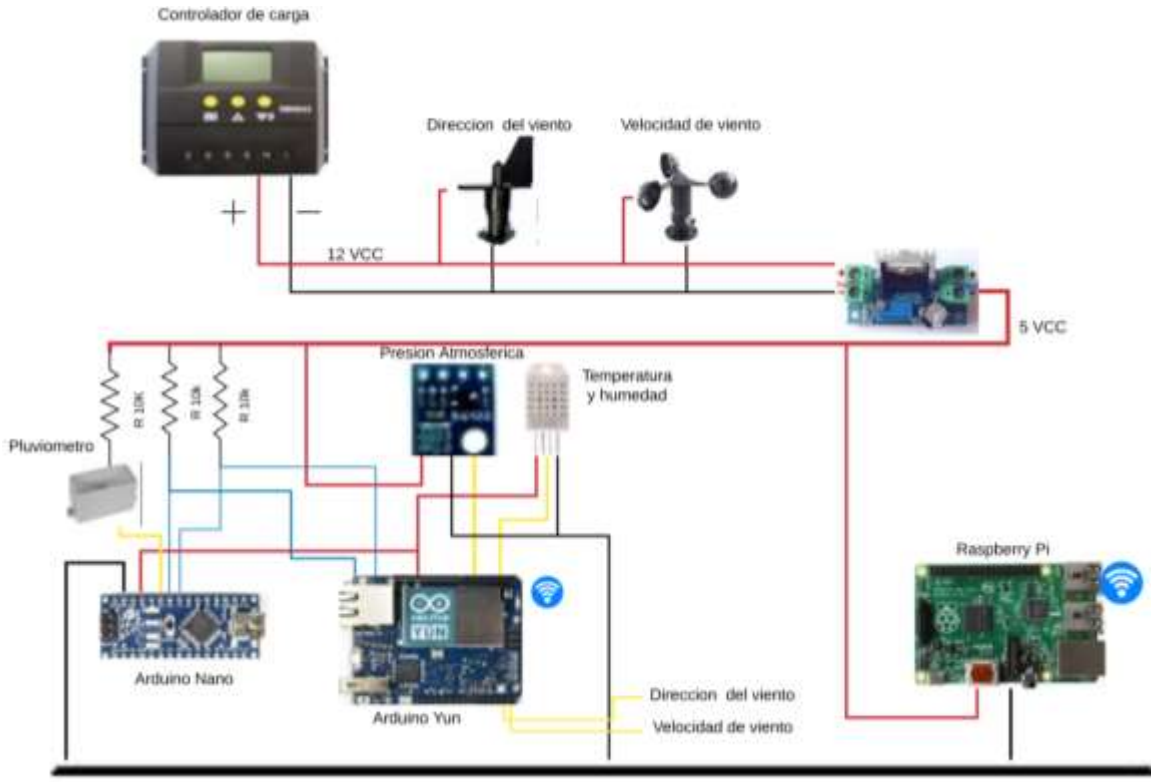


Figura 58. Diagrama eléctrico
Fuente: Autor

4.2 Listado de Materiales

Una vez determinada el desarrollo de software, se inicia el montaje de la estructura de hardware de la estación meteorológica, para la cual ubicamos la presente la lista de materiales a utilizar.

Descripción	Cantidad	Unidad
Estructura		
Soporte	1	U
Pantalla de Stevenson	1	U
Estructura Metálica	1	U
Elementos electrónicos		

Raspberry Pi, incluye memoria y ventilador	1	U
Arduino Yún	1	U
Arduino nano	1	U
Cable de alimentación	1	Lote
Tarjetas perforadas para ardunio 1	1	U
Conectores	5	U
Anemómetro	1	U
Veleta	1	U
Pluviómetro (balancín)	1	U
Sensor DHT22	1	U
Sensor BMP180	1	U
Sensor ML8511	1	U
Reloj RTC DS1307	1	U
Resistencias varias	5	U
Condensadores	1	U
Adaptador AC/DC	1	U
Controlador de carga	1	U

Tabla 4. Lista de materiales
Fuente: Autor

4.3 Presupuesto y Compra de Materiales

Una vez realizado el listado de materiales, hay que empezar con la construcción de la estación para lo cual se debe hacer un presupuesto para la compra de los distintos materiales a utilizar, es por ello que en la siguiente tabla se definirá los costes del montaje de la Estación Meteorológica

Cabe recalcar, que gran cantidad de materiales se importaron vía ALIEXPRES, debido a su ventaja económica en algunos casos y en otros porque no se podía encontrar en el medio local.

El presupuesto es el siguiente:

Descripción	**Cantidad**	**Unidad**	**Precio unitario**	**Total**
Estructura				
Soporte	1	U	25	25.00
Pantalla de Stevenson	1	U	10	10.00
Estructura Metálica	1	U	25	25.00
Subtotal 1				60.00
Elementos electrónicos				

Raspberry Pi, incluye memoria y ventilador	1	U	90	90.00
Arduino Yún	1	U	120	120.00
Arduino nano	1	U	10	10.00
Cable de alimentación	1	Lote	3	3.00
Tarjetas perforadas para ardunio 1	1	U	5	5.00
Conectores	5	U	0.45	2.25
Anemómetro	1	U	35	35.00
Veleta	1	U	35	35.00
Pluviómetro (balancín)	1	U	12	12.00
Sensor DHT22	1	U	12	12.00
Sensor BMP180	1	U	8	8.00
Sensor ML8511	1	U	15	15.00
Reloj RTC DS1307	1	U	7	7.00
Resistencias varias	5	U	0.1	0.50
Condensadores	1	U	0.1	0.10
Adaptador AC/DC	1	U	12	12.00
Controlador de carga	1	U	25	25.00
Subtotal 2				391.85
Personal Ingeniería				
Diseño Estructura	2	Hora	10	20.00
Diseño Eléctrico / Electrónico	40	Hora	10	400.00
Diseño Software	300	Hora	10	3,000.00
Subtotal 3				3,420.00

TOTAL	**3,871.85**

Tabla 5. Presupuesto
Fuente: Autor

4.4 Ensamblaje de la Estación Meteorológica

Conocido el presupuesto y adquiridos los elementos para la construcción de la estación, es necesario ensamblar todas sus elementos. En este apartado se explica como se desarrolló el diseño electrónico.

Como parte inicial se considera los diferentes voltajes que requieren los sensores para su adecuado funcionamiento, tanto sensores analógicos como digitales. Se considera que los sensores que trabajan a nivel de tensión 3.3 y 5 voltios se alimentan directamente del arduino Yún y su pin del sensor directamente a una de las entradas del mismo, pero en el caso de los

sensores que van a las entradas SDA y SCL (comunicación I2C), se debe tener muy en cuenta las resistencias Pull-up.

4.4.1 Consideraciones iniciales

Como la primera consideración son los voltajes con los cuales funciona cada una de los equipos de adquicición de datos y sensores, es una parte importante, en la siguiente tabla se va a describir los mismos.

Descripción	Voltaje de funcionamiento
Anemómetro	12 V
Veleta	12 V
Pluviómetro (balancín)	5V pull up
Sensor DHT22	3.3 V
Sensor BMP180	5V
Sensor ML8511	3.3V - 5V
Reloj RTC DS1307	5V
Nano Arduino	5V
Arduino	5V

Tabla 6. Tabla de voltajes
Fuente: Autor

4.4.2 Ensamblaje de la estructura

La estructura metálica se va a soportar los diferentes componentes, tiene una forma tubular vertical de los cuales se desprender unos soportes planos para cada uno de los componentes.

Por el hecho que hay vientos muy fuertes esta debe estar asentada en algún soporte que por lo cual se ha encontrado una solución en un soporte para parlantes de música marca “Boway”, para las pruebas y para ensambalje definitivo se lo realizará empotrando en un tubo metalico soportado por abrasaderas.

Figura 59. Estructura Metálica y soporte
Fuente: Autor

4.4.3 Sensores

4.4.3.1 Sensor de Velocidad de viento (Anemómetro)

El sensor de viento instalado en la estación meteorológica, es tipo analógico que da una señal de tensión dependiendo la velocidad del viento. El principio de funcionamiento de este sensor es un minigenerador de inducción de tensión que dependiendo de la velocidd induce tensió de salida de 0 a 5V. Se escogió el sensor de la Figura . 2 por su robustez metálica y su resistencia al agua, de características inoxidables

Para su conexión, simplemente se conecta el cable negro a tierra de la señal, el cable rojo a 12 VDC y para medir el voltaje analógico es el cable amarillo. La tensión oscilará entre 0,4 V (0 m / s de viento) hasta 5 V (velocidad de 30 m/s de viento).

Figura 60. Conexión Anemómetro- Arduino Yún
Fuente: Autor

4.4.3.2 Sensor de Dirección de viento (Veleta)

La veleta utilizada es de tipo analógica, su principio de funcionamiento se basa en un potenciómetro interno que cambia con la posición, consta de tres cables de conexiones el de color rojo es a +12V, el de color amarrillo tierra (-) y color verde es el cable de datos.

El cable de datos alimenta una señal analogica al arduino YÚN en el pin A2, el cual mediante rangos de tensión ubican las diferentes posisiones de la veleta determinada en la rosa de los vientos.

Figura 61. Conexión Veleta
Fuente: Autor

4.4.3.3 Ensamblaje de sensor DHT22 Temperatura y humedad

El sensor DHT 22 es alimentado por una tensión de 5VCC ; la señal del sensor ingresa por la entrada digital D4 del Arduino YÚN , mediente la cual lleva tanto la señal de temperatura como de humedad a la vez, la misma es manejada por la librería del arduino para separarla y determinar su medicicón individual.

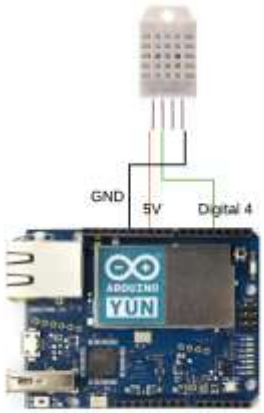


Figura 62. DHT22 conexión Arduino Yun
Fuente: Autor

El sensor DHT 22, este debe estar protegido contra la radiación del sol directa, la lluvia y otros fenómenos naturales; para que no se alteren las mediciones, la mejor opción encontrada en el mercado es la de utilizar una Pantalla de Stevenson, esta es de material plástico y muy resistente.

Figura 63. Pantalla Stevenson
Referencia: Autor

Dentro de esta se ha adaptado un pequeño pedazo de una caja plástica para que sostenga el sensor y no interfiera ningún factor para la toma de datos del mismo.

Figura 64. Caja contiene DH22
Fuente: Autor

4.4.3.4 Sensor de lluvia (pluviómetro).

El sensor de lluvia utilizado es un tipo balancín que al acumular en su estructura lluvia, cada 0,25 mm de lluvia da un pulso y mediante un contador va acumulando la cantidad de pluviosidad, este contador se reiniciará. A las 23h59h40, este se hace en base de la programación del arduino nano

Este sensor ha sido conectado a la entrada digital D2 de un arduino NANO, ya que el principo de finciameniento de conteo es a travez de interrupciones, y en caso de tener

conectado al arduino principal YÚN , las interrupciones altera el ciclo del programa, debido a la preferencia de las interrupciones, produciendo un mal funcionamiento.

En este mismo sensor, es necesario la incorporación de un reloj, debido a que cada 24 horas debe reiniciar su conteo, por la necesidad de segmentar la cantidad de lluvia recogida por día.

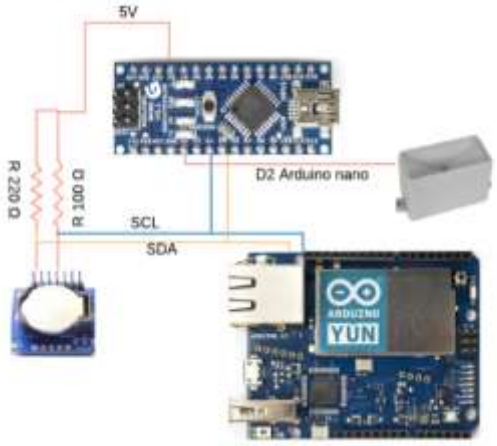


Figura 65. Conexión pluviómetro
Fuente: Autor

4.4.3.5 Sensor BMP180 presión atmosférica,

Este sensor es alimentado por 5VCC, y para la lectura de datos se lo realiza medinte comunicación con el arduino YÚN mediante el bus de datos I2C; su conección es entre los pines SCL y SDA del sensor a los mismos del arduino YÚN

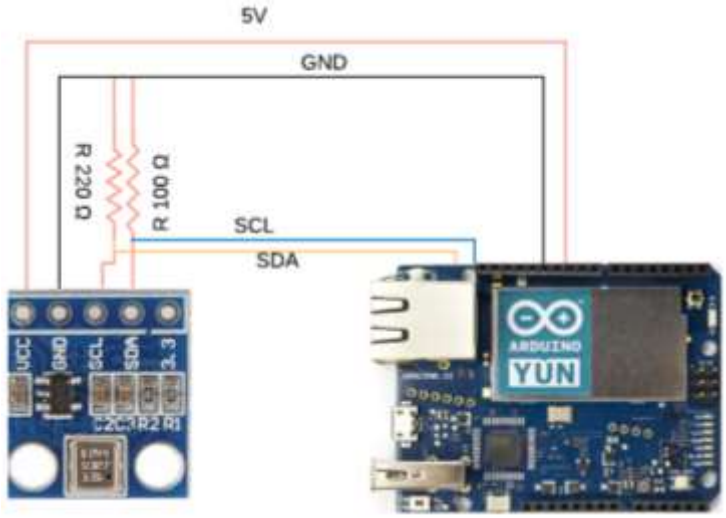


Figura 66. Conexión BMP180
Fuente: Autor

4.4.3.6 Sensor UV

Este sensor es conectado a un nivel de tensión de 3.3 V desde el arduino YÚN , y la salida del sensor UV se conecta a la entrada analogica A4, el cual ingresa un valor entre 1V hasta 2.5 voltios como maximo, determinado la minima y maxima radiación UV.

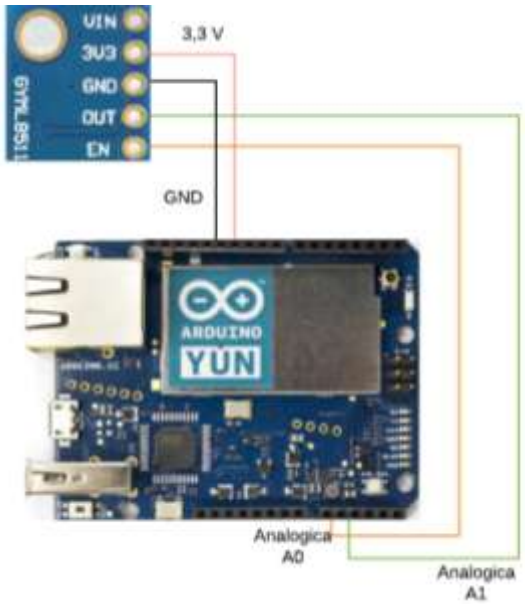


Figura 67. Conexión de Sensor UV
Fuente: Autor

4.5 Calibración de Sensores

Con el fin de realizar el ajuste de las variables físicas medidas por los diferentes sensores, se realizan pruebas de verificación de que las medidas sean coherentes mediante valores estandar a recibir con equipo patrón.

4.5.1 Calibración del Sensor de temperatura y humedad

El sensor de humedad y temperatura DH22, viene calibrado de fábrica el sensor, es decir envía directamente la medición de las dos variables al arduino para procesamiento de su señal. Lo que se realizo fue una comprobación de que la temperatura y la humedad este de acuerdo con los valores promedios de estaciones meterologicas cercanas encontradas en la WEB

Se determina que la medición que esta de acorde con lo requerido y los parametros dentro de los normales

En la siguiente Figura .65 se observa la medicicón de un día cualquiera de la estación diseñada funcionando, comparando con los datos de la Figura .66 tomados de la estación meteorologica “Sacay” Wunderground.

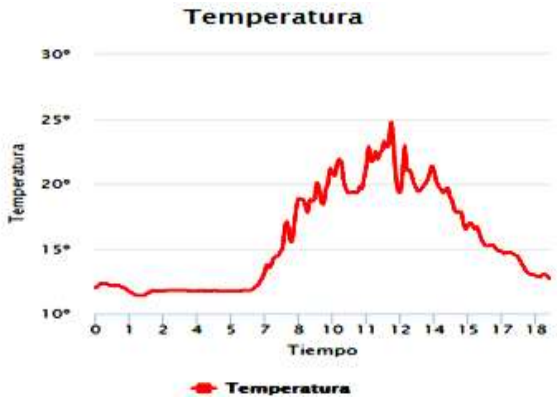


Figura 68. Temperatura 12 de enero
Fuente: Autor

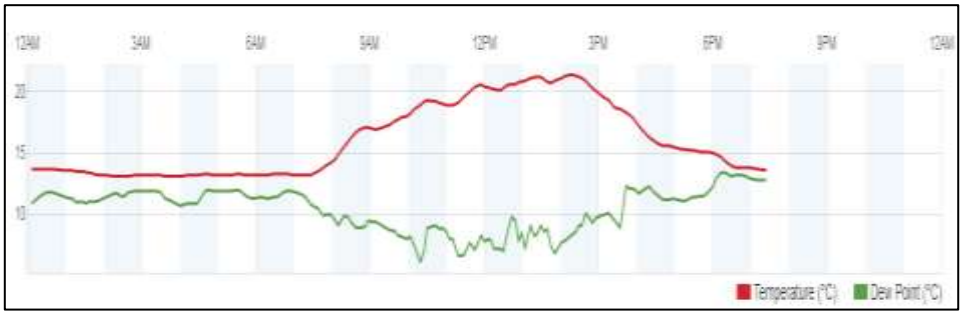


Figura 69. Temperatura Wunderground Estacion Sacay
Fuente: https://www.wunderground.com/weather/ec/sacay/-2.8999591%2C-79.03387790000001

4.5.2 Sensor de viento (Anemómetro)

Para la calibracion del sensor de viento se lo realizó en base de la recomendación del fabricante del sensor , es decir el arranque de la velocidad desde 0.1 voltios y la velocidad maxima 30 m/s a 5 voltios en el sensor. La señal que entra es de tipo analógica determinandose la siguiente curva Tensión vs Velocidad.

Voltajes V	Velocidad m/s
0,1	0,61
0,2	1,22
0,3	1,84
0,4	2,45
0,5	3,06
0,6	3,67
0,7	4,29
0,8	4,9
0,9	5,51
1	6,12
1,1	6,73
1,2	7,35

1,3	7,96
1,4	8,57
1,5	9,18
1,6	9,8
1,7	10,41
1,8	11,02
1,9	11,63
2	12,24
2,1	12,86
2,2	13,47
2,3	14,08
2,4	14,69
2,5	15,31
2,6	15,92
2,7	16,53
2,8	17,14
2,9	17,76
3	18,37
3,1	18,98
3,2	19,59
3,3	20,2
3,4	20,82
3,5	21,43
3,6	22,04
3,7	22,65
3,8	23,27
3,9	23,88
4	24,49
4,1	25,1
4,2	25,71
4,3	26,33
4,4	26,94
4,5	27,55
4,6	28,16
4,7	28,78
4,8	29,39
5	30

Tabla 7. Voltaje vs velocidad Anemómetro
Fuente: Autor

Además cabe indicar que el sensor viene con calibración de fabrica.

4.5.3 Sensor dirección de VIENTO (Veleta).

El sensor de dirección del viento se calibro según los niveles de transión repartidos en los 5 voltios. Para lo cual se toma un punto que se denomina punto cero que indica el norte.

Con la ayuda de una brujula, se fue identificando el rango de voltaje de cada pocición de la veleta para su programación, de lo que se obtuvo la siguiente tabla:

Rango de Voltaje	Direccion
0.1V a 0.5 V	NORTE
0.6V a 1.1 V	NE
1.2V a 1.7 V	ESTE
1.8V a 2.3 V	SE
2.4V a 2.9 V	SUR
2.9V a 3.5 V	SO
3.5V a 4.1 V	OESTE
4.1V a 4.7 V	NO

Tabla 8. Datos calibración veleta
Fuente: Autor

4.5.4 Sensor de lluvia (pluviómetro)

La calibración del sensor de lluvia se realizó según la recomendación del fabricante del sensor, es decir 0,25 mm de lluvia por cada pulso. Ademas se realizó la comprobación con mediciones mediante un gotero de precición para comprobar que el pulso este en el valor recomendado por el fabricante.

Además, como la necesidad estadistica de la lluvia es HORA-HORA y Diaria, se incrporó un reloj, el cual permite realizar el reseteo de la cuenta de lluvia a las 24:00 de cada día.

4.5.5 Sensor de presión BMP180

El sensor de presión **BMP180** viene calibrado de fabrica, solamente se reliza la comprobación de que la presión marque de acuerdo con los niveles de presión a la altura de la ciudad de cuenca, es decir 2500 msnm, ubicando una preción de 750 MMHG.

4.5.6 Sensor De Radiación Solar

El sensor de radiacion solar se lo calibra dependiendo del valor de voltaje de alimentación del sensor, en esta caso el volatje de entrada es de 3.3, por lo cual la funcion mathfloat, antes mencionada, realiza la operación para obtener los valores de radiacion solar.

La funcion es :

float mapfloat(float x, float in_min, float in_max, float out_min, float out_max)

```
{
 return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Mediante esta funcion hemos calculado la siguiente tabla de valore en excel.

voltage (x)	in_min	in_max	out_min	out_max	Uv
0	0.9	2.5	0	15	0
1	0.9	2.5	0	15	0
1.1	0.9	2.5	0	15	1
1.2	0.9	2.5	0	15	2
1.3	0.9	2.5	0	15	3
1.4	0.9	2.5	0	15	3
1.5	0.9	2.5	0	15	4
1.6	0.9	2.5	0	15	5
1.7	0.9	2.5	0	15	6
1.8	0.9	2.5	0	15	7
1.9	0.9	2.5	0	15	8
2	0.9	2.5	0	15	8
2.1	0.9	2.5	0	15	9
2.2	0.9	2.5	0	15	10
2.3	0.9	2.5	0	15	11
2.4	0.9	2.5	0	15	12
2.5	0.9	2.5	0	15	13

Tabla 9. Valores para sensor uv
Fuente: Autor

4.6 Pruebas de Campo.

Las pruebas de campo tiene como objetivo hacer el seguimiento del trabajo de la estación meteorologica, verficar su funcionamiento adecuado y confiabilidad. Además solventar ciertos imprevistos no contemplados en el diseño.

4.6.1 Problemas presentados:

Dentro de los problemas presentados durante las pruebas de campo se describen las siguientes:

4.6.1.1 Perdida de alimentación de energía.

Para las pruebas de la estación se realizó con energía de la fuente de la empresa de distribución, ya qe la misma brinda mayor garantia.

Cuando se tenia una suspensión de energía se perdia el registro de datos de la estacion meteorologica, y al volver la energía de una forma automática volvia a recolectar datos de la misma, esto en caso del arduino y sus sensores asocicados.

Para el caso del raspberry se tiene la dificultad de arrancar los procesos que envían al SCADA como es el NODE RED, corrigiendo con un scrip en el raspberry. Para el resto de señales los procesos estan en el arranque teniendo toma de datos inmediatos.

Además se probo con un sistema de respaldo de batería y panel solar, la cual efectivamente tiene una batería de respaldo de 7 A/H, que por su tamaño y costo se probo con esta, recomendando colocar una mas en paralelo para llegar a 14 a/h que garantizaría el funcionamiento con 16 horas de autonomía.

4.6.1.2 Errores de medición en los sensores

Durante las pruebas se ubicaron los siguiente errores de medición:

Sensor de velocidad: Este sensor inicalmente falto el valor de escala X6 que indicaba el fabricante que se debia aplicar al nivel de voltaje entregado por el sensor para obtener la velocidad del viento.

Sensor de dirección del viento: Este sensor entrega información imprecisa en caso de que no se ubique para la dirección norte determinda en la calibración, o en caso que este algunos grados girada, por lo que para la instalación se deberá tomar en cuenta claramente el norte con la marca dejada en la estación meteorologica.

Sensor de temperatura y Humedad:El sensor de temperatura y Humedad, inicialmente se detectaron picos de temperatura altos,por lo que se optó por colocar una resitsencia de 10.000 ohm entre la alimentación de 5V y el cablede datos.

Sensor de presión:

El sensor de presión varia en función con la temperatura, por lo que inicialmente se probaron en diferentes ubicaciones sin tener incidencia mayor en la medición.

Otro factor que determina la presentación de datos es que se puede presentar la presión relativa o la presión absoluta. La presión absoluta se determina de acuerdo a lo indicado en [34] la presion que se mide tomando como origen, cero de presión, esto corresponde al vacío absoluto, y la relativa se mide tomando como origen la presión atmosférica a nivel del mar

Sensor de radiación solar UV

El sensor de radiación solar UV, presenta incombenintes en la cubierta protectora de vidrio colocada para aislar de la lluvia y polvo. Este cobertor influye disminuyendo la radiación

UV comprado con una prueba de radiación directa, por lo que se realiza la corrección correspondiente en pruebas.

En el siguiente grafico se puede observar mediciones seguidas cada 5 minutos aproximadamente, que con protector da medición UV 5 y sin protector da medición UV 6. De esto se determina la corrección en la formula de calibración utilizada en la calibración.

Datos meteorológicos — Generar reporte

Fecha	Hora	Temperatura	Humedad	Cantidad de lluvia	Velocidad del viento	Presión	Irradiación
13/01/2018	12:03:34	25,4	39,7	0,25	1,29	749	5
13/01/2018	11:57:50	25,3	47,8	0,25	0,72	749	6
13/01/2018	11:52:05	23,6	47,8	0,25	1,15	749	6
13/01/2018	11:46:20	22,2	53,4	0,25	0,66	749	5
13/01/2018	11:40:36	22,3	46,3	0,25	0	749	5
13/01/2018	11:34:51	23,5	47,3	0,25	1,64	749	6
13/01/2018	11:30:24	24	44,3	0,25	1,26	749	3
13/01/2018	11:24:39	23,7	53,2	0,25	1	749	3
13/01/2018	11:18:55	22,8	53,7	0,25	0,97	750	3
13/01/2018	11:13:11	21,4	53,3	0,25	0	750	3

Figura 70. Tabla de datos Web
Fuente: Autor

Sensor de cantidad de Lluvia:

El sensor de lluvia, su principio de funcionamiento es tipo pulsante que da 0.25 mm de lluvia cada llenado del cucharón ocacionando un pulso, contando la lluvia diaria a trvez de su arduino auxiliar nado arduino, qien reporta al arduino principla YÚN . Se tuvo el incombeniente de que es necesario que el arduino nano reinicie la señal al cumplir las 24 horas del día, para lo cual se acoplo un reloj que da la mando de autoreseteo al nano ardunino y pueda iniciar una nueva cuenta.

Como incobeninte se obtuvo que la pila del reloj se desgastaba y perdia la hora en un lapso de 3 meses, Se debe seguir monitoreando un vez que se conectó el reloj a la alimentación del arduno a 3.3 vcc

4.6.2 Pruebas de la estación

En la tabla N° 11, se presenta el informe de consulta a la estación meteorologica, sacado un reporte a travez de la pagina www.emeteo.dnns.net

Fecha	Hora	Humedad	Temperatura	Cantidad de lluvia	Velocidad del viento	Dirección	Presión	Irradiación
14/01/2018	11:05:56	51,4	23,8	0	0	'NO'	749	4
14/01/2018	11:11:41	48,9	24,2	0	0,4	'NO'	749	4
14/01/2018	11:17:26	48,9	24	0	0	'NO'	749	1
14/01/2018	11:26:27	55,7	21,7	0	0,63	'NORTE'	749	5
14/01/2018	11:32:12	64,2	22,8	0	1,52	'NORTE'	749	6
14/01/2018	11:37:57	47,8	23,9	0	1,49	'NORTE'	749	6
14/01/2018	11:43:42	46,6	24,7	0	1,15	'NO'	749	7
14/01/2018	11:49:27	42,8	24,7	0	0	'NORTE'	749	6
14/01/2018	11:55:12	46,2	23,9	0	1,64	'NE'	748	8
14/01/2018	12:00:57	47,6	24	0	0,57	'NORTE'	748	8
14/01/2018	12:06:42	43,4	25,6	0	0	'NO'	748	7
14/01/2018	12:12:26	41,5	26	0	0,72	'NORTE'	748	8
14/01/2018	12:18:11	41,7	26,1	0	1,87	'NO'	748	9
14/01/2018	12:23:56	43,7	25	0	0,4	'NO'	748	10
14/01/2018	12:29:41	40,7	25,3	0	1,44	'NORTE'	748	6
14/01/2018	12:35:26	44,3	25,8	0	1,35	'NO'	748	1
14/01/2018	12:41:11	52,2	22,4	0	3,51	'NO'	748	1

Tabla 10. Reporte Estación Meteorológica
Fuente: Autor

Además en la tabla N° 12 se presenta los datos obtenidos a travez del SCADA Laquis descargado como DEMO, los cuales son coincidentes con los datos de la estación. Cabe Recalcar que el SCADA Laquis (version DEMO) se utilizó para demostrar la conectividad del protocolo MODBUS.

Time	Group name	Lluvia	Velocidad	Presion	HUM	TEMP	DIRECCION	IRRADIACION
11:20:45	ESTACION	0	0	749	48.9	24	0	1
11:25:45	ESTACION	0	0	749	48.9	24	0	1
11:30:45	ESTACION	0	0.63	749	55.7	21.7	0	5
11:51:09	ESTACION	0	0	749	42.8	24.7	0	8
11:56:09	ESTACION	0	1.64	748	46.2	23.9	0	8
12:01:09	ESTACION	0	0.56	748	47.6	24	0	8
12:06:09	ESTACION	0	0.56	748	47.6	24	0	8
12:11:09	ESTACION	0	0	748	43.4	25.6	0	7
12:16:09	ESTACION	0	0.72	748	41.5	26	0	8
12:21:09	ESTACION	0	1.87	748	41.7	26.1	0	9
12:26:09	ESTACION	0	0.4	748	43.7	25	0	10
12:31:09	ESTACION	0	1.44	748	40.7	25.3	0	6
12:36:09	ESTACION	0	1.35	748	44.3	25.8	0	1

115 registers.

Print | Combos...

Tabla 11. Reporte SCADA
Fuente: Autor

En la Figura 75, se puede observar la aplicación para ANDROID instalada en el celular, la cual hace consultas en Linea a la estación, siendo coherentes los mismos con los datos de la pagina WEB y del SCADA.

Figura 71. Datos aplicación movil
Fuente: Autor

4.6.3 Reportes

Los reportes de la estación se generan a travez de la misma pagina WEB, colocando las fechas y consulta a realizar. Además se puede descargar del Raspberry Pi los archivos que se van almacenando en la memoria SD del dispositivo. Se forma un archivo con un peso de 64,4 KB por día, con un rango de grabación cada 5 minutos de los datos de la estación, pudiendo ser mayor o menor dendiendo de la necesidad. Para las pruebas se lo ha colocado en 5 minutos.

En la tabla. 11 se puede visualizar el reporte

cdatos ▲ 1 Clave primaria	fecha fecha lectura	hora hora de lectura	humedad cantidad humedad 0 a 100 se mide en porcentaje p(%...	temperatura cantidad temperatura 0 a 100 se mide grados centig...	cantidadlluvia cantidad 0 a >100 se mide en milimetros mm	velocidadviento cantidad 0 a >100 se mide kilometros por hora k...	direccion direccion del viento NORTE, SUR, ESTE, OESTE	presion valor 0 a >100 se mide libra-fuerza por pulgada...
1651	2018-01-14	17:39:20	98.6	13.7	2	0.89	'NE'	747
1652	2018-01-14	17:45:05	98.8	13.5	2	0.66	'NE'	747
1653	2018-01-14	17:50:49	98.7	13.3	2	0	'NE'	747
1654	2018-01-14	17:56:34	99.5	13.2	2	0	'NORTE'	747
1655	2018-01-14	18:02:19	99.9	13.2	2	0	'NORTE'	747
1656	2018-01-14	18:08:03	99.4	13.1	2	0	'NORTE'	747
1657	2018-01-14	18:13:48	99.4	13	2	0	'NORTE'	748
1658	2018-01-14	18:19:32	99.9	12.9	2	0	'NORTE'	748
1659	2018-01-14	18:25:17	99.8	12.8	2	0	'NORTE'	748
1660	2018-01-14	18:31:01	99.5	12.8	2.25	0.51	'NORTE'	748
1661	2018-01-14	18:36:46	98.9	12.8	2.25	0	'NORTE'	748
1662	2018-01-14	18:42:30	99.9	12.8	2.25	0	'NE'	748
1663	2018-01-14	18:48:15	99.9	12.7	2.25	0	'NORTE'	748
1664	2018-01-14	18:54:00	99.9	12.6	2.25	0.43	'NORTE'	748
1665	2018-01-14	18:59:44	99.9	12.6	2.25	0	'NORTE'	748

Figura 72. Base de Datos PHP Admin
Fuente: Autor

4.7 Resultados

Para observar los resultados obtenidos de los diferentes sensores de la Estación Meteorológica, se realiza una comparación con la estación Meteorológica más cerca al lugar de la instalación, siendo esta la Estación Meteorológica Sacay, la cual presenta los datos mediante la página https://www.wunderground.com/.

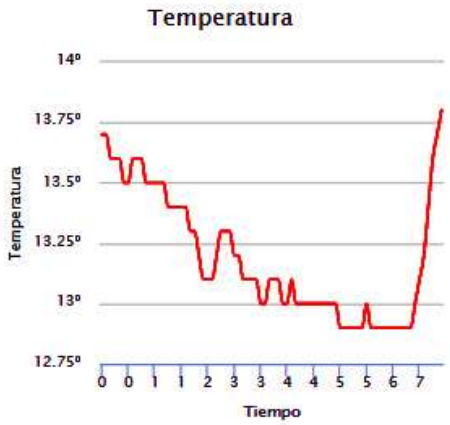


Figura 73. Grafica individual Temperatura

Fuente: Autor

Grafica General

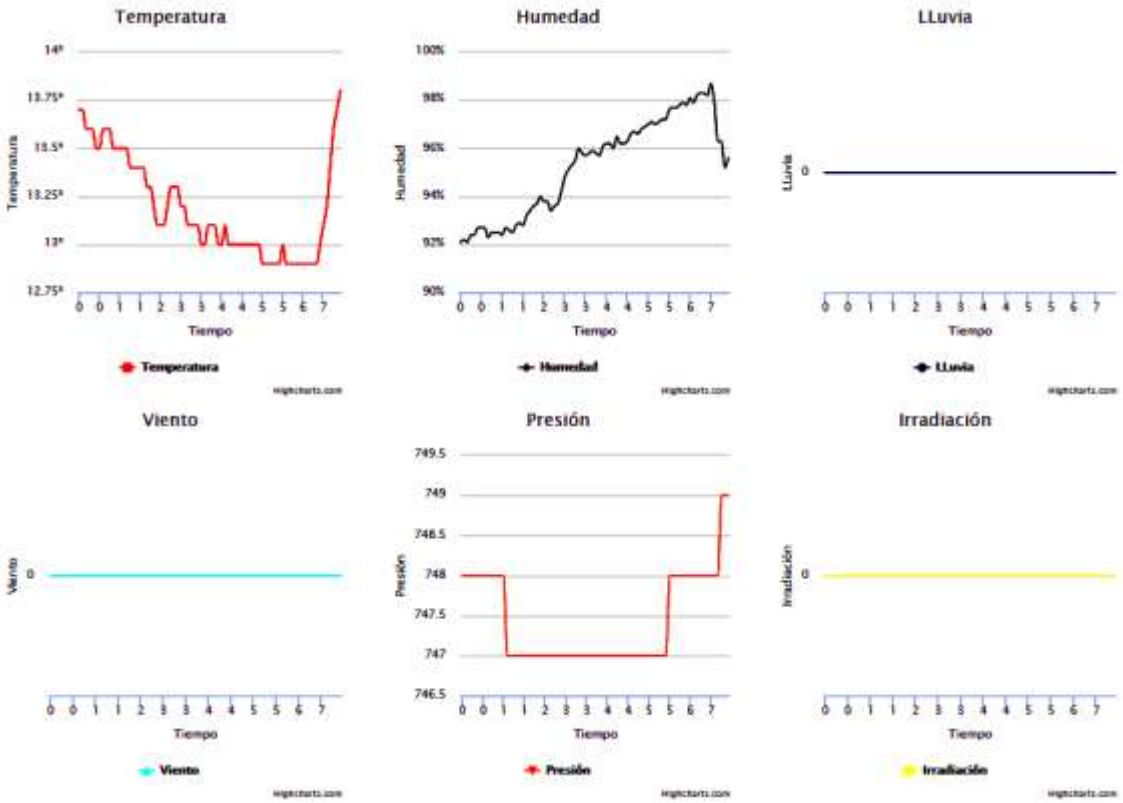


Figura 74. Grafica General de los sensores Climáticos
Fuente: Autor

Datos meteorológicos

Generar reporte

Fecha	Hora	Temperatura	Humedad	Cantidad de lluvia	Velocidad del viento	Presión	Irradiación
26/01/2018	12:07:30	20,5	54,6	0	1,35	749	7
26/01/2018	12:05:17	20,1	57,4	0	1,35	749	5
26/01/2018	11:59:32	19,7	59,1	0	0,43	749	3
26/01/2018	11:53:47	19,4	62,2	0	1,12	749	2
26/01/2018	11:45:35	19,5	59,5	0	1,12	749	1
26/01/2018	11:39:50	21,8	55,8	0	1,58	749	1
26/01/2018	10:30:01	21,4	55,4	0	1,35	750	3
26/01/2018	10:24:15	20,4	63,3	0	1,03	750	3
26/01/2018	10:18:30	20,2	60,5	0	0,8	750	3
26/01/2018	10:12:45	20	62,1	0	0	750	2

Figura 75. Valores recogidos por los sensores
Fuente: Autor

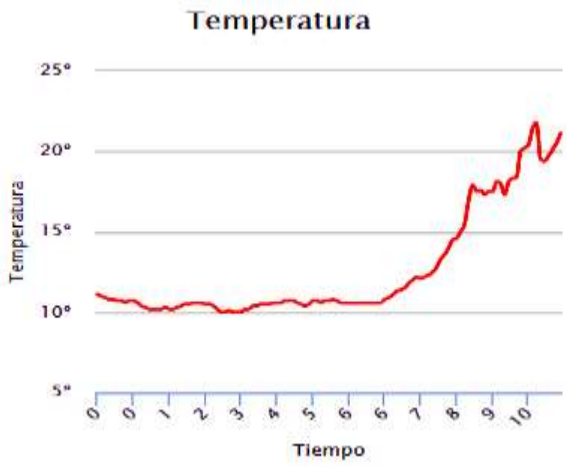


Figura 76. Valor de temperatura.
Fuente: Autor

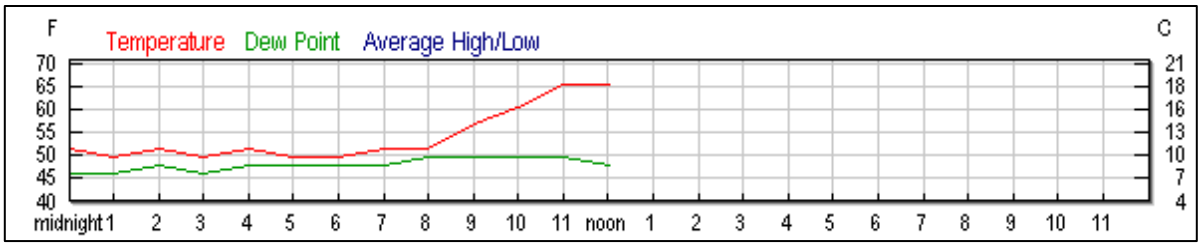


Figura 77. Valores temperatura de Wunderground Estación Sacay
Fuente: https://www.wunderground.com/hourly/ec/sacay/-2.8999591%2C-79.03387790000001?cm_ven=localwx_hour

En la figura 80 se visualiza los valores de temperatura tomados por el proyecto de la Estación, si se la compara con la figura 81, mediante las graficas se puede apreciar que los valores estas por el mismo rango, esta fue tomada el dia 26 de enro de 2018.

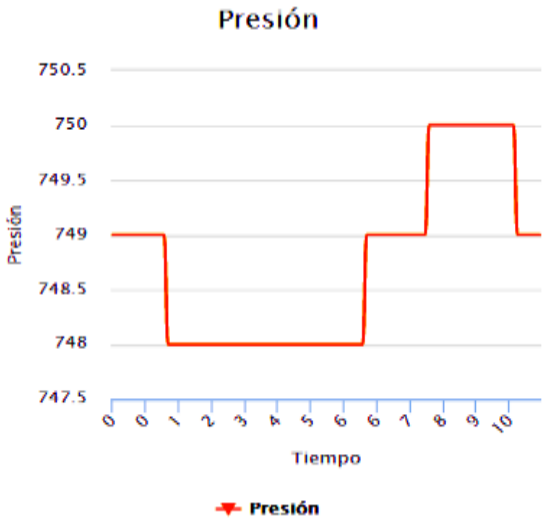


Figura 78. Valor de presión atmosférica
Fuente: Autor

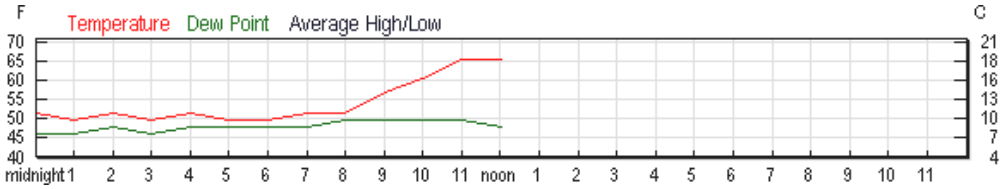


Figura 79. Valores Presión atmosférica de Wunderground Estación Sacay
Fuente: : https://www.wunderground.com/hourly/ec/sacay/-2.8999591%2C-79.03387790000001?cm_ven=localwx_hour

5. CONCLUSIONES

- El desarrollo de la estación meteorológica, en las condiciones del tema de tesis determinado en la denuncia, se han desarrollado íntegramente, es decir contiene el Diseño, desarrollo e implementación de una estación meteorológica basada en una red jerárquica de sensores, software libre y sistemas embebidos utilizando comunicación MQTT y MODBUS.
- Se ha implementado una red de sensores en la estacion meteorológica y las variables medidas son transmitidas en tiempo real hacia un servidor, en donde son almacenados en una base de datos, los que luego pueden ser visualizados y analizados.
- La estación meteorológica ha implementado el uso del protocolo MQTT para la adquisición de datos de los distintos sensores, para luego ser visualizados mediante una interfaz web y aplicación móvil basada en android. Los resultados obtenidos nos indica que el protocolo MQTT tiene la capacidad de transferir datos exactos y en tiempo real, de una forma segura en un puerto especifico.
- MQTT debido al tiempo que está en el mercado, este tiene una alta adaptación y la disponibilidad de recursos, además es dirigido a redes de bajo ancho de banda, alta latencia y debido a su método de publicación/suscripción este es extremadamente ligero, también trata de asegurar la confiabilidad y brinda un cierto grado de seguridad de trasmisión con calidad del servicio, por lo cual se ha adaptado muy bien en este proyecto ya que se maneja poca cantidad de datos.
- La seguridad que brinda MQTT depende del puerto que se utiliza, siendo conveniente tener un nivel medio-bajo que permita un flujo de información sin demasiadas restricciones: para este caso se utilizó el puerto 1883. Esta consideración se la hizo porque la información no es de prioridad alta para el usuario final.
- El control de las medidas climatológicas básicas es la función principal del sistema y considerando que estas cambian lentamente. Las muestras de Temperatura, presión, radiación, Dirección del Viento y humedad se las tomaran cada cinco minutos, la medición de la velocidad de viento se las tomo cada 7 segundos haciendo un promedio y luego mostrándola con los demás datos. Dando como resultado una lectura de datos válida para realizar las distintas predicciones y la respectiva toma de decisiones.
- La frecuencia de muestreo nos indica el número de muestras que se tomarán a cabo de un intervalo de tiempo, en la estación meteorológica se ha estimado un intervalo de 5 minutos, considerando que pueden ocurrir cambios climáticos drásticos en periodos de tiempo muy pequeños. Con este tiempo y tomándolos de forma continua se asegura tener datos mucho más precisos.

- El simulador Modbus TCP, permite la comunicación entre el Modbus Slave y SCADA LAquis teniendo como intermediario al Node Red , consiguiendo captar los valores con las direcciones de los registros asignados por el mismo, con esto se consigue que cualquier dispositivo industrial con protocolo MODBUS pueda comunicarse e informar sobre las variables meteorológicas.
- Las bondades del Raspberry Pi todavía son limitadas en cuanto a memoria RAM, y disco, teniendo limitaciones hasta 32 G, teniendo que buscar software liviano para las distintas aplicaciones. Así mismo es limitado su utilización ya que necesita un desarrollo de un buen sistema de ventilación para el procesador.

6. RECOMENDACIONES

- Reemplazar la Raspberry PI por Mini PC que tiene características con mucho mejores prestaciones manteniendo la portabilidad del equipo
- Utilizar sensores mucho más robustos para la adquisición de datos, principalmente en el tema de temperatura e irradiación solar.
- Dotar fuentes de refrigeración para el procesador del Raspberry PI adecuadas, debido a que no soporta muchos procesos abiertos a la vez.
- Desarrollar este tipo de proyectos entre carreras interdisciplinarias para que se tenga un mejor aporte en el área de la electrónica y diseño de circuitos.
- Realizar un seguimiento continuo al trabajo desarrollado, con el fin de poder ir aplicando mejoras a posibles problemas no detectados en el desarrollo.

7. REFERENCIAS

[1] Gerardo Echeto,Revista Electrónica de Estudios Telemáticos, PROTOCOLO IEC-104/VSAT APLICADO AL SEGUIMIENTO Y CONTROL DESUBESTACIONES ELÉCTRICAS, Volumen 8 Edición No 2, Año 2009

[2] Ministerio De Medio Ambiente Y Agua, Servicio Nacional De Meteorología E Hidrología www.senamhi.gob.pe/main_down.php?ub=mmt&id=cap2.

[3] Sistematización del Componente 2 del Proyecto PRAA, http://www.ambiente.gob.ec/wp-content/uploads/downloads/2014/07/Sistematizacio%CC%81n-regional-PRAA.pdf, 5 de agosto de 2013.

[4] M. Pérez, D. Martínez; http://bibdigital.epn.edu.ec/bitstream/15000/4331/1/CD-3940.pdf, "Diseño de un sistema de comunicación para la transmisión de datos en tiempo real entre tres estaciones meteorológicas ubicadas en el Volcán Antisana y el Instituto Nacional de Meteorología e Hidrología", en Quito, Escuela Politécnica Nacional, Ecuador, 2011.

[5] http://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport

[6] http://docs.oasis-open.org/MQTT/MQTT/v3.1.1/cos02/MQTT-v3.1.1-cos02.doc

[7] MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta.. OASISStandard. http:// docs.oasis-open.org/ MQTT/ MQTT/ v3.1.1/ os/ MQTT-v3.1.1-os.html,29 October 2014

[8] J. Rodríguez E; Https://Www.Dspace.Espol.Edu.Ec/Retrieve/97425/D-103513.Pdf, "Diseño De Una Arquitectura Genérica De Iot Aplicada A Casos De Emergencias Para Dispositivos Médicos Inalámbricos Implantados", en Guayaquil, Escuela Superior Politécnica Del Litoral, Ecuador, 2016.

[9] Simply Modbus http://www.simplymodbus.ca/TCP.htm, 2017

[10] Modbus Java, https://documents.mx/documents/modbus-java.html, Jul 09, 2015

[11] Ingeniería MCI Ltda.; Que es Arduino, http://arduino.cl/que-es-Arduino, 2014

[12] E. Crespo: Aprendiendo Arduino, https://aprendiendoarduino.wordpress.com/,2014

[13] ESPE, REVISTA Congreso de Ciencia y Tecnología Memorias Sesiones Técnicas;, http://ciencia.espe.edu.ec/wp-content/uploads/2016/06/Revista_08_06_2016_2-final3.pdf , junio 2016

[14] Xataka; Raspberry Pi; http://www.xataka.com/gadgets/altavoces/raspberry-pi, 2014

[15] Noticias Raspberry Pi; https://www.raspberryshop.es/,2014

[16] Alejandro Gálvez Morgado, Desarrollo E Implementación De una Estación Meteorológica mediante La Plataforma hardware/Software Libre Raspberry Pi, https: //es.scribd.com/document/316930926/Desarrollo-e-Implementacion-de-Una-Estacion-Meteorologica-Mediante-La-Plataforma-Hardware-software-Libre-Raspberry-Pi-Alejandro-Galvez-Morgado, Septiembre 2015.

[17] Jonathan Baquero, Brayan Castañeda, Dora Lilia, Dispositivo Para Monitorear En Tiempo Real Vía Web Variables Como Temperatura, Humedad Y Co2, http://repository.udistrital.edu.co/bitstream/11349/5437/1/PuentesRiveroJonathan2017.pdf,2017

[18] Desarrollo de una aplicación basada en RF mediante micro controladores programados en Python , http://upcommons.upc.edu/bitstream/handle/2117/77847/MemoriaRogerGustems.pdf?sequence=1

[19] Software Raspberry Pi 2014:https://www.raspberryshop.es/software-raspberry-pi.php

[20] Sensores DHT22 y DS18b20, http://visystem.ddns.net:7442/graficas-sensores-ds-dth22/,2016

[21] Diseño de un anemómetro basado en el efecto piezoresistivo, https://repositoriotec.tec.ac.cr/bitstream/handle/2238/740/Informe_Final.pdf?sequence=1&isAllowed=y,2008

[22] Fred V. Brock, Scott J. Richardson. Meteorological Measurement Systems. Oxford New York. Oxford University Press. Inc.

[23] Jorge Valdivia Ponce. Meteorología general. Lima Perú. UNMSM, 1977

[24] Kyle Brown, Distinguished Engineer, CTO for Cloud and EmergingTechnologies.IBM ibm.com/developerWorks/ssa/, 2014

[25] Steve. Understanding the MQTT Protocol, http://www.steves-internet-guide.com/MQTT-protocol-messages-overview/ June 13, 2017

[26] Messaging con AMQP, MQTT y STOMP, https://geeks.ms/jorge/2017/07/11/messaging-con-amqp-mqtt-y-stomp/, 11 de Julio de 2017.

[27] Leonard Barolli and Olivier Terzo. Complex, Intelligent, and Software Intensive Systems.Italia: Department of Information and Communication Engineering.

[28] Rokitoh, Instalación de servidor Apache Tomcat 7 en Debian, http://red-orbita.com/?p=5833, 18 febrero, 2013

[29] Wikipedia, Tomcat, https://es.wikipedia.org/wiki/Tomcat, 2 dic 2017

[30] Hector García Galán, ¿Por qué usar Tomcat sobre Servidores Cloud? , https://www.arsys.es/blog/programacion/tomcat-servidores-cloud/, 26 de enero de 2017

[31] libros web , Conectarse a la base de datos y ejecutar consultas,http://librosweb.es/libro/python/capitulo_12/conectarse_a_la_base_de_datos_y_ejecutar_consultas.html, 2016

[32] Archivos WAR, http://www.myjavazone.com/2012/07/archivos-war.html, Julio 2013

[33] Gastón, Mundo geek, http://mundogeek.net/archivos/2005/09/28/no-ip/, 2005

[34] Jesús Darío Rivera, Programación Visual con Node-Red, https://www.toptal.com/nodejs/programaci%C3%B3n-visual-con-node-red-conectando-el-internet-de-las-cosas-con-facilidad/es.

[35] modbus tolos, http://www.modbustools.com/modbus_slave.html,2017

[36] aucadenas, Presión Atmosférica, Absoluta y Relativa, https://kupdf.com/download/presion-relativa-absoluta-atmosferica_597cc129dc0d60db452bb180_pdf, Julio de 2017

8. ANEXOS

Anexos 1

Código Arduino YUN

```
#include <SPI.h>
#include <Bridge.h>
#include <Yún Client.h>
#include <IPStack.h>
#include <Countdown.h>
#define MQTTCLIENT_QOS2 1
#include <MQTTClient.h>
#include <Wire.h>
#include <DHT.h>
#include <Adafruit_BMP085.h>
////
int contador = 0;  //Variable para guardar la cuenta de pulsaciones
int estadoAnteriorBoton = 0; //Declaramos e inicializamos la variable
int valorBoton = 0;
////
char printbuf[100];
// DHT11 sensor pins
#define DHT_pin 4
#define DHTTYPE DHT22
DHT dht(DHT_pin,DHTTYPE);
Yún Client Yún Client;
///pluviometro
volatile int pluvioContador = 0;
//int contador = 0;
float canLluvia =0;
//variables anemometro
  int serial_in = 0;
  double x = 0;
```

```
 double y = 0;
 const int sensorPin = A0;
 float  sensorValue = 0;
 int sensorVoltage = 0;
    // Direccion Viento;
  char direccion[10]="";
//variables temperatura humedad
   float h,t =0;
// variable presion atmosferica
Adafruit_BMP085 bmp;
// variables rayos uv
int intensidadUv=0;
int uvSalida = A4; //Output del  sensor
//conexion Arduino Yún
IPStack ipstack(Yún Client);
MQTT::Client<IPStack, Countdown> client = MQTT::Client<IPStack, Countdown>(ipstack);
byte mac[] = {0xB4, 0x21, 0x8A, 0xF0, 0x4A, 0x3C}; // MAC  de su dispositivo
const char* topic = "/dato/Yún "; // nombre de topic
//--------------conectar----------------
void connect() {
    char hostname[] = "192.168.1.210"; // IP del servidor
    int port = 1883;
    sprintf(printbuf, "Conectando a %s:%d\n", hostname, port);
    Serial.print(printbuf);
    int rc = ipstack.connect(hostname, port);
    if (rc != 1) {
        sprintf(printbuf, "rc from TCP connect is %d\n", rc);
        //Serial.print(printbuf);
    }
    Serial.println("MQTT conectado");
    MQTTPacket_connectData data = MQTTPacket_connectData_initializer;
```

```
    data.MQTTVersion = 3;
    data.clientID.cstring = (char*) "arduino-Yún "; //nombre de en mosquitto
    rc = client.connect(data);
    if (rc != 0) {
        sprintf(printbuf, "rc from MQTT connect is %d\n", rc);
        //Serial.print(printbuf);
    }
}
void setup() {
    Wire.begin();
    Bridge.begin();
    Serial.begin(9600);
    dht.begin();
    bmp.begin();
    pinMode(uvSalida, INPUT);
    connect();
}
void loop(void) {
    ///humedad
  float h = dht.readHumidity(); //Se lee la humedad
  float t = dht.readTemperature(); //Se lee la temperatura
  long presion1 = bmp.readPressure(); //Se lee la presion atmosferica
  long presion = presion1/100;
  unsigned int lectura;
  Wire.beginTransmission(1);  // Enviamos a la dirección del esclavo 1
  Wire.write(1);              // Enviamos un 1
  Wire.endTransmission();     // Terminamos la transmision
  delayMicroseconds(100);       // Esperamos para poder hacer visibles los datos (posiblemente no haga falta en otras circunstancias)
  Wire.requestFrom(1, 1);     // Pedimos 1 bytes al esclavo 1
  while(Wire.available())     // Mientras tengamos datos en la entrada
```

```
 {
  lectura = Wire.read();    // Leemos el resultado y lo asignamos a la variable lectura
  if (contador != lectura)
    {
      contador = lectura;
      canLluvia=contador*0.25;
    }
   Serial.print("\"lluvia\":");
   Serial.println(canLluvia);
   delay(1000);
 }
 // sensor uv
 int uvLevel = promedioLecturaAnalog(uvSalida);
 float voltajeSalida = uvLevel*0.00488281;//(3.3 / 1024);
 int uvIntensidad = mapfloat(voltajeSalida, 0.9, 2.9, 0.0, 15.0);
 abs(uvIntensidad);
 ///Anemometro
 //float sensorValue = analogRead(sensorPin);
 float sensorValue = promedioLecturaAnalog(sensorPin);
 float  sensorVoltage = sensorValue *0.0048;//*5/1024;
 float velocidadViento= sensorVoltage;
//veleta
 float m= analogRead(A2);
 float volt= m*0.0048;//conversion de voltaje a 5v (5/1024);
     if (volt>=0.1&& volt<0.5){
          sprintf(direccion, "%s"," 'NORTE' ");
       }
       else if (volt>=0.6 && volt<1.1){
         sprintf(direccion, "%s"," 'NE' ");
       }
```

```
   else if (volt>=1.2 && volt<1.7){
        sprintf(direccion, "%s"," 'ESTE '");
   }

   else if (volt>=1.8 && volt<2.3){
        sprintf(direccion, "%s"," 'SE' ");
   }

   else if (volt>=2.4 && volt<2.9){
         sprintf(direccion, "%s"," 'SUR' ");
   }

   else if (volt>=3 && volt<3.5){
         sprintf(direccion, "%s"," 'SO' ");
   }

   else if (volt>=3.6 && volt<4.1){
    sprintf(direccion, "%s"," 'OESTE' ");
  }
     else if (volt>=4.2 && volt<4.7){
       sprintf(direccion, "%s"," 'NO' ");
  }
    /// MQTT
if (!client.isConnected()){
  connect();
  }
 MQTT::Message message;
  message.qos = MQTT::QOS1;
  message.retained = false;
  char str[50] = "";
  sprintf(str, "%d.%02d,%d.%02d,%d.%02d,%d.%02d,%s,%ld,%d",(int)h,(int)(h*100)%100
```

```
,(int)t,(int)(t*100)%100,(int)canLluvia,(int)(canLluvia*100)%100,(int)velocidadViento,(int)(velocidadViento*100)%100,direccion,presion,uvIntensidad);
        message.payload = str;
        message.payloadlen = strlen(str);
        int rc = client.publish("/dato", message);
           Serial.print("mensaje estado publicacion ");
           Serial.println(rc);
           Serial.print("Datos:   ");
           Serial.println(str);
           if (rc != 0) {
         Serial.print("Message publish failed with return code : ");
         Serial.println(rc);
         }
            // client.yield(10000);
delay (300000);
}
int promedioLecturaAnalog(int pinLectura)
{
  byte numLecturas = 8;
  unsigned int contaValor = 0;
  for(int x = 0 ; x < numLecturas ; x++)
    contaValor += analogRead(pinLectura);
    contaValor /= numLecturas;

  return(contaValor);
}
float mapfloat(float x, float in_min, float in_max, float out_min, float out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

ANEXO 2

Código Arduino nano

```
#include <Wire.h>
#include <TimeLib.h>
#include <DS1307RTC.h>  // Liberia para reloj DS1307
// CONFIGURACIÓN DE USUARIO
int hora_alarma = 23; // Horas alarma. Formato 24h
int minutos_alarma = 59; // Minutos alarma
int segundos_alarma = 49; // Segundos alarma
//int duracion_alarma = 10; // Tiempo activo de la alarma. Debe ser menor que la próxima alarma.
boolean comp_hora = false; // Alarma para todas las horas
boolean comp_minuto = false; // Alarma para todos los minutos
boolean comp_segundo = false; // Alarma para todos los segundos
int LedPin = 4;
unsigned char lectura=0;          // Establecemos una variable global, y la seteamos en cero
const int intPin = 2;
//variables pluviometro:
const int timeLimite = 150;
volatile int pluvioContador = 0;
int contador = 0;
long timeContador = 0;
float canLluvia =0;
int segundo=0;
void setup()
{
  Wire.begin(1);                // Nos asignamos el numero 1 como esclavos
  Serial.begin(9600);           // Iniciamos el serial 9600 baudios, para testear
  pinMode(intPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(intPin), debounceCount, CHANGE); // interrupcion del pluviometro
```

```
  Wire.onRequest(respuestaEvento);  // Al activarse el evento de peticion, se ejecuta la funcion respuestaEvento
  Wire.onReceive(recibidoEvento);    // Al activarse el evento de lectura, se ejecuta la funcion recibidoEvento
  pinMode(LedPin, OUTPUT);
   while (!Serial) ;
    setSyncProvider(RTC.get);   // funcion  get para el  RTC
     if(timeStatus()!= timeSet)
     Serial.println("No se puede sincronizar");
  else
     Serial.println("RTC active hora del sistema");
}
void loop()
{
   lectura = pluvioContador;  // La variable global es igual a la lectura analogica del pin A0 dividido en 4 para que entre en un byte (0 a 255)
   Serial.println(lectura);
   //void digitalClockDisplay();
   //delay(1000);
//}
//void digitalClockDisplay(){
  // digital clock display of the time
  Serial.print(hour());
  printDigits(minute());
  printDigits(second());
  if (hora_alarma == hour() || comp_hora) {
         if (minutos_alarma == minute() || comp_minuto) {
           if (segundos_alarma ==second() || comp_segundo) {
           digitalWrite(LedPin, HIGH);
           Serial.println("----------");
           pluvioContador=0;
```

```
            lectura=0;
            Serial.println("ALARMA");
            Serial.println("------------");
            delay(1000);
            digitalWrite(LedPin, LOW);
            }
       }
  }
  delay(1000);
}
void respuestaEvento()// Evento de peticion responde cuando pide que le enviemos algun dato
{
     Wire.write(lectura);// Enviamos el resultado de la lectura al maestro que lo solicito
     Serial.print("lectura");
     Serial.println(lectura);        // Enviamos el resultado por el puerto serie para testear
    }
void recibidoEvento(int howMany)  // Evento de recepcion
{
  unsigned char pedido;
  while( Wire.available())        // Leemos hasta que no haya datos.
  {
    pedido = Wire.read();         // Leemos el 1
     }
}
void debounceCount()
{
 if (millis() > timeContador + timeLimite)
 {
 pluvioContador++;
 timeContador = millis();
 }}
```

ANEXO 3

Clases Java

Conexión a la base de datos

```
package ec.edu.ups.conection;
import java.sql.*;
public class Conexion {
        private static Connection cnx = null;
          public static Connection getConexion() throws SQLException, ClassNotFoundException
{
            if (cnx == null) {
              try {
                Class.forName("com.mysql.jdbc.Driver");
                cnx = DriverManager.getConnection("jdbc:mysql://localhost/bdmeteo", "root",
"root");
              } catch (SQLException ex) {
                throw new SQLException(ex);
              } catch (ClassNotFoundException ex) {
                throw new ClassCastException(ex.getMessage());
              }
            }
            return cnx;
          }
          public static void cerrar() throws SQLException {
            if (cnx != null) {
              cnx.close();
            }
          }
}
```

Inserta los Datos obtenidos por los sensores

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import ec.edu.ups.modelo.dao.Tdatosmeteo;
public class ManagerConexion {
	Connection conn;
	public ManagerConexion() throws ClassNotFoundException, SQLException {
		// constructor Auto-generado stub
		conn = Conexion.getConexion();
	}
	public void insert(Tdatosmeteo reg) throws SQLException{
		String insert = "insert into bdmeteo.tdatosmeteo(fecha,hora,humedad,temperatura,cantidadlluvia,velocidadviento,direccion,presion,irradiacion,coorHora,coorMinutos,coorSegundos) "
				+ "values (?,?,?,?,?,?,?,?,?,?,?,?)";
		PreparedStatement preparedStmt = conn.prepareStatement(insert);
		preparedStmt.setString(1, reg.getFecha());
		preparedStmt.setString(2, reg.getHora());
		preparedStmt.setDouble(3, reg.getHumedad());
		preparedStmt.setDouble(4, reg.getTemperatura());
		preparedStmt.setDouble(5, reg.getCantidadlluvia());
		preparedStmt.setDouble(6, reg.getVelocidadviento());
		preparedStmt.setString(7, reg.getDireccion());
		preparedStmt.setDouble(8, reg.getPresion());
		preparedStmt.setDouble(9, reg.getIrradiacion());
		preparedStmt.setInt(10, reg.getCoorHora());
		preparedStmt.setInt(11, reg.getCoopMinuto());
		preparedStmt.setInt(12, reg.getCoopSegundos());
		preparedStmt.execute();
	}  }
```

```
package ec.edu.ups.mqtt;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttException;
public class Subscriber {
    public static final String BROKER_URL = "tcp://192.168.1.210:1883";
    private MqttClient client;
    public Subscriber() {
        String clientId = "demonio";
        try {
            client = new MqttClient(BROKER_URL, clientId);
        }
        catch (MqttException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
    public void start() {
        try {
             System.out.println("inicializa clase de respuesta");
            client.setCallback(new SubscribeCallback());
            System.out.println("espera conexion");
            client.connect();
            System.out.println("envia informacion recibida de mqtt");
            client.subscribe("/dato");
            client.subscribe("/dato");
        }
        catch (MqttException e) {
            e.printStackTrace();
            System.exit(1);
        }
   } }
```

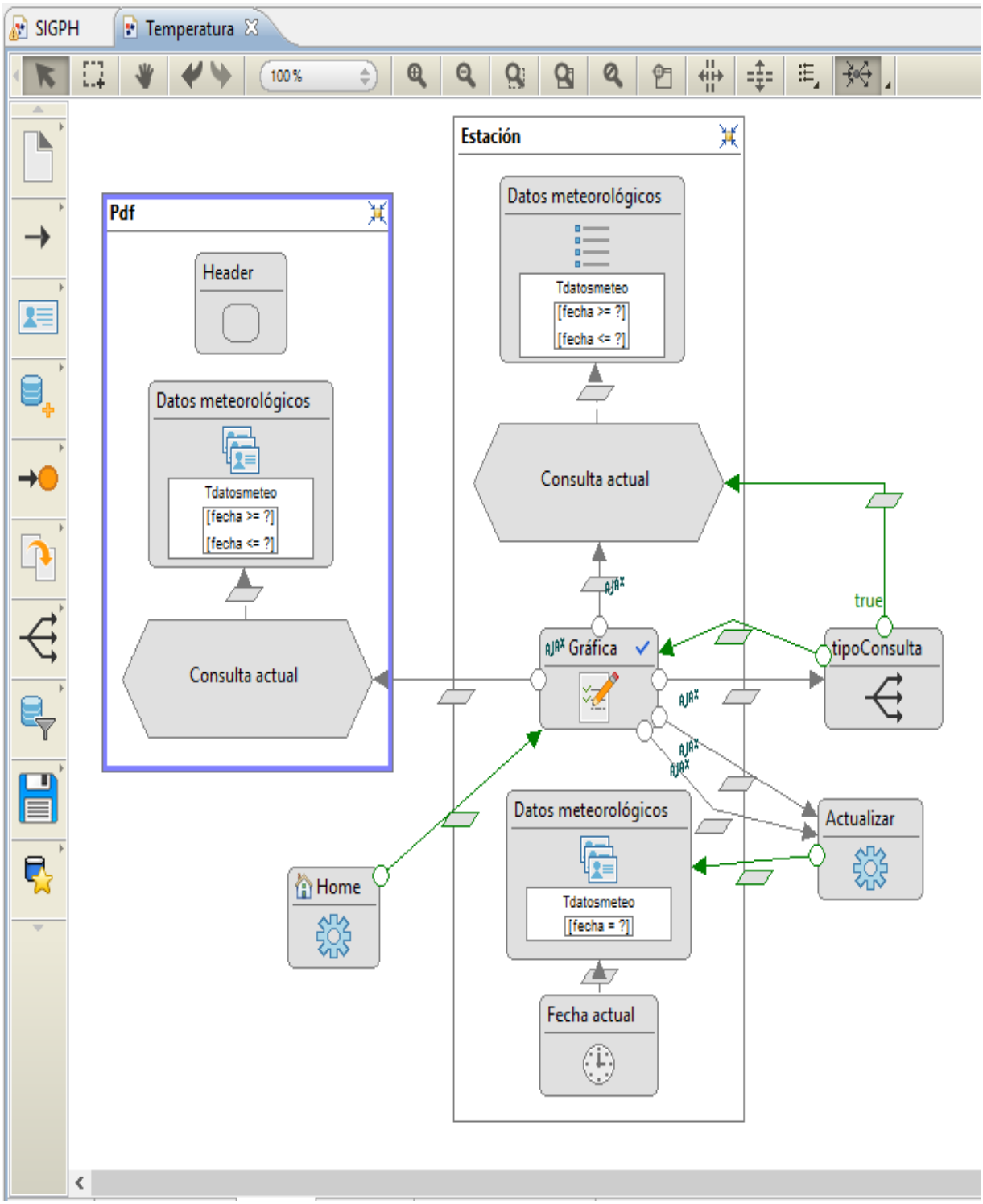
Lectura de los datos obtenidos y separación de los mismo para la introducción en la base de datos

```
package ec.edu.ups.app.controller;
import java.sql.SQLException;
import ec.edu.ups.controller.dto.TdatosmeteoController;
import ec.edu.ups.mqtt.LanzarDemonio;
import ec.edu.ups.mqtt.SubscribeCallback;
public class Excecute {

	public Excecute() {
		// TODO Auto-generated constructor stub
	}
	public static void executar(boolean isTest) throws ClassNotFoundException, SQLException{
		new LanzarDemonio(isTest).start();
		String datosLeidos = null;
		while(true){
			while(SubscribeCallback.getLeer()){
				do{
					datosLeidos = SubscribeCallback.getInformacion();
				}while(datosLeidos==null);
				System.out.println("Datos leidos="+datosLeidos);
				new TdatosmeteoController().grabarDatos(datosLeidos.split(","));
			}
		}
	}
}
```

ANEXO 4

Código PAGINA WEB



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
        <%@ taglib prefix="html" uri="http://struts.apache.org/tags-html" %>
        <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
        <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
        <%@ taglib prefix="bean" uri="http://struts.apache.org/tags-bean" %>
        <%@ taglib prefix="webratio" uri="http://www.webratio.com/2006/TagLib/JSP20" %>
        <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ page contentType="text/html; charset=UTF-8"%>
<webratio:Page page="page1"/>
<html>
  <head>
    <link href="Temperatura_Style/img/favicon.ico" rel="shortcut icon">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="Content-type" content="text/html;charset=UTF-8">
    <title>Estaci&oacute;n</title>
    <c:set var="wrAjaxDebugLevel" value="minimal"/>
        <link href="<webratio:Resource path="WRResources/ajax/jquery-
ui/themes/smoothness/jquery-ui.min.css"/>" type="text/css" rel="stylesheet"><!--[data] {"wr-
resname": "jquery-ui-style", "wr-resver": "1.9.2"}-->
        <script src="<webratio:Resource path="WRResources/ajax/jquery/jquery.min.js"/>"
type="text/javascript"></script><!--[data] {"wr-resname": "jquery", "wr-resver": "1.7.2"}-->
        <script src="<webratio:Resource path="WRResources/script.js"/>"
type="text/javascript"></script><!--[data] {"wr-resname": "wr-utils-supportscripts", "wr-resver":
"7.2.12"}-->
        <script src="<webratio:Resource path="WRResources/ajax/jquery-ui/jquery-ui.min.js"/>"
type="text/javascript"></script><!--[data] {"wr-resname": "jquery-ui", "wr-resver": "1.9.2"}-->
        <link href="<webratio:Resource path="WRResources/ajax/jquery-
ui/themes/webratio/style.css"/>" type="text/css" rel="stylesheet"><!--[data] {"wr-resname": "wr-ui-
style", "wr-resver": "7.2.12"}-->
```

<c:if test="${not(empty webratio:expandLayoutResourceContent('BUILTIN/wr-runtime', 'WRResources/ajax/webratio/', pageContext))}">
<script src="<webratio:Resource path="${webratio:expandLayoutResourceContent('BUILTIN/wr-runtime', 'WRResources/ajax/webratio/', pageContext)}"/>" type="text/javascript"></script><!--[data] {"wr-resname": "wr-runtime", "wr-resver": "7.2.12"}-->
<c:if test="${not wrAjax and not webratio:isWindow(pageContext)}">
<script type="text/javascript">
<c:choose>
<c:when test="${wrBoxed}">
(wr._configs = (wr._configs || {}))["${wrClientAppName}"] = (function() {
var app = new wr.app.App("${wrClientAppName}");
app.configure({
</c:when>
<c:otherwise>
(function() {
wr.getApp().mergeConfig({
</c:otherwise>
</c:choose>
log: {
implementation: "wr.log.LogPlugin",
appenders: [
"wr.log.HtmlAppender"
],
categories: {
<c:choose>
<c:when test="${wrAjaxDebugLevel eq 'full'}">
"": wr.log.Level.DEBUG
</c:when>

```
                                        <c:otherwise>
                                                "": wr.log.Level.WARN,
                                                "wr.logic.AjaxRequestActor._response":
wr.log.Level.DEBUG,
                                                "wr.ui.html.ElementViewer._code":
wr.log.Level.DEBUG,
                                                "wr.widgets": wr.log.Level.DEBUG
                                        </c:otherwise>
                                </c:choose>
                        }
                },
                ui: {
                        implementation: "wr.ui.UIPlugin",
                        factories: {
                                "display": "wr.ui.DisplayControlFactory",
                                "window": "wr.ui.WindowControlFactory",
                                "default": "wr.ui.ViewerBasedControlFactory"
                        },
                        display: {
                                factory: "display",
                                viewer: "wr.ui.web.BrowserViewer",
                                views: {
                                        "${wrCurrentWindowName}": {
                                                factory: "window",
                                                viewer: "${wrBoxed ? 'wr.ui.web.BoxViewer' :
'wr.ui.web.WindowViewer'}",
                                                viewerConfig: { <c:if
test="${wrBoxed}">boxSelector: "#wr-${wrClientAppName}"</c:if> },
                                                views: {
                                                        "${wrCurrentWindowName}_page": {
```

```
                                                    viewer:
"wr.ui.html.ContainerElementViewer",
                                                    viewerConfig: { containersFilter:
".wr-ajaxDiv" }
                                                }
                                            }
                                        }
                                    }
                                }
                            },
                            nav: {
                                implementation: "wr.nav.NavPlugin",
                                routers: [ {
                                    name: "wr.nav.PropagationRouter",
                                    priority: -50
                                }, {
                                    name: "wr.nav.HttpWebRouter",
                                    priority: -100
                                } ]
                            },
                            logic: {
                                implementation: "wr.logic.LogicPlugin"
                            },
                            oldajax: {
                                implementation: "wr.LegacyAjaxPlugin",
                                    containersFilter: ".wr-ajaxDiv"
                                }
                            <c:choose>
                                <c:when test="${wrBoxed}">
                                        }).chain(function() {
                                            app.init();
```

```
                                        }).chain(function() {
                                                app.start();
                                        });
                                });
                        </c:when>
                        <c:otherwise>
                                        });
                                })();
                        </c:otherwise>
                </c:choose>
        </script>
</c:if>
                        </c:if>
        <link href="<webratio:Resource path="wr-css/global.css"/>" type="text/css"
rel="stylesheet"><!--[data] {"wr-resname": "WebRatio-global-style"}-->
        <link href="<webratio:Resource path="wr-css/menu.css"/>" type="text/css"
rel="stylesheet"><!--[data] {"wr-resname": "WebRatio-menu-style"}-->
                        <!--[if lt IE 9]>
        <script src="<webratio:Resource path="wr-js/respond.min.js"/>"
type="text/javascript"></script><!--[data] {"wr-resname": "WebRatio-respond-js"}-->
                        <![endif]-->
        <link href="<webratio:Resource path="wr-css/units.css"/>" type="text/css"
rel="stylesheet"><!--[data] {"wr-resname": "WebRatio-units-style"}-->
        <link href="<webratio:Resource path="builtin/960_Fluid_Nestable_12.css"/>"
type="text/css" rel="stylesheet"><!--[data] {"wr-resname": "wr-960gs-12"}-->
                        <c:if test="${not(empty
webratio:expandLayoutResourceContent('BUILTIN/jquery-ui-datepicker-lang',
'WRResources/ajax/jquery-ui/i18n/', pageContext))}">
        <script src="<webratio:Resource
path="${webratio:expandLayoutResourceContent('BUILTIN/jquery-ui-datepicker-lang',
'WRResources/ajax/jquery-ui/i18n/', pageContext)}"/>" type="text/javascript"></script><!--[data]
{"wr-resname": "jquery-ui-datepicker-lang", "wr-resver": "1.9.2"}-->
```

```
                </c:if>
        <script src="<webratio:Resource path="wr-js/responsive-menu.js"/>"
type="text/javascript"></script><!--[data] {"wr-resname": "WebRatio-responsive-menu-js"}-->
        <script src="<webratio:Resource path="Temperatura_Style/js/highcharts.js"/>"
type="text/javascript"></script><!--[data] {"wr-resname": "highcharts"}-->
        <script src="<webratio:Resource path="WRResources/ajax/webratio/calendar-utils.js"/>"
type="text/javascript"></script><!--[data] {"wr-resname": "wr-calendar-utils", "wr-resver":
"7.2.12"}-->
        <link href="<webratio:Resource path="wrdefault/css/default.css"/>" type="text/css"
rel="stylesheet"><!--[data] {"wr-resname": "wrdefault-style"}-->
        <link href="<webratio:Resource path="builtin/grid_elements.css"/>" type="text/css"
rel="stylesheet"><!--[data] {"wr-resname": "wr-ui-gridsystem"}-->
<script type="text/javascript">if (typeof wr !== "undefined") { wr.ui.html.resx.refreshLoaded();
}</script>
 </head>
 <body>
        <c:if test="${wrBoxed}">
                <div <c:if test="${not wrAjax}">id="wr-${wrClientAppName}" data-wr-
appid="${wrClientAppName}"</c:if> class="wr-appui wr-appui-${wrClientAppName}">
        </c:if>
        <webratio:CollectScripts var="inlineScripts" enabled="${wrBoxed}"
eventHandlerWrapper="wr.keepScoped">
                <script type="text/javascript">
                        if (typeof wr !== "undefined") {
                                WRAjax.onReady(function() {
                                        wr.getApp().getPlugin("ui").configureViews(null, {
                                                _oldajax_ww: {
                                                        factory: "window",
                                                        viewer: "wr.ui.jquery.WaitDialogViewer"
                                                }
                                        });
                                });
```

```
                }
            </script>
                <html:form action="form_page1.do${wrAjax ? '#!ajax=true' : ''}"
styleId="page1FormBean" enctype="multipart/form-data">
            <div class="wr-ajaxDiv" id="page1HiddenFields">
                <html:hidden property="lastURL" styleId="lastURL_page1"/>
                    <input type="hidden" name="ln6" value="<webratio:Link
link="ln6"/>">
                        <input type="hidden" name="ln6_sr"
value="<webratio:Link link="ln6" selectiveRefresh="true"/>">
                    <input type="hidden" name="ln22" value="<webratio:Link
link="ln22"/>">
                        <input type="hidden" name="ln22_sr"
value="<webratio:Link link="ln22" selectiveRefresh="true"/>">
                    <input type="hidden" name="ln8" value="<webratio:Link
link="ln8"/>">
                        <input type="hidden" name="ln8_sr"
value="<webratio:Link link="ln8" selectiveRefresh="true"/>">
                    <input type="hidden" name="ln12" value="<webratio:Link
link="ln12"/>">
                        <input type="hidden" name="ln12_sr"
value="<webratio:Link link="ln12" selectiveRefresh="true"/>">
                    <input type="hidden" name="ln13" value="<webratio:Link
link="ln13"/>">
            </div>
        <div id="top">
            <div class="user">
                <div class="container">
                </div>
            </div>
            <div class="logo">
                <div class="container">
```

```
                                <div class="header-container">
                                        <img src="Temperatura_Style/img/header-logo.jpg"
class="header-logo">
                                </div>
                        </div>
                </div>
                <div class="main-menu">
                        <div class="container">
                                <a href="#navmenu" title="Menu" class="toggle-
menu">Menu</a>
                                <ul id="navmenu">
                                        </ul>
                                </div>
                </div>
            </div>
                <div id="body" class="container">
                        <div id="grid" class=" ">
                        <div class="wr-ajaxDiv" id="page1_grid_0">
<div class="container_12">
                                <div class="grid_4 alpha agrd_8">
                                        <div class="wr-ajaxDiv" id="page1_cell_0">
        <div class="wr-ajaxDiv" id="enu2_0">
<div class="frame">
        <div class="frame-title">Tipo de consulta</div>
        <div class="frame-content">
         <div class="plain ">
                <div class="plain EntryUnit">
                                <table>
                                        <tr>
                                                <td colspan="3" class="error">
                                                        <html:errors property="enu2" />
```

```
                                        </td>
                                    </tr>
                                    <tr class="row">
                                            <th valign="middle" class=" header">
                                                Consulta actual
                                            </th>
                                        <td valign="middle" nowrap="nowrap" class="
value">
                                                <div>
                                    <html:radio styleClass="wr-submitButtons:ln12,ln22  field"
property="fld3_locale" value="true" disabled="false"/>
                                    <bean:message key="boolean.yes"/>
                                    <html:radio styleClass="wr-submitButtons:ln12,ln22  field"
property="fld3_locale" value="false" disabled="false"/>
                                    <bean:message key="boolean.no"/>
                                </div>
            <c:if test="false">
                    <html:hidden property="fld3_locale"/>
            </c:if><script type="text/javascript">
                    new Form.Element.RadioEventObserver("page1FormBean", "fld3_locale",
function() { ajaxRequest('page1FormBean', $H({isForm: true, pressedLink: 'button:ln8',
selectiveRefresh: true, sourcePage: 'page1', indicator: 'fld3_locale_indicator'})) });
            </script>
            <span id="fld3_locale_indicator" style="display: none"><img
src="WRResources/ajax/jquery-ui/themes/webratio/indicator.gif"></span>
                                        </td>
                                            <td class="error"><span class="
error"><html:errors property="fld3_locale"/></span></td>
                                        </tr>
<c:if test="${not(webratio:evaluateCondition('cexpr_var1', null, pageContext))}">
                                    <tr class="row">
                                            <th valign="middle" class=" header">
```

```
                                                    Fecha desde
                                                </th>
                                                <td valign="middle" nowrap="nowrap" class="
value">
                                                    <c:if
test="${not(webratio:evaluateCondition('cexpr_var1', null, pageContext))}"><div>
                                                <div style="padding-right: 24px; white-space: nowrap">
                                                    <html:text  size="25" styleId="fld4_locale"
styleClass="wr-submitButtons:ln12,ln22  field" property="fld4_locale" readonly="false"
style="width: 100%; margin: 0"/>
                                                <c:choose><c:when test="false">
                                                    <img src="Resources/calendar_disabled.gif" />
                                                </c:when><c:otherwise>
                                                    <script type="text/javascript">
                                                        jQuery(function($) {

                                                    var pattern =
$.wr.calendar.dateConfigFromJava("${datePattern}");
                                                    $('#fld4_locale').datepicker({
                                                        showOn: "button",
                                                        showWeek: true,
                                                        buttonImage:
"Resources/calendar.gif",
                                                        buttonImageOnly: true,
                                                        showButtonPanel: true,
                                                        dateFormat:
pattern.dateFormat,
                                                        firstDay: <%=
java.util.Calendar.getInstance((java.util.Locale)
session.getAttribute(org.apache.struts.Globals.LOCALE_KEY)).getFirstDayOfWeek() - 1 %>
                                                    });
                                                });
```

```
                                        </script>
                                    </c:otherwise></c:choose>
                                </div>
                            </div></c:if>
                                        </td>
                                                <td class="error"><c:if
test="${not(webratio:evaluateCondition('cexpr_var1', null, pageContext))}"><span class="
error"><html:errors property="fld4_locale"/></span></c:if></td>
                                        </tr>
</c:if>
<c:if test="${not(webratio:evaluateCondition('cexpr_var1', null, pageContext))}">
                                        <tr class="row">
                                            <th valign="middle" class=" header">
                                                Fecha hasta
                                            </th>
                                        <td valign="middle" nowrap="nowrap" class="
value">
                                                <c:if
test="${not(webratio:evaluateCondition('cexpr_var1', null, pageContext))}"><div>
                                    <div style="padding-right: 24px; white-space: nowrap">
                                        <html:text  size="25" styleId="fld5_locale"
styleClass="wr-submitButtons:ln12,ln22  field" property="fld5_locale" readonly="false"
style="width: 100%; margin: 0"/>
                                        <c:choose><c:when test="false">
                                            <img src="Resources/calendar_disabled.gif" />
                                        </c:when><c:otherwise>
                                            <script type="text/javascript">
                                                jQuery(function($) {

                                                var pattern =
$.wr.calendar.dateConfigFromJava("${datePattern}");
                                                $('#fld5_locale').datepicker({
```

```
                                                                showOn: "button",
                                                                showWeek: true,
                                                                buttonImage:
"Resources/calendar.gif",
                                                                buttonImageOnly: true,
                                                                showButtonPanel: true,
                                                                dateFormat:
pattern.dateFormat,
                                                                firstDay: <%=
java.util.Calendar.getInstance((java.util.Locale)
session.getAttribute(org.apache.struts.Globals.LOCALE_KEY)).getFirstDayOfWeek() - 1 %>
                                                                        });
                                                        });
                                                </script>
                                        </c:otherwise></c:choose>
                                </div>
                        </div></c:if>
                                                </td>
                                                        <td class="error"><c:if
test="${not(webratio:evaluateCondition('cexpr_var1', null, pageContext))}"><span class="
error"><html:errors property="fld5_locale"/></span></c:if></td>
                                                </tr>
</c:if>
                                        <tr>
                                                <td colspan="3">
                                        <table>
                                <tr>
<c:if test="${not(webratio:evaluateCondition('cexpr_var1', null, pageContext))}">
                                                <td> <c:if
test="${not(webratio:evaluateCondition('cexpr_var1', 'index', pageContext))}"><input
title="Consultar" onclick="return ajaxRequest('page1FormBean', $H({isForm: true, pressedLink:
'button:ln12', selectiveRefresh: true, sourcePage: 'page1', waitingWindow: $H({message:
```

'<bean:message key="ajax.computingRequest"/>', width: 200, height: 100, duration: 1000, className: "})})) " class=" button" id="button:ln12" name="button:ln12" type="submit" value="Consultar"></c:if>

</td>

</c:if>

<td> <input title="Refrescar" onclick="return ajaxRequest('page1FormBean', $H({isForm: true, pressedLink: 'button:ln22', selectiveRefresh: true, sourcePage: 'page1'})) " class="boton-refrescar button" id="refrescar" name="button:ln22" type="submit" value="Refrescar">

</td>

</tr>

</table>

</td>

</tr>

</table>

Datos meteorológicos

<input title="Generar reporte" onclick=" $('#page1FormBean')[0].target='_blank'" id="button:ln13" name="button:ln13" type="submit" value="Generar reporte">

```
    </div>
        </div>
        <div class="frame-content">
                        <div class="wr-ajaxDiv" id="pwu1_0">
<c:choose>
<c:when test="${not(empty pwu1) and (pwu1.dataSize gt 0)}">
                <div class="plain ">
                        <div class="plain PowerIndexUnit">
                                <div class="table-responsive">
                                <table border="0" cellspacing="1" cellpadding="2">
                                        <!-- begin header -->
                                        <tr>
                                                <th class=" header"></th>
                                        <th nowrap="nowrap" class=" header">
                                                Fecha
                                        </th>
                                        <th nowrap="nowrap" class=" header">
                                                Hora
                                        </th>
                                        <th nowrap="nowrap" class=" header">
                                                Temperatura
                                        </th>
                                        <th nowrap="nowrap" class=" header">
                                                Humedad
                                        </th>
                                        <th nowrap="nowrap" class=" header">
                                                Cantidad de lluvia
                                        </th>
                                        <th nowrap="nowrap" class=" header">
                                                Velocidad del viento
                                        </th>
```

```
                                    <th nowrap="nowrap" class=" header">
                                        Presi&oacute;n
                                    </th>
                                    <th nowrap="nowrap" class=" header">
                                        Irradiaci&oacute;n
                                    </th>
                                    <th nowrap="nowrap" class=" header">
                                        Direcci&oacute;n
                                    </th>
                                    </tr>
                                    <!-- end header -->
                                <!-- instances -->
                                <c:forEach var="current" varStatus="status"
items="${pwu1.data}">
                                    <c:set var="index" value="${status.index}"/>
                                        <tr class="row<c:if test="${pwu1.currentIndex eq
index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if>">
                                                <td class=" value<c:if
test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq
0}">Alternate</c:if>">
                                                    <img
src="WRResources/spacer.gif" class="bullet<c:if test="${pwu1.currentIndex eq
index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if>" border="0" alt=""/>
                                                </td>
                                                  <td class=" value<c:if
test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq
0}">Alternate</c:if> date">
                                                        <fmt:formatDate
value="${current.fecha}" pattern="${datePattern}"/>
                                                </td>
```

<td class=" value<c:if test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if> time">
<fmt:formatDate value="${current.hora}" pattern="${timePattern}"/>
</td> <td class=" value<c:if test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if> float">
<fmt:formatNumber value="${current.temperatura}" pattern="${floatPattern}"/> </td>
<td class=" value<c:if test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if> float">
<fmt:formatNumber value="${current.humedad}" pattern="${floatPattern}"/></td><td class=" value<c:if test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if> float">
<fmt:formatNumber value="${current.cantidadlluvia}" pattern="${floatPattern}"/> </td>
<td class=" value<c:if test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if> float"> <fmt:formatNumber value="${current.velocidadviento}" pattern="${floatPattern}"/> </td>
<td class=" value<c:if test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if> float">
<fmt:formatNumber value="${current.presion}" pattern="${floatPattern}"/> </td>
<td class=" value<c:if test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if> float">
<fmt:formatNumber value="${current.irradiacion}" pattern="${floatPattern}"/> </td>
<td class=" value<c:if test="${pwu1.currentIndex eq index}">Current</c:if><c:if test="${index mod 2 eq 0}">Alternate</c:if> string"> <c:out value="${current.direccion}"/></td></tr>
</c:forEach></table>
</c:when>
<c:otherwise>

```
                <div class="plain ">
                        <div class="plain PowerIndexUnit">
                                <table>
                                  <tr>
                                    <td><bean:message key="emptyUnitMessage"/></td>
                                  </tr>
                                </table>
                        </div>
                </div>
</c:otherwise>
</c:choose>
</div>  </div>   </div></div>  </div> </div> </div>
                                        <div class="clear"></div>
                                <div class="grid_12 alpha omega agrd_24">
                                        <div class="wr-ajaxDiv" id="page1_cell_13">
        <div class="wr-ajaxDiv" id="mdu3_0">
<c:if test="${not(empty mdu3) and (mdu3.dataSize gt 0)}">
        <c:set var="x" value=""/>
        <c:set var="y" value=""/>
        <c:set var="y2" value=""/>
        <c:set var="y3" value=""/>
        <c:set var="y4" value=""/>
        <c:set var="y5" value=""/>
        <c:set var="y6" value=""/>
        <c:forEach var="current" varStatus="status" items="${mdu3.data}">
                <c:set var="x" value="${x}${empty x ? " : ', '}${current.coorHora}" />
                <c:set var="y" value="${y}${empty y ? " : ', '}${current.temperatura}" />
                <c:set var="y2" value="${y2}${empty y2 ? " : ', '}${current.humedad}" />
                <c:set var="y3" value="${y3}${empty y3 ? " : ', '}${current.cantidadlluvia}" />
                <c:set var="y4" value="${y4}${empty y4 ? " : ', '}${current.velocidadviento}" />
                <c:set var="y5" value="${y5}${empty y5 ? " : ', '}${current.presion}" />
```

```
                <c:set var="y6" value="${y6}${empty y6 ? '' : ', '}${current.irradiacion}" />
        </c:forEach>
        <div class="container_12">
                <div class="grid_4 alpha agrd_8">
<div id="container_Temperatura" style="min-width:310px;height:400px; margin:0 auto"></div>
<script type="text/javascript">
                                        Highcharts.chart('container_Temperatura', {
                                                chart: {
                                                        type: 'spline'
                                                },
                                                title: {
                                                        text: 'Temperatura'
                                                },
                                                xAxis: {
                                                        title: {
                                                            text: 'Tiempo'
                                                        },
                                                        categories: [${x}]
                                                },
                                                yAxis: {
                                                        title: {
                                                                text: 'Temperatura'
                                                        },
                                                        labels: {
                                                             formatter: function () {
                                                                return this.value + 'º';
                                                             }
                                                        }
                                                },
                                                tooltip: {
                                                        crosshairs: true,
```

```
                shared: false
            },
            plotOptions: {
                spline: {
                    marker: {
                        radius: 4,
                        lineColor: '#FF0000',
                        lineWidth: 1    }}},
            series: [{
                name: 'Temperatura',
                color: '#FF0000',
                marker: {
                    symbol: 'square'
                },
                data: [${y}]
            }]        });
        </script>
    </div>
    <div class="grid_4 agrd_8">
<div id="container_Humedad" style="min-width:310px;height:400px; margin:0 auto"></div>
        <script type="text/javascript">
            Highcharts.chart('container_Humedad', {
                chart: {
                    type: 'spline'
                },
                title: {
                    text: 'Humedad'
                },
                xAxis: {
                    title: {
                    text: 'Tiempo'
```

```
    },
    categories: [${x}]
},
yAxis: {
    title: {
            text: 'Humedad'
    },
    labels: {
        formatter: function () {
            return this.value + '%';
        }} },
tooltip: {
    crosshairs: true,
    shared: false
},
plotOptions: {
    spline: {
        marker: {
            radius: 4,
            lineColor: '#666666',
            lineWidth: 1
        }
    }
},
series: [{
        name: 'Humedad',
    color: '#666666',
    marker: {
        symbol: 'diamond'
    },
    data: [${y2}]
```

```
                }]
            });
        </script>
    </div>
    <div class="grid_4 omega agrd_8">
<div id="container_LLuvia" style="min-width:310px;height:400px; margin:0 auto"></div>
        <script type="text/javascript">
            Highcharts.chart('container_LLuvia', {
                chart: {    type: 'spline'    },
                title: {    text: 'LLuvia'    },
                xAxis: {   title: {
                    text: 'Tiempo' },
                  categories: [${x}]  },
                yAxis: {title: {  text: 'LLuvia'
                  },
                  labels: {
                    formatter: function () {
                     return this.value;
                    }  } },
                tooltip: {
                  crosshairs: true,
                  shared: false
                },
                plotOptions: {
                  spline: {
                    marker: {
                      radius: 4,
                      lineColor: '#004080',
                      lineWidth: 1
                    }
                  }
```

```
                                    },
                                    series: [{name: 'LLuvia',
                                        color: '#004080',
                                        marker: {
                                            symbol: 'circle'
                                        },
                                        data: [${y3}]
                                    }] });
                            </script>
                        </div>
                        <div class="clear"></div>
                        <div class="grid_4 alpha agrd_8">
<div id="container_Viento" style="min-width:310px;height:400px; margin:0 auto"></div>
                            <script type="text/javascript">
                                Highcharts.chart('container_Viento', {
                                    chart: {type: 'spline'
                                    },
                                    title: {text: 'Viento'
                                    },
                                    xAxis: { title: {
                                            text: 'Tiempo'
                                        },
                                        categories: [${x}]
                                    },
                                    yAxis: { title: {
                                                text: 'Viento'
                                        },
                                        labels: {
                                            formatter: function () {
                                                return this.value;
                                            } }},
```

```
                tooltip: {
                        crosshairs: true,
                        shared: false
                },
                plotOptions: {
                        spline: {
                           marker: {
                              radius: 4,
                              lineColor: '#00FFFF',
                              lineWidth: 1
                           }}},
                series: [{
                        name: 'Viento',
                        color: '#00FFFF',
                        marker: {
                             symbol: 'triangle'
                        },
                        data: [${y4}]
                }] });
        </script>
    </div>
    <div class="grid_4 agrd_8">
<div id="container_Presion" style="min-width:310px;height:400px; margin:0 auto"></div>
        <script type="text/javascript">
                Highcharts.chart('container_Presion', {
                   chart: {
                      type: 'spline'
                   },
                   title: {
                      text: 'Presión'
                   },
```

```
xAxis: {
        title: {
       text: 'Tiempo'
    },
    categories: [${x}]
},
yAxis: {
    title: {
                text: 'Presión'
    },
    labels: {
        formatter: function () {
             return this.value;
        }
    }
},
tooltip: {
    crosshairs: true,
    shared: false
},
plotOptions: {
    spline: {
        marker: {
            radius: 4,
            lineColor: '#FF8000',
            lineWidth: 1
        }}},
series: [{
             name: 'Presión',
    color: '#FF8000',
    marker: {
```

```
                                    symbol: 'triangle-down'
                                },
                                data: [${y5}]
                            }]
                        });
                </script>
            </div>
            <div class="grid_4 omega agrd_8">
<div id="container_Irradiacion" style="min-width:310px;height:400px; margin:0 auto"></div>
                <script type="text/javascript">
                    Highcharts.chart('container_Irradiacion', {
                        chart: {type: 'spline'
                        },
                        title: { text: 'Irradiación'
                        },
                        xAxis: { title: {
                                text: 'Tiempo'
                            },
                            categories: [${x}]
                        },
                        yAxis: { title: { text: 'Irradiación'
                            },
                            labels: {
                                formatter: function () {
                                 return this.value;
                                }} },
                        tooltip: {
                            crosshairs: true,
                            shared: false
                        },
                        plotOptions: {
```

```
                                    spline: {
                                        marker: {
                                            radius: 4,
                                            lineColor: '#FFFF00',
                                            lineWidth: 1
                                        }
                                    }
                                },
                                series: [{
                                        name: 'Irradiación',
                                    color: '#FFFF00',
                                    marker: {
                                        symbol: 'square'
                                    },
                                    data: [${y6}]
                                }]
                            });
                        </script>
                    </div>  </div> </c:if>
</div> </div>
                                </div>
                                        <div class="clear"></div>
                                <div class="grid_12 alpha omega agrd_24">
                                        <div class="wr-ajaxDiv" id="page1_cell_25">
        <div class="wr-ajaxDiv" id="fld1_0"><div class="fullField">
        <div class="labelWrapper" style="width: 15ex; float: left">
<span class="boton-refrescar header">oculto</span>
</div>  <div style="float: right">
<span class="boton-refrescar error"><html:errors property="fld1"/></span>
</div> <div class="valueWrapper" style="overflow: hidden">
```

```
                        <html:text  size="25" styleId="oculto" styleClass="wr-
submitButtons:ln12,ln22,ln13 boton-refrescar" property="fld1" readonly="false"/>
</div>
</div></div>
</div> </div>            <div class="clear"></div>
</div>
</div>            </div>
                    </div>
                    </html:form>
                    <script type="text/javascript">
                            if (typeof wr !== "undefined" && wr.getApp().isConfigurable()) {
                                    wr.getApp().mergeConfig({
                                            "ui+": {            autoFocusFirstWindow: true
                                            }});}
</script>
        <script type="application/json" class="wr-linkInfos">
                <webratio:LinkInfos page="page1"/>
        </script>
                <script type="application/json" class="wr-linkInfosSelective">
                        <webratio:LinkInfos page="page1" selectiveRefresh="true"/>
                </script>
        <script type="application/json" class="wr-linkData">
                <webratio:LinkData page="page1"/>
        </script>
</webratio:CollectScripts>
<c:if test="${wrBoxed}">
                    </div>
                    <c:if test="${not wrAjax}">
                            <script type="text/javascript">
```

```
                    jQuery(function($) {
        wr.ui.html.addRemoveListener($("#wr-${wrClientAppName}")[0], $.proxy(wr.runScoped,
this, "${wrClientAppName}", wr.LegacyAjaxPlugin.exit)); });
                    wr._configs["${wrClientAppName}"]();
                </script>
            </c:if>
            <c:if test="${not(empty inlineScripts)}">
                <script type="text/javascript">
                    wr.runScoped("${wrClientAppName}", function(ajaxRequest,
WRAjaxRequest, $H, WRAjax, WRAjaxRequestUtils, WRAjaxRequestQueue, WREvent,
WREventUtils, Form) {${inlineScripts} });
                </script>
            </c:if>   </c:if>   <script type="text/javascript">
            $(document).ready(function() {
  // GRAFICA
               function graficar(){
                    $("#oculto").focus();
                        $("#refrescar").click();
               }                  // TIEMPO EN MILISEGUNDOS
               setInterval(graficar, 10000);            });
        </script>
 </body>
</html>
```

ANEXO 4

Código APP

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    ProgressDialog dialog;
    Button btnDatePicker, btnTimePicker;
    EditText txtDate, txtTime;
    private int mYear, mMonth, mDay, mHour, mMinute, mSegundos;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Date d=new Date();
        //SACAMOS LA FECHA COMPLETA
        //TextView fechaCompleta = (TextView) findViewById(R.id.txtfecha);
        SimpleDateFormat fecc=new SimpleDateFormat("yyyy-MM-dd");
        String fechacComplString = fecc.format(d);
        //fechaCompleta.setText(fechacComplString);
        //SACAMOS LA HORA
        //TextView hora = (TextView) findViewById(R.id.txthora);
        SimpleDateFormat ho=new SimpleDateFormat("hh:mm");
        String horaString = ho.format(d.getTime());
        //hora.setText(horaString);
        btnDatePicker=(Button)findViewById(R.id.btn_date);
        btnTimePicker=(Button)findViewById(R.id.btn_time);
        txtDate=(EditText)findViewById(R.id.in_date);
        txtTime=(EditText)findViewById(R.id.in_time);
        btnDatePicker.setOnClickListener(this);
        btnTimePicker.setOnClickListener(this);
        txtDate.setText(fechacComplString);
        txtTime.setText(horaString);   }
```

```
@Override
public void onClick(View v) {
   if (v == btnDatePicker) {
      // Get Current Date
      final Calendar c = Calendar.getInstance();
      mYear = c.get(Calendar.YEAR);
      mMonth = c.get(Calendar.MONTH);
      mDay = c.get(Calendar.DAY_OF_MONTH);
      DatePickerDialog datePickerDialog = new DatePickerDialog(this,
            new DatePickerDialog.OnDateSetListener() {
               @Override
               public void onDateSet(DatePicker view, int year,
                                 int monthOfYear, int dayOfMonth) {
                  String sDayOfMonth="";
                  if(dayOfMonth<10){
                     sDayOfMonth="0"+String.valueOf(dayOfMonth);
                  }else{
                     sDayOfMonth=String.valueOf(dayOfMonth);
                  }
                  String sMonthOfYear="";
                  if((monthOfYear + 1)<10){
                     sMonthOfYear="0"+String.valueOf(monthOfYear + 1);
                  }else{
                     sMonthOfYear=String.valueOf(monthOfYear + 1);
                  }
                  txtDate.setText(year + "-" +sMonthOfYear+ "-" +sDayOfMonth);
               }
            }, mYear, mMonth, mDay);
      datePickerDialog.show();
   }
```

```
    if (v == btnTimePicker) {
        // Obtener hora actual
        final Calendar c = Calendar.getInstance();
        mHour = c.get(Calendar.HOUR_OF_DAY);
        mMinute = c.get(Calendar.MINUTE);
        // tiempo de lanzamiento
        TimePickerDialog timePickerDialog = new TimePickerDialog(this,
                new TimePickerDialog.OnTimeSetListener() {
                    @Override
                    public void onTimeSet(TimePicker view, int hourOfDay,
                                          int minute) {
                        String sMinute="";
                        if(minute<10){
                            sMinute="0"+String.valueOf(minute);
                        }else{
                            sMinute=String.valueOf(minute);
                        }
                        String sHourOfDay="";
                        if(hourOfDay<10){
                            sHourOfDay="0"+String.valueOf(hourOfDay);
                        }else{
                            sHourOfDay=String.valueOf(hourOfDay);
                        }
                        txtTime.setText(sHourOfDay + ":" + sMinute);
                    }
                }, mHour, mMinute, false);
        timePickerDialog.show();
    }
}
public void loadCountryInfo(View view) {
    String fechacComplString=txtDate.getText().toString();
```

```
    String horaString=txtTime.getText().toString().concat(":00");
    TextView txtip=(TextView) findViewById(R.id.txt_ingresoip);
    String stxt =  txtip.getText().toString();
    String
        strURL                                                                    =
        "http://"+stxt+"/Temperatura/Servicios/datosTemperatura/getTemperatura?fecha=".concat(f
        echacComplString).concat("&hora=").concat("horaString");
   wsAsyncTask ws=new wsAsyncTask();
    ws.execute(strURL);


}
//Sub clases  Tarea asíncrona
private class wsAsyncTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String[] strURL) {
        return requestWebService(strURL[0]);
    }
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        dialog = new ProgressDialog(MainActivity.this);
        dialog.setTitle("Please Wait..");
        dialog.setMessage("Loading...");
        dialog.setCancelable(false);
        dialog.show();
    }
    @Override
    protected void onPostExecute(String result) {
        TextView thumedad=(TextView) findViewById(R.id.txtHumedad);
        TextView ttemperatura=(TextView) findViewById(R.id.txtTemperatura);
        TextView tcantidadlluvia=(TextView) findViewById(R.id.txtcantidadlluvia);
```

```
    TextView tvelocidadviento=(TextView) findViewById(R.id.txtvelocidadviento);
    TextView tdireccion=(TextView) findViewById(R.id.txtdireccion);
    TextView tpresion=(TextView) findViewById(R.id.txtpresion);
    TextView tirradiacion=(TextView) findViewById(R.id.txtirradiacion);
    try {
        JSONArray rootArray=new JSONArray("["+result+"]");
        JSONObject rootObject=rootArray.getJSONObject(0);
        thumedad.setText(rootObject.optString("humedad"));
        ttemperatura.setText(rootObject.optString("temperatura"));
        tcantidadlluvia.setText(rootObject.optString("cantidadlluvia"));
        tvelocidadviento.setText(rootObject.optString("velocidadviento"));
        tdireccion.setText(rootObject.optString("direccion"));
        tpresion.setText(rootObject.optString("presion"));
        tirradiacion.setText(rootObject.optString("irradiacion"));
    } catch (Exception e) {
        String mensaje = "No se puede establecer la conexion, intentente nuevamente. ";
        Toast.makeText(MainActivity.this, mensaje, Toast.LENGTH_SHORT).show();
    }
    dialog.dismiss();
}
public String requestWebService(String serviceUrl) {
    HttpURLConnection urlConnection = null;
    try {
        // CREA CONEXION
        URL urlToRequest = new URL(serviceUrl);
        urlConnection = (HttpURLConnection) urlToRequest.openConnection();
        urlConnection.setConnectTimeout(15000);
        urlConnection.setReadTimeout(10000);
        // get JSON data
        InputStream in = new BufferedInputStream(urlConnection.getInputStream());
        // converting InputStream into String
```

```
            Scanner scanner = new Scanner(in);
            String strJSON = scanner.next();
            scanner.close();
            return strJSON;
        } catch (MalformedURLException e) {
            e.printStackTrace();
            // URL ES INVALIDO
        } catch (SocketTimeoutException e) {
            e.printStackTrace();
            // RECUPERACIÓN DE DATOS O CONEXIÓN AGOTADA.
        } catch (IOException e) {
            e.printStackTrace(); // NO SE PUDO CREAR LA SECUENCIA DE ENTRADA
        } finally {
            if (urlConnection != null) {
                urlConnection.disconnect();      }     }
        return null;
    }
  }
}
```

activity_Main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/layPrincipal"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <LinearLayout
      android:id="@+id/layFecha"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:baselineAligned="true"
```

```
    android:orientation="horizontal" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fecha:"
        android:id="@+id/btn_date"
        android:textColor="#005500"
        android:textSize="20sp"/>
    <EditText
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:id="@+id/in_date"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:enabled="false"/>
</LinearLayout>

<LinearLayout
    android:id="@+id/layHora"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:baselineAligned="true"
    android:orientation="horizontal" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hora :"
        android:id="@+id/btn_time"
        android:textColor="#005500"
        android:textSize="20sp"/>
    <EditText
```

```
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:id="@+id/in_time"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:enabled="false"/>
</LinearLayout>


<LinearLayout
    android:id="@+id/layDatos"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:baselineAligned="true"
    android:orientation="horizontal" >

    <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/Tabla"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:stretchColumns="1" >

        <TableRow
            android:id="@+id/Cabecera"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" >

            <TextView
                android:id="@+id/ColumnaParametro"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
```

```
        android:padding="5px"
        android:text="Parámetro"
        android:textColor="#005500"
        android:textSize="26sp" />

    <TextView
        android:id="@+id/ColumnaValor"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="5px"
        android:text="Valor"
        android:textColor="#005500"
        android:textSize="26sp" />

    <TextView
        android:id="@+id/ColumnaUnidad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="5px"
        android:text="U."
        android:textColor="#005500"
        android:textSize="26sp" />
</TableRow>
<TableRow
    android:id="@+id/SeparadorCabecera"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >
    <FrameLayout
        android:id="@+id/LineaCabecera"
```

```
        android:layout_width="fill_parent"
        android:layout_height="2px"
        android:layout_span="6"
        android:background="#FFFFFF" >
    </FrameLayout>
</TableRow>
<TableRow
    android:id="@+id/Fila0"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >
    <TextView
        android:id="@+id/lblHumedad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Humedad:"
        android:textSize="20sp" />
    <TextView
        android:id="@+id/txtHumedad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="20sp"
        android:hint="@string/shumedad"/>
    <TextView
        android:id="@+id/lblUhumedad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/uhumedad"
        android:textSize="20sp" />
```

```
</TableRow>
<TableRow
   android:id="@+id/Fila1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:id="@+id/lblTemperatura"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:gravity="left"
      android:text="Temperatura:"
      android:textSize="20sp" />

   <TextView
      android:id="@+id/txtTemperatura"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:textSize="20sp"
      android:hint="@string/stemperatura"/>
   <TextView
      android:id="@+id/lblUtemperatura"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:text="@string/utemperatura"
      android:textSize="20sp" />
</TableRow>

<TableRow
```

```
    android:id="@+id/Fila2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/lblcantidadlluvia"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Cantidad Lluvia:"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/txtcantidadlluvia"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="20sp"
        android:hint="@string/scantidadlluvia"/>
    <TextView
        android:id="@+id/lblUcantidadlluvia"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/ucantidadlluvia"
        android:textSize="20sp" />
</TableRow>

<TableRow
    android:id="@+id/Fila3"
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/lblvelocidadviento"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Velocidad Viento:"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/txtvelocidadviento"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="20sp"
        android:hint="@string/svelocidadviento"/>
    <TextView
        android:id="@+id/lblUvelocidadviento"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/uvelocidadviento"
        android:textSize="20sp" />
</TableRow>

<TableRow
    android:id="@+id/Fila4"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >
    <TextView
```

```
        android:id="@+id/lbldireccion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Direccion Viento:"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/txtdireccion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="20sp"
        android:hint="@string/sdireccionviento"/>
    <TextView
        android:id="@+id/lblUdireccion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/udireccionviento"
        android:textSize="20sp" />
</TableRow>


<TableRow
    android:id="@+id/Fila5"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >
    <TextView
        android:id="@+id/lblpresion"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Presion:"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/txtpresion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="20sp"
        android:hint="@string/spresion"/>
    <TextView
        android:id="@+id/lblUpresion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/upresion"
        android:textSize="20sp" />
</TableRow>
<TableRow
    android:id="@+id/Fila6"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >
    <TextView
        android:id="@+id/lblirradiacion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Irradiacion:"
        android:textSize="20sp" />
```

```
        <TextView
            android:id="@+id/txtirradiacion"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:textSize="20sp"
            android:hint="@string/sirradiacion"/>
        <TextView
            android:id="@+id/lblUirradicion"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="@string/uirradiacion"
            android:textSize="20sp" />
      </TableRow>
   </TableLayout>
</LinearLayout>
<LinearLayout
   android:id="@+id/layDatosIp"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:baselineAligned="true"
   android:orientation="vertical" >
   <Button
      android:id="@+id/buttonconsultarid"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_gravity="center"
      android:onClick="loadCountryInfo"
      android:text="@string/enviaConsulta" />
   <LinearLayout
```

```
        android:id="@+id/ingresoip"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:baselineAligned="true"
        android:orientation="horizontal" >
        <TextView
            android:id="@+id/lbl_ingresoip"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="left"
            android:text="IP:"
            android:textSize="20sp" />
        <EditText
            android:layout_width="200dp"
            android:layout_height="wrap_content"
            android:id="@+id/txt_ingresoip"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:text="192.168.88.127:8080"
            android:enabled="true"/>
      </LinearLayout>
   </LinearLayout>

</LinearLayout>
```