



SISTEMA DE RECOGIDA, ALMACENAMIENTO Y ENVÍO INALÁMBRICO DE INFORMACIÓN DE SENsoRES

Trabajo Final de Grado

Realizado en la facultad de

*Escola Tècnica Superior d'Enginyeria de Telecomunicació
de Barcelona*

Universitat Politècnica de Catalunya

por

Óscar Liñán López

En cumplimiento parcial

de los requisitos para el grado en

Ciències i Tecnologies de les Telecomunicacions

Tutor: Juan Antonio Chávez

Barcelona, Mayo 2018



Abstract

The aim of this project is designs a system to be able to manage sensor networks easily by the final user. This sensor network can measuring the malolactic fermentation in wines. We propose, design and implement a wireless system for the sensors in this project. After this, will be possible manage and get data of each of them. In addition, it will be possible to synchronize data to a database and plot data into chart. Throughout the project we will specify and justify the hardware used and the software implementation methodology. Finally, we will verify the correct behaviour of the designed system.

Resum

El projecte pretèn desenvolupar un sistema que permeti a l'usuari final gestionar de manera senzilla xarxes de sensors que mesuren el nivell de fermentació malolàctica del vi. S'ha proposat, dissenyat i implementat un sistema que sigui capaç de proveir de comunicacions sense cable als sensors de la xarxa mencionada. Gràcies a això es podrà gestionar i obtenir-ne les dades de cadascun d'ells. A més, mitjançant una APP executant-se en un dispositiu amb connexió a internet, es podran sincronitzar les dades cap a una base de dades i es podran visualitzar gràficament. Durant el projecte s'explicaran i justificaran les decisions preses en quant al hardware utilitzat i la implementació software. Finalment, es verificarà el correcte funcionament del sistema.



Resumen

El proyecto pretende desarrollar un sistema que permita al usuario final gestionar de manera sencilla redes de sensores que miden el nivel de fermentación maloláctica del vino. Se ha propuesto, diseñado e implementado un sistema capaz de dotar de comunicación inalámbrica a los sensores de dicha red. Gracias a ello se podrá gestionar y obtener datos del sensor. Además, mediante una APP ejecutándose en un dispositivo con conexión a internet se podrán sincronizar los datos a una base de datos y visualizarlos gráficamente. A lo largo del proyecto se explican y argumentan las decisiones tomadas en cuanto al hardware utilizado y la implementación software. Finalmente, se verifica el correcto funcionamiento del sistema.



Agradecimientos

Agradecer a mi hermano, también Ingeniero en Telecomunicaciones, los consejos y las charlas de debate acerca del desarrollo del proyecto y, en definitiva, a lo largo de toda la carrera. También agradecer a mi familia y a mi pareja hacerme un poco más llevadera la realización de este proyecto.

Por último, y no en menor medida, agradecer a Juan Antonio Chávez sus consejos e indicaciones acerca de la realización del proyecto y su completa disponibilidad para cualquier duda que me pudiera surgir.



Historial de revisión y aprobación

Revisión	Fecha	Propósito
0	15/02/2018	Creación del documento
1	03/05/2018	Primera revisión del documento
2	09/05/2018	Segunda revisión del documento
3	11/05/2018	Revisión final del documento

LISTADO DE DISTRIBUCIÓN DEL DOCUMENTO

Nombre	e-mail
Óscar Liñán López	oscar.linan.lopez@gmail.com
Juan Antonio Chávez	juan.antonio.chavez@upc.edu

Escrito por:		Revisado y aprobado por:	
Fecha	11/05/2018	Fecha	11/05/2018
Nombre	Óscar Liñán López	Nombre	Juan Antonio Chávez
Posición	Autor del proyecto	Posición	Supervisor del proyecto

Tabla de contenidos

Abstract	1
Resum.....	2
Resumen.....	3
Agradecimientos.....	4
Historial de revisión y aprobación	5
Tabla de contenidos	6
Lista de figuras:	8
Lista de tablas:	10
1. Introducción:.....	11
1.1. Antecedentes del proyecto	11
1.2. Objetivos	12
1.3. Requerimientos y especificaciones.....	13
2. Estado del arte de la tecnología usada o aplicada:.....	14
3. Desarrollo del proyecto:.....	17
3.1. Propuesta de diseño.....	17
3.2. Funcionamiento del sistema y diagrama de bloques	18
3.3. Protocolo de comunicaciones alámbrico.....	22
3.3.1. Estructura de trama	22
3.3.2. Tipos de trama	23
3.3.2.1. Trama de configuración.....	23
3.3.2.2. Trama de solicitud de datos	23
3.3.2.3. Trama de datos del sensor.....	23
3.3.2.4. Tramas de confirmación: ACK y NACK	24
3.3.3. Funcionamiento del protocolo.....	24
3.3.3.1. Detección de errores	24
3.3.3.2. Flujo del programa entre extremos del canal de comunicaciones.....	25
3.4. Bloque Arduino.....	27
3.4.1. Placa Arduino NANO	27
3.4.2. Sensor AM2302.....	28
3.4.3. Software de comunicaciones alámbricas	29
3.4.4. Adquisición y visualización de datos del sensor	32
3.5. Bloque Raspberry.....	33
3.5.1. Placa Raspberry Pi Zero W	34

3.5.2. Software de comunicaciones alámbricas	35
3.5.3. Software de comunicaciones inalámbricas	38
3.5.3.1. Intercambio inicial de mensajes para la identificación del sistema.....	40
3.5.3.2. Localización del sistema en la planta	40
3.5.3.3. Inicio/Parada del proceso de mediciones	41
3.5.3.4. Transferencia de datos vía socket.....	41
3.6. Bloque Android APP	42
3.6.1. Fundamentos y pasos iniciales.....	42
3.6.2. ¿Por qué Firebase?	43
3.6.3. Adecuación del entorno para trabajar con Firebase.....	44
3.6.4. Funciones y características principales de la APP.....	45
3.6.5. Manejo de Firebase desde la APP	47
4. Puesta en marcha y resultados	51
5. Presupuesto	62
6. Conclusiones y futuros pasos:.....	63
Bibliografía:	65
Anexos:	66

Lista de figuras:

Figura 1: módulos Wi-Fi ESP-05 y ESP-201	15
Figura 2: modulo Wi-Fi NodeMCU	15
Figura 3: módulos Bluetooth HC-05 y HC-06	16
Figura 4: diferentes tipologías de diseño del sistema.....	17
Figura 5: diagrama de bloques del sistema completo	19
Figura 6: estructura de trama.....	22
Figura 7: trama de configuración	23
Figura 8: trama de solicitud de datos	23
Figura 9: trama de datos del sensor	24
Figura 10: trama de ACK	24
Figura 11: trama de NACK.....	24
Figura 12: diálogo sin errores entre extremos del canal de comunicaciones	25
Figura 13: casos en que el Master contesta con NACK.....	26
Figura 14: no recepción de trama de confirmación y reenvío de la trama anterior	26
Figura 15: pinout Arduino NANO	27
Figura 16: estructura y decodificación de trama en AM2302.....	28
Figura 17: Arduino pide datos al sensor.....	28
Figura 18: el sensor envía datos a Arduino.....	29
Figura 19: diagrama de bloques del bloque Arduino	30
Figura 20: diagrama de flujo de main_wired_arduino.ino	30
Figura 21: flujograma de la función loop()	31
Figura 22: flujograma de la función serialEvent()	32
Figura 23: conexionado del sensor auxiliar.....	32
Figura 24: formato de los datos mostrados en el LCD	33
Figura 25: diagrama de bloques del bloque Raspberry	33
Figura 26: pinout de la placa Raspberry Pi Zero W.....	34
Figura 27: flujo de funcionamiento de main_wired_rpi.py	35
Figura 28: flujograma de la función readIntoFrame()	37
Figura 29: flujograma de la función processFrame()	38
Figura 30: arquitectura servidor - cliente.....	38
Figura 31: configuración de red Wi-Fi en Raspberry	39
Figura 32: flujograma del servidor.....	39
Figura 33: montaje LED de localización.....	40

Figura 34: ciclo de vida de un Activity	42
Figura 35: diagrama de bloques del bloque Android APP	43
Figura 36: plantilla Project/bulid.gradle	44
Figura 37: plantilla Module/bulid.gradle	44
Figura 38: ciclo de vida de la clase AsyncTask.....	45
Figura 39: flujograma genérico de una petición	46
Figura 40: guardado de datos en Firebase	48
Figura 41: métodos interfaz ChildEventListener.....	48
Figura 42: estado de Firebase después del registro del sensor "tinto".	49
Figura 43: montaje final del sistema	51
Figura 44: script para pruebas serial en Arduino.....	52
Figura 45: código para pruebas serial en Raspberry	53
Figura 46: proceso de medidas visto desde Raspberry	53
Figura 47: estructura fichero measurements.txt.....	54
Figura 48: pantalla inicial de la APP	54
Figura 49: pantalla inicial con IPs registradas.....	55
Figura 50: sensores registrados pero no activos.....	55
Figura 51: pantalla Offline Options.....	55
Figura 52: información del sensor en Offline Options.....	56
Figura 53: pantalla de borrado en Offline Options.....	56
Figura 54: pantalla Online Options.....	57
Figura 55: ventana de registro	57
Figura 56: pantalla Online Options de un sensor registrado.....	58
Figura 57: pantalla Online Options con localización activada.....	59
Figura 58: ventana Erase Sensor	60
Figura 59: pantalla de gráficas.....	60
Figura 60: trama de solicitud a través de PicoScope	61

Lista de tablas:

Tabla 1: estándares de comunicaciones inalámbricas	14
Tabla 2: campos de trama	23
Tabla 3: métodos y variables de la clase myFrame.py.....	36
Tabla 4: listado de materiales	62
Tabla 5: presupuesto final	62

1. Introducción:

El presente proyecto toma como precedente el PFC “*Sistema de medida on-line ultrasónico de la fermentación maloláctica en vinos*” realizado en 2016 por Víctor Solé en el Grupo de Sistemas de Sensores (GSS) de la UPC. En dicho proyecto se implementó un sistema de medidas en tiempo real del proceso de fermentación del vino, basado en la cuantificación de la variación del tiempo de vuelo (TOF) de un eco ultrasónico.

En la industria vitivinícola es de vital importancia el proceso de fermentación maloláctica del vino, ya que después de esta fermentación tiene lugar la fermentación acética que convierte el vino en vinagre. El trabajo que se describirá en este documento supone un avance en el desarrollo del sistema de control y medida del proceso de fermentación del vino.

1.1. Antecedentes del proyecto

En los últimos años las nuevas tecnologías se han ido abriendo camino también en el sector industrial, y el sector del vino no iba a ser una excepción. En este sector era necesario implementar un sistema para poder optimizar el proceso de fermentación del vino, ya que de esta manera se puede obtener un producto de mayor calidad y además se evitan las pérdidas de producción que conllevaría obtener como resultado vinagre en lugar de vino. Gracias al sensor de ultrasonidos es posible y nos permite tener un control en tiempo real del proceso de fermentación. Este sensor ha sido probado durante procesos de fermentación en cubas de vino reales aunque, inicialmente, con instalación cableada a un PC para hacer la gestión del sistema.

En la industria alimentaria se acostumbra a utilizar productos de limpieza tanto para el interior de los depósitos como en el exterior de los mismos que pueden deteriorar notablemente cualquier instalación cableada. Además, las bodegas suelen ser lugares donde es difícil realizar instalaciones cableadas. Esta es precisamente la motivación del presente proyecto.

Por otro lado, es cada vez más frecuente la monitorización de los sistemas que nos rodean, un ejemplo serían los sistemas domóticos en viviendas. Es por esto que sería de gran utilidad que el sensor ultrasónico pudiese monitorizarse sin necesidad de cableado. Para ello entraría en juego dotarlo de comunicaciones inalámbricas, como el Bluetooth o el Wi-Fi.

Antes de centrarnos en los objetivos es necesario aclarar que para la realización de este proyecto se ha decidido no trabajar directamente con el sensor ultrasónico, ya que el tiempo es ajustado y conocer, familiarizarnos y seguramente modificar el Firmware existente del sensor requeriría demasiado tiempo. Por tanto, se ha optado por diseñar el sistema mediante un sensor auxiliar de tal manera que el sistema sea fácilmente transportable a otros sensores digitales, como por ejemplo el sensor ultrasónico.

1.2. Objetivos

El objetivo principal de este proyecto es la implementación de un sistema de comunicación inalámbrica adecuado a las condiciones de trabajo de una bodega vitivinícola, ya que como decíamos anteriormente, una bodega de vino no es el ambiente ideal para que la transmisión de datos se realice mediante una instalación cableada.

Los objetivos específicos del proyecto son los siguientes:

- Transmitir los datos generados por el sensor de forma inalámbrica a un dispositivo externo.
- Minimizar costes de instalación y mantenimiento.
- Ofrecer al usuario final mayor comodidad a la hora de gestionar el sensor.
- El sistema debe ser fácilmente transportable a diferentes tipos de sensores digitales.



1.3. Requerimientos y especificaciones

Requerimientos del proyecto:

- Utilización de los sistemas de desarrollo disponibles en el GSS.

Especificaciones del proyecto:

- Control de hasta 10 sistemas de sensado.
- Alcance inalámbrico de al menos 10 m.
- Capacidad de almacenamiento suficiente para guardar los datos producidos durante al menos un mes.
- Rango de tensiones de trabajo: 0 - 5V.
- Uso de lenguajes de programación extendidos: C, Python y JAVA.
- Coste máximo de 100€ y tamaño máximo de 15 cm x 15 cm x 10 cm, ya que cada cuba contará con su propio equipo.
- Alimentación con baterías.

2. Estado del arte de la tecnología usada o aplicada:

En este proyecto se trabaja principalmente con comunicaciones, tanto alámbricas como inalámbricas, y con el diseño e implementación de una aplicación para monitorizar un sistema de sensado a través de un dispositivo externo.

El campo de las comunicaciones alámbricas es un viejo conocido del sector de las telecomunicaciones. Cualquier microprocesador incluye interfaces de comunicaciones por cable, con lo cual el abanico de posibilidades es enorme.

Por el contrario, las comunicaciones inalámbricas (sin incluir las radiocomunicaciones) son relativamente nuevas en el sector. Aun así, la evolución del sector tecnológico en los últimos años ha sido de tal magnitud que es fácil encontrar módulos o PCB que dispongan de este tipo de comunicaciones. En sistemas como el que vamos a desarrollar podríamos hablar de dos estándares de comunicaciones inalámbricas principales:

1. Wi-Fi 802.11b
2. Bluetooth 802.15.1

En la Tabla 1 podemos ver las especificaciones principales de los dos estándares mencionados.

	Wi-Fi 802.11b	Bluetooth 802.15.1
Velocidad de Transmisión	11 Mbps	720 Kbps
Tamaño de la red	32 nodos	7 nodos
Cobertura	100 m	10 m
Banda de trabajo	2,4 GHz	

Tabla 1: estándares de comunicaciones inalámbricas

En cuanto a módulos Wi-Fi nos encontramos con la familia de los ESP fabricados por Espressif y basados en el chip ESP8266. Estos chips/módulos se acostumbran a usar en PCB Arduino o con arquitecturas basadas en PIC. Dentro de la familia tenemos módulos que ofrecen únicamente conectividad Wi-Fi u otros que además ofrecen puertos GPIO para poder utilizarlos a nuestro antojo. En la Figura 1 aparecen los dos módulos principales: el ESP-05 y el ESP-201. El ESP-201 incluye un conector para una antena externa y 11 puertos GPIO.

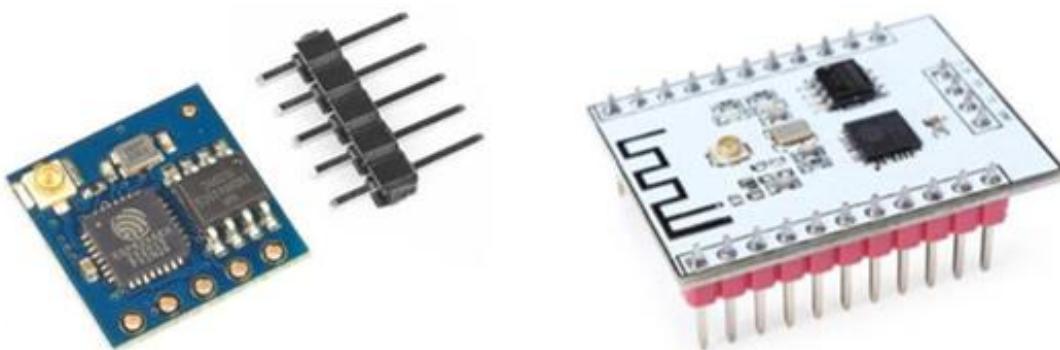


Figura 1: módulos Wi-Fi ESP-05 y ESP-201

Por último, Espressif también ha diseñado un módulo capaz de trabajar por sí solo de manera autónoma, el NodeMCU, que podemos observar en la Figura 2. Este módulo incluye un adaptador serie/USB y se alimenta a través de su puerto µ-USB. Lo más interesante de este módulo es que permite instalar un firmware base para poder trabajar con lenguajes como Python o JAVA.



Figura 2: modulo Wi-Fi NodeMCU

En cuanto a módulos Bluetooth nos encontramos con dos destacados: el HC-05 y el HC-06. En la Figura 3 podemos ver que son prácticamente iguales, salvo que el HC-05 puede actuar de Master y Slave, por el contrario el HC-06 solo puede actuar de Slave. Este tipo de módulos suelen utilizarse en conjunto con PCB Arduino o en arquitecturas basadas en PIC.

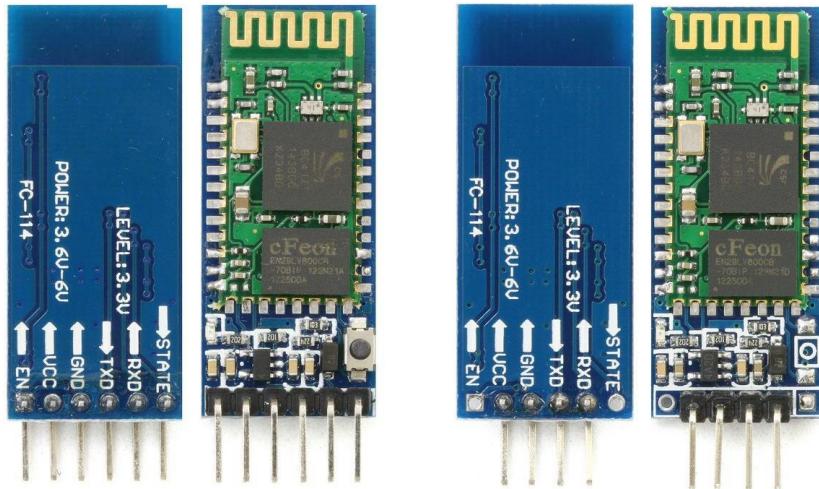


Figura 3: módulos Bluetooth HC-05 y HC-06

Englobando las dos categorías anteriores, podemos encontrar el chip CYW43438 fabricado por Cypress. Las últimas PCB Raspberry incluyen este chip, que es un “todo en uno” ya que incorpora comunicaciones Wi-Fi, Bluetooth e incluso un receptor FM.

Actualmente la sociedad es cada vez más dependiente de los dispositivos electrónicos que nos facilitan algunos aspectos del día a día. Lo cual también sucede en el ámbito profesional, en concreto en el sector tecnológico. Cada vez hay menos interacción humana en los procesos industriales ya que en muchas ocasiones están automatizados y monitorizados. Es por eso que cada vez son más necesarias aplicaciones que nos ayuden a interactuar con estos procesos para verificar que todo está sucediendo con normalidad y no existe ninguna anomalía. Aquí entraría en juego el diseño software de herramientas de control y monitorización de sistemas.

Respecto al diseño de este tipo de aplicaciones, no existen actualmente herramientas genéricas que se puedan aplicar a sensores con un uso tan específico como el sensor de ultrasonidos. Suelen tratarse de aplicaciones hechas a medida por empresas especializadas en base a un sensor o sistema a monitorizar concreto.

3. Desarrollo del proyecto:

3.1. Propuesta de diseño

En primer lugar, vamos a hacer un breve inciso en las diferentes posibilidades que se barajaron inicialmente a la hora de implementar el proyecto. Además, cabe destacar que se decidió no diseñar ninguna placa electrónica dedicada, ya que los plazos de tiempo eran demasiado cortos.

Inicialmente se plantearon dos posibles tipologías, que aparecen en formato gráfico en la Figura 4:

- Sensor auxiliar conectado directamente a la placa con comunicaciones inalámbricas más dispositivo externo (diagrama superior).
- Sensor auxiliar conectado a una placa previa y ésta conectada por cable a la placa con comunicaciones inalámbricas más dispositivo externo (diagrama inferior).

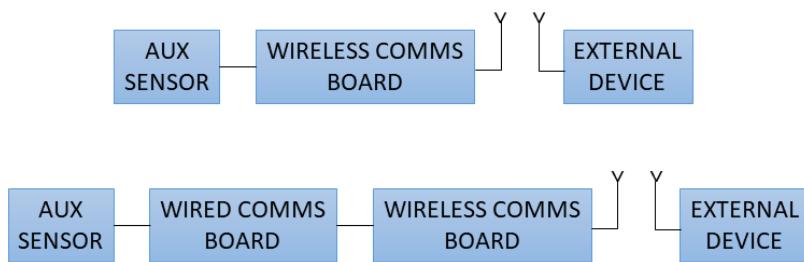


Figura 4: diferentes tipologías de diseño del sistema

La opción a) implica que la captura, el procesado y la transmisión inalámbrica de datos lo tiene que realizar la misma placa. En cambio, la opción b) nos permitiría repartir las tareas entre el bloque alámbrico y el inalámbrico. Para que el sistema sea más completo y escalable se ha decidido escoger la segunda opción. De esta manera ya quedará implementado un protocolo de comunicaciones alámbrico entre el bloque generador de datos (sensor + placa de comunicaciones alámbricas) y la placa de comunicaciones inalámbricas.

Por otro lado, se necesita una placa dotada de comunicaciones inalámbricas y con una capacidad de almacenamiento elevada, ya que el sensor puede estar almacenando datos durante mucho tiempo sin ser descargados. Aquí las opciones eran claras:

Bluetooth o Wi-Fi. Debido a que el sensor puede llegar a almacenar gran cantidad de datos antes de ser obtenidos por el sistema y a las diferencias entre especificaciones mencionadas en la Tabla 1, se ha optado por escoger el Wi-Fi.

Por último, en el dispositivo externo es necesaria una aplicación con la que poder gestionar el sensor, se barajaron las siguientes opciones:

- a) Dispositivo portátil (Smartphone o Tablet).
- b) Dispositivo fijo en la bodega.

Usar un dispositivo fijo en bodega podría suponer algunos problemas. El dispositivo necesitará disponer de conexión a internet, con lo cual tener un dispositivo fijo en planta implicaría llevar red por cable a la bodega. Las bodegas pueden estar en sitios aislados donde no sea fácil realizar una instalación de red para dar servicio, además esto podría conllevar un coste elevado. Es por este motivo que hemos elegido usar un dispositivo portátil con conexión a internet.

3.2. Funcionamiento del sistema y diagrama de bloques

En primer lugar haremos una breve descripción del funcionamiento de los bloques que aparecen en la Figura 4 (diagrama inferior). En concreto especificaremos que tareas tiene que desempeñar cada uno de ellos.

Como ya hemos comentado en el apartado 1.1, no utilizaremos el sensor de ultrasonidos sino que usaremos un sensor auxiliar. Dicho sensor debe ser fácil de manejar ya que a la hora de la verdad no formará parte del sistema. El sensor como tal simplemente se encargará de proporcionar datos a la placa de comunicaciones alámbricas cuando ésta los solicite.

La placa de comunicaciones alámbricas debe disponer de varios protocolos de comunicación ya que serán necesarios al menos dos: para comunicar con el sensor auxiliar y para transmitir los datos a la placa de comunicaciones inalámbricas. Esta placa deberá desempeñar las siguientes funciones:

- I. Leer datos del sensor auxiliar.
- II. Enviar datos por cable a la placa de comunicaciones inalámbricas.

La placa de comunicaciones inalámbricas deberá disponer también, además de Wi-Fi, de al menos un protocolo de comunicaciones por cable para comunicarse con la placa de comunicaciones alámbricas. Por la parte alámbrica deberá realizar las siguientes funciones:

- I. Solicitar datos a la placa de comunicaciones alámbricas.
- II. Recibir y almacenar los datos pertinentes de manera provisional.

Por la parte inalámbrica deberá realizar las siguientes funciones:

- I. Esperar posible conexión con la aplicación.
- II. Sincronización inicial con la aplicación.
- III. Activar indicador visual en el sensor para poder localizarlo.
- IV. Transferencia de datos.
- V. Inicio/Parada del proceso que define la parte alámbrica.
- VI. Inicialización de variables.

La aplicación que se ejecutará en el dispositivo externo deberá ser capaz de realizar las siguientes funciones:

- I. Conectarse al sensor seleccionado e identificarlo.
- II. Registrar/eliminar el sensor.
- III. Solicitar la activación/desactivación del indicador visual.
- IV. Solicitar el inicio/parada del proceso de medidas.
- V. Solicitar transferencia de archivos y sincronizarlos en la base de datos.
- VI. Obtención y presentación gráfica de los datos.
- VII. Eliminación de los datos alojados en la base de datos.

Para poder satisfacer todas las necesidades mencionadas, se ha diseñado el diagrama de bloques de la Figura 5. En dicho diagrama se ha desgranado cada uno de los bloques generales.

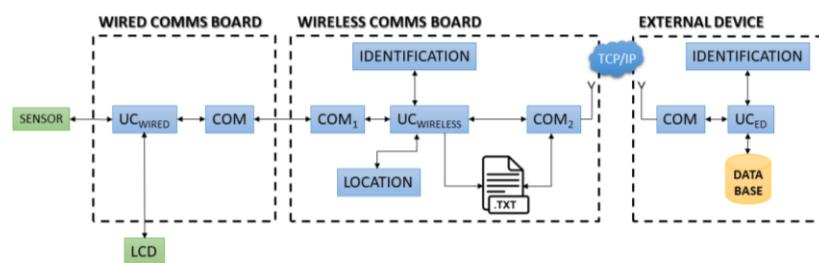


Figura 5: diagrama de bloques del sistema completo

En segundo lugar, aclararemos qué es cada uno de los sub-bloques principales que aparecen en la Figura 5:

- **Sensor:** sensor para generar datos en el sistema.
- **LCD:** display para presentar los datos obtenidos del sensor.
- Bloque placa de comunicaciones alámbricas (WIRED COMMS BOARD).
 - **UC_{WIRED}:** unidad de control para realizar las principales funciones del bloque.
 - **COM:** comunicaciones alámbricas hacia bloque inalámbrico.
- Bloque placa de comunicaciones inalámbricas (WIRELESS COMMS BOARD).
 - **UC_{WIRELESS}:** unidad de control para realizar las principales funciones del bloque.
 - **COM₁:** comunicaciones alámbricas hacia bloque alámbrico.
 - **COM₂:** comunicaciones inalámbricas hacia el dispositivo externo.
 - **IDENTIFICATION:** identificación única del sistema.
 - **LOCATION:** gestión del indicador visual.
 - **TXT:** almacenamiento temporal de los datos obtenidos del bloque alámbrico.
- Bloque dispositivo externo (EXTERNAL DEVICE).
 - **UC_{ED}:** unidad de control para realizar las principales funciones desempeñadas por el dispositivo externo.
 - **COM:** comunicaciones inalámbricas hacia el bloque inalámbrico.
 - **DATABASE:** almacenamiento de los datos obtenidos del bloque inalámbrico.

Para el bloque de comunicaciones alámbricas nos hemos decantado por una placa Arduino NANO ya que, al solo encargarse de captar datos del sensor auxiliar y enviarlos por cable, no nos era necesaria una placa muy potente ni con demasiada memoria puesto que no se almacenarán datos en este bloque. Para el sensor auxiliar se ha escogido el AM2302: un sensor de temperatura y % de humedad. El principal motivo es que necesitábamos un sensor que fuéramos capaces de poner en marcha con facilidad en Arduino. Este sensor dispone de librerías específicas para su uso en Arduino y además su salida es digital, como en el caso del sensor ultrasónico.

Para placa de comunicaciones inalámbricas hemos decidido utilizar Raspberry ya que es un sistema muy extendido, en concreto la Raspberry Pi Zero W. Cumple con los requisitos necesarios del bloque y además se puede adquirir a un precio muy competitivo. Inicialmente se barajaron algunas placas para uso industrial de Arduino o Beaglebone pero la relación calidad precio no era tan buena. También fue un motivo de peso el hecho de que ya había trabajado anteriormente con sistemas Raspberry. El diodo LED de este bloque se utilizará como indicador visual del sistema.

Como ya hemos comentado antes, el dispositivo externo será un dispositivo portátil, ya sea un Smartphone o una Tablet. Hoy en día, en este ámbito podemos hablar de dos sistemas operativos principales: iOS y Android. Nos hemos decantado por Android ya que no todo el mundo puede tener acceso a terminales de elevado coste como los de Apple y además los sistemas iOS están más protegidos frente a APP's ajenas al Apple Store. Para la base de datos hemos escogido Firebase por razones que se explicarán en el apartado 3.6.2. También se barajó la posibilidad de utilizar una base de datos local en la memoria del dispositivo, pero se desestimó ya que de cara al usuario no era demasiado útil. El usuario puede cambiar de Smartphone, lo cual provocaría la pérdida de los datos o, en su defecto, supondría realizar un backup del contenido de la base de datos que tendría que copiarse manualmente en el nuevo terminal, para finalmente restaurar los datos.

A modo de resumen, el sistema estará formado por el siguiente hardware y se utilizarán las siguientes herramientas de desarrollo software:

- Sensor auxiliar.
 - Hardware compuesto por un sensor AM2302 y un LCD 16x2.
- Bloque placa de comunicaciones alámbricas.
 - Hardware compuesto por una placa Arduino NANO.
 - Desarrollo software sobre entorno Arduino IDE 1.6.0 en lenguaje C.
- Bloque placa de comunicaciones inalámbricas.
 - Hardware compuesto por una placa Raspberry Pi Zero W y un diodo LED.
 - Desarrollo software sobre entorno Linux (Raspbian) y en Python 2.7.
- Bloque dispositivo externo.
 - Hardware compuesto por cualquier Smartphone con Android 4.0 Ice Cream o superior.

- Desarrollo software sobre entorno Android Studio 3.1 en Java sincronizado con Google Cloud Database Service (Firebase).

3.3. Protocolo de comunicaciones alámbrico

Para realizar la comunicación por cable entre Arduino y Raspberry se ha optado por diseñar un protocolo de comunicaciones serie que de robustez al sistema. Para ello se han definido unas tramas específicas, se ha acordado cómo será la interacción entre los dos extremos del canal y finalmente se ha diseñado un control de errores. Además el protocolo se ha planteado de tal manera que Arduino actuará como SLAVE y Raspberry como MASTER. A continuación, se describirán las tramas utilizadas, se especificará cual es el flujo del programa y su dinámica de funcionamiento. En nuestro caso, utilizaremos la UART con la siguiente configuración:

- 1 bit de stop
- 8 bits de datos
- Sin bit de paridad
- 9600 bps

Aunque en ambos extremos del canal de comunicaciones se puede llegar a velocidades de hasta 115200 bps, se ha desestimado usar velocidades tan altas ya que la toma de medidas tiene una periodicidad de al menos un minuto y por tanto no habrá un tráfico elevado de datos.

3.3.1. Estructura de trama

La estructura de trama que se muestra en la Figura 6 será válida para todos los tipos de trama.



Figura 6: estructura de trama

A continuación, se muestra la Tabla 2 en la que se detalla el uso y tamaño de cada campo de la trama.

CAMPO DE TRAMA	DESCRIPCIÓN	TAMAÑO
Cabecera	Indicador de inicio de trama	1 byte
Longitud	Indica el número de bytes que aún faltan por llegar, sin incluir ni la cabecera ni a sí mismo	1 byte
ID Slave	Indica el Slave de destino u origen de la trama	1 byte
ID Frame	Indica el tipo de trama	1 byte
Datos	Campo de información	0 - 4 bytes
CKS	Check-sum de 8 bits para control de errores en recepción	1 byte

Tabla 2: campos de trama

3.3.2. Tipos de trama

3.3.2.1. Trama de configuración

Trama reservada para futuras necesidades. Actualmente está en desuso. La trama es enviada por Raspberry y su estructura se muestra en la Figura 7.



Figura 7: trama de configuración

3.3.2.2. Trama de solicitud de datos

Trama utilizada para solicitar al sensor los datos correspondientes a una medida. La trama es enviada por Raspberry y su estructura se muestra en la Figura 8.



Figura 8: trama de solicitud de datos

3.3.2.3. Trama de datos del sensor

Trama utilizada para el envío de información del sensor y que a su vez ya le sirve a Raspberry de ACK de la previa solicitud de datos. La trama es enviada por Arduino y su estructura se muestra en la Figura 9.

0xAA	0x07	ID SLAVE	0x02	DATOS SENSOR	CKS
------	------	----------	------	--------------	-----

Figura 9: trama de datos del sensor

El campo de datos consta de 4 bytes, los dos primeros son para la temperatura y dos siguientes para el % de humedad. El formato de los datos se especifica en el apartado 3.4.4.

3.3.2.4. Tramas de confirmación: ACK y NACK

La trama de ACK es utilizada para informar a Arduino de la correcta recepción de la trama de datos por parte de Raspberry. La trama es enviada por Raspberry y su estructura se muestra en la Figura 10. La trama de NACK es utilizada para informar a Arduino de la no recepción o recepción de la trama de datos con errores por parte de Raspberry. La trama es enviada por Raspberry y su estructura se muestra en la Figura 11.

0xAA	0x03	ID SLAVE	0x04	CKS
------	------	----------	------	-----

Figura 10: trama de ACK

0xAA	0x03	ID SLAVE	0x05	CKS
------	------	----------	------	-----

Figura 11: trama de NACK

3.3.3. Funcionamiento del protocolo

3.3.3.1. Detección de errores

Para la detección de errores se genera un check-sum de 8 bits aplicando sucesivamente el operador XOR a todos los bytes de la trama de salida (de la cabecera hasta los datos). Una vez en destino se recalcula a partir de toda la trama entrante (incluido el CKS). Como en la comprobación se incluye el byte de CKS recibido, el resultado que esperamos obtener es 0. De lo contrario significará que la trama ha llegado con errores.

Además se han añadido dos tipos de timeout: timeout interbyte y timeout de espera de respuesta.

- El **timeout interbyte** se ocupa de que el sistema nunca se quede esperando a acabar de recibir una trama que se ha recibido incompleta. Pasado el timeout se resetearían todas las variables y el sistema quedaría a la espera para recibir la siguiente trama. Este timeout es igual en ambos extremos del canal de comunicaciones.
- El **timeout de espera** de respuesta hace que el sistema nunca quede esperando una respuesta del otro extremo que no llega. Este timeout es ligeramente diferente en cada extremo aunque la finalidad es la misma.

3.3.3.2. Flujo del programa entre extremos del canal de comunicaciones

El diálogo básico de comunicación sin errores entre Master y Slave es el siguiente y lo podemos ver en la Figura 12:

1. Master envía trama de solicitud de datos.
2. Slave responde con la trama de datos del sensor.
3. Master confirma la correcta recepción de la trama de datos con un ACK.

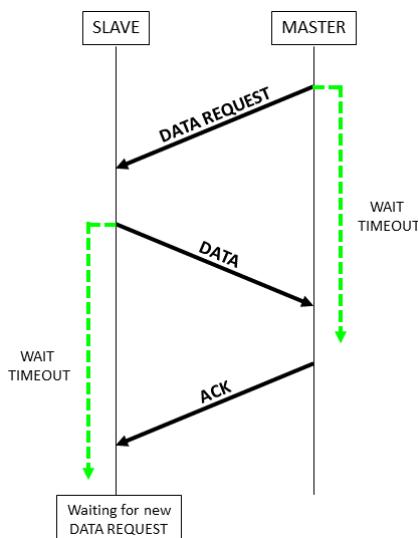


Figura 12: diálogo sin errores entre extremos del canal de comunicaciones

Lógicamente no siempre van a ser tan ideales las transmisiones, es por eso que se añadieron los timeouts que se han explicado en el apartado 3.3.3.1. Uno de los problemas que podrían ocurrir es que la trama de datos del sensor no llegue o llegue

con errores al Master. Si la trama de datos llegase con errores, el Master enviaría un NACK al Slave, al igual que si no se recibiese la trama de datos y saltase el timeout. El Slave al recibir un NACK contesta con la retransmisión de la última trama enviada. Ambos casos se muestran en la Figura 13.

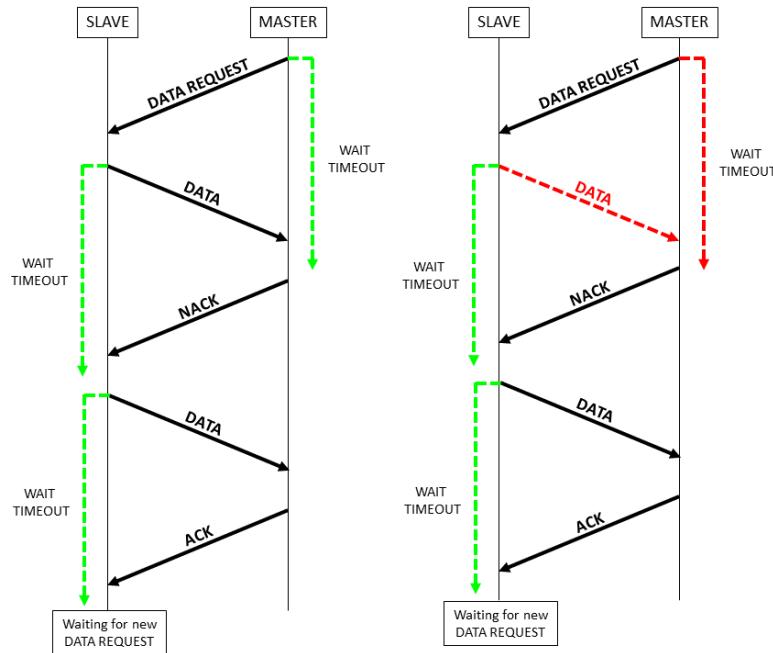


Figura 13: casos en que el Master contesta con NACK

De la misma manera podría ocurrir algo parecido en el extremo del Slave. Si el Slave no recibe contestación a la trama de datos enviada se produciría también la retransmisión de la última trama. Se puede apreciar esta interacción entre extremos en la Figura 14.

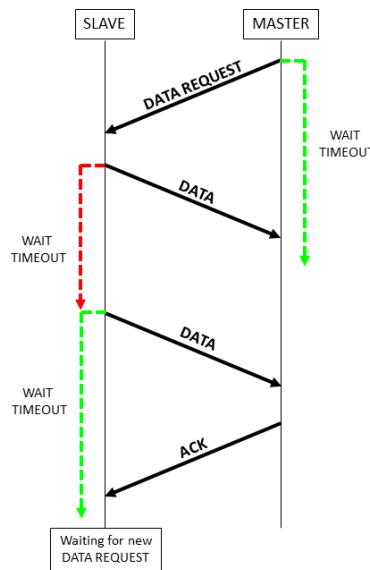


Figura 14: no recepción de trama de confirmación y reenvío de la trama anterior

Para acabar, vamos a explicar brevemente la política de retransmisiones del protocolo. Las retransmisiones de la trama de datos del sensor o las transmisiones de los NACK enviados por el MASTER después de saltar el timeout de espera están acotadas a un máximo de tres en ambos casos. En el momento que el protocolo lo solicite por cuarta vez, el extremo en cuestión reseteará su estado y esperará a realizar o recibir la próxima acción.

3.4. Bloque Arduino

3.4.1. Placa Arduino NANO

Las especificaciones necesarias de la placa Arduino NANO que vamos a utilizar son las siguientes:

- Puerto serie UART para comunicarse con Raspberry (por GPIO o USB).
- SPI o I2C para comunicarse con el LCD.
- Al menos un puerto GPIO para conectar con el sensor AM2302.
- Salida de +5V y GND para la alimentación del LCD y el sensor AM2302.

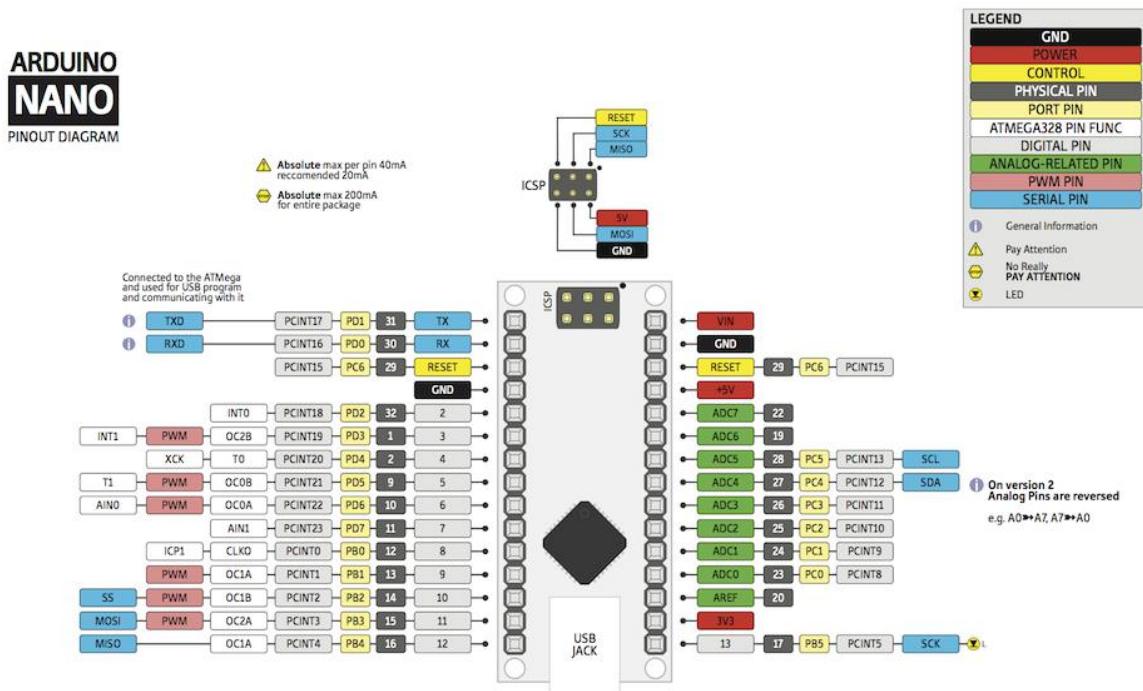


Figura 15: pinout Arduino NANO

En la Figura 15 podemos comprobar que cumple con todas ellas y por lo tanto la convierten en la candidata perfecta para este bloque.

3.4.2. Sensor AM2302

El sensor AM2302 se encarga de medir temperatura y % de humedad. Utiliza el bus de datos 1-Wire, que como su nombre indica está formado por un único canal bidireccional donde se solicita y se entrega la información recogida por el sensor. Las tramas de datos del sensor son de 40 bits: 16 bits de temperatura, 16 bits de RH y 8 bits de check-sum. En la Figura 16 podemos ver la estructura de la trama y como decodificarla.

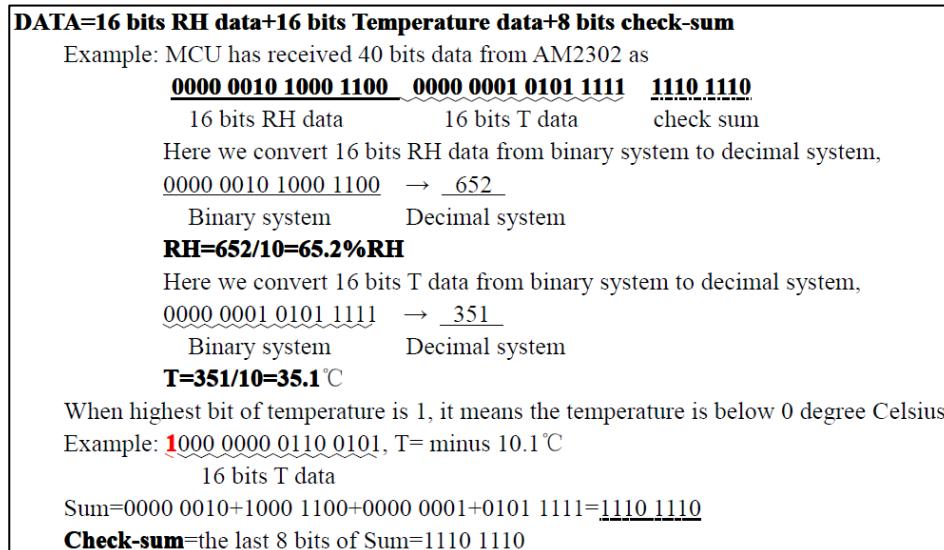


Figura 16: estructura y decodificación de trama en AM2302

Finalmente, vamos a explicar el diálogo entre Arduino y el sensor para la solicitud y obtención de datos. El diálogo consta de dos pasos:

1. Arduino envía señal de start a AM2302 y éste responde a Arduino. La Figura 17 recoge la interacción de este paso. En color oscuro vemos la señal enviada por Arduino y en color claro vemos la respuesta del sensor. Ambas señales por el mismo cable ya el sensor usa el protocolo 1-Wire.

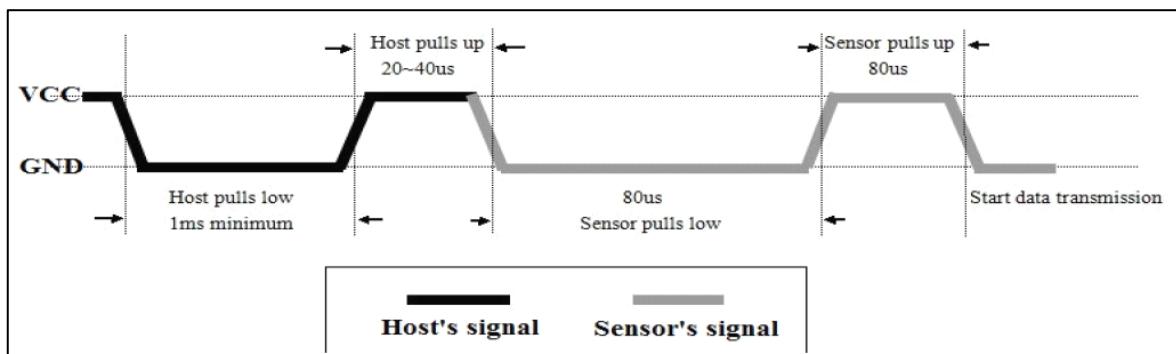


Figura 17: Arduino pide datos al sensor

- AM2302 envía la trama de datos a Arduino. La Figura 18 recoge la interacción de este paso. Vemos que después del tiempo de “Start transmit 1 bit data” envía un bit de la trama. Será un “0” o un “1” en función del tiempo que este el bus a nivel alto.

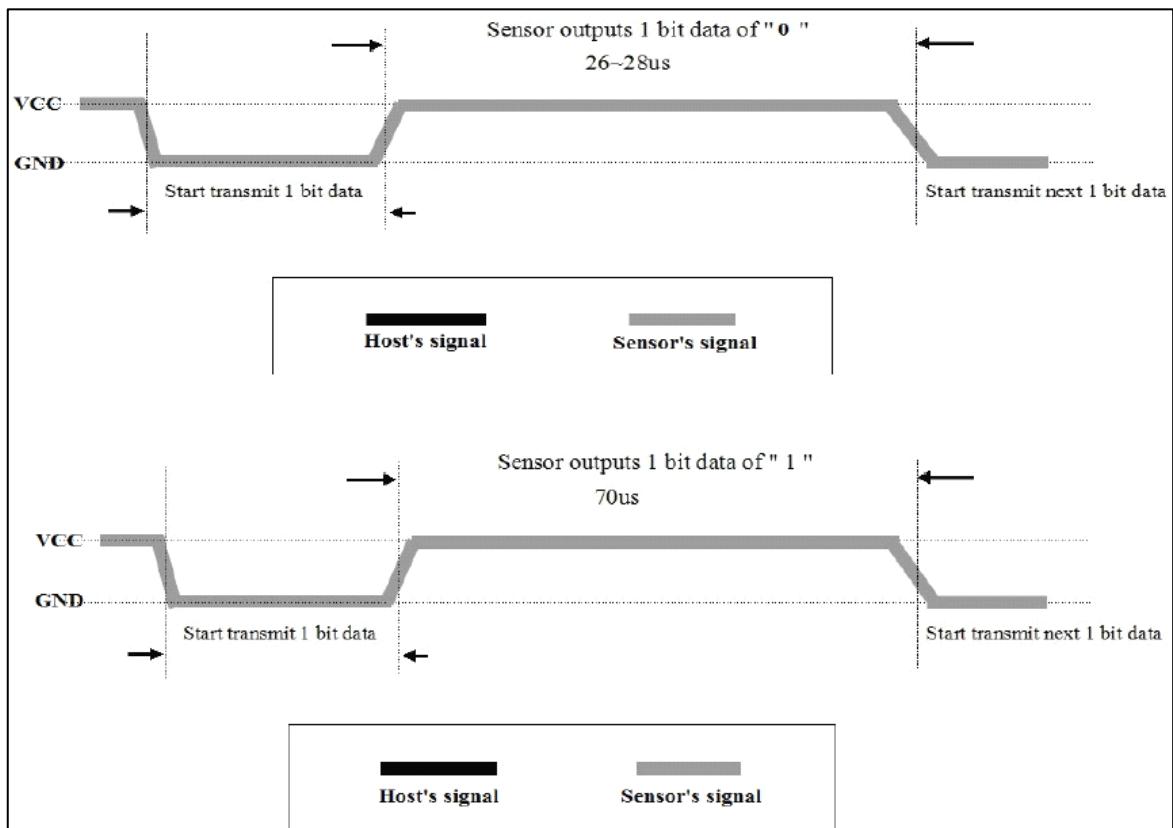


Figura 18: el sensor envía datos a Arduino

La tabla de especificaciones técnicas del sensor AM2302 la podemos encontrar en el Anexo XIV.

3.4.3. Software de comunicaciones alámbricas

Este bloque se encarga de la obtención y visualización de datos mediante el sensor AM2302 y el LCD 16x2 respectivamente. Además también se encarga de transmitir los datos recogidos por el sensor cuando Raspberry los solicita. El diagrama de bloques aparece en la Figura 19.

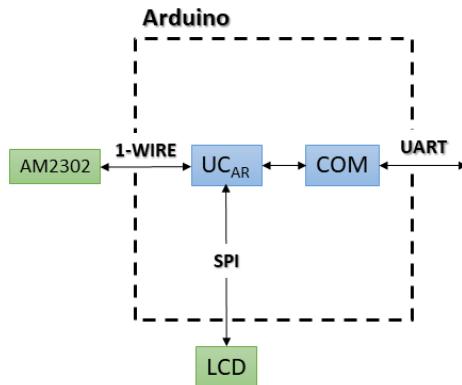


Figura 19: diagrama de bloques del bloque Arduino

El código está en el fichero **main_wired_arduino.ino** que podemos encontrar en el Anexo **¡Error! No se encuentra el origen de la referencia..**. El código sigue el diagrama de flujo que se muestra en la Figura 20.

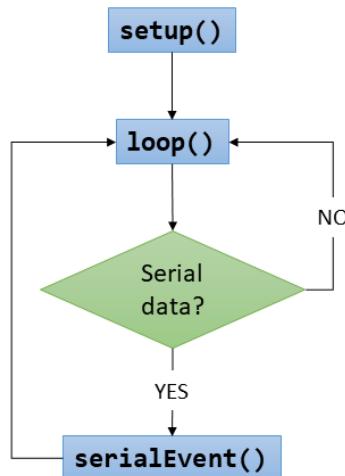


Figura 20: diagrama de flujo de **main_wired_arduino.ino**

El código implementa el protocolo de comunicaciones definido en el apartado 3.3 y consta de tres funciones principales:

- **setup()**: inicializa los valores de las variables, el LCD y el puerto serie.
- **loop()**: se ejecuta en bucle. Esta función realiza dos funciones principales. En primer lugar comprueba el timeout de espera, si ha saltado retransmite la última trama de datos, de acuerdo con la política de retransmisiones del apartado 3.3.3.2, y resetea el timeout de espera, sinó continua. En segundo lugar, y si ya dispone el sistema de una trama completa, se comprueba el CKS de la misma. Si es incorrecto se resetean todas las variables y se espera a la siguiente

petición del Master. Si los CKS coinciden se pasa a procesar la trama. Si se ha recibido una trama de solicitud de datos, se captura el dato del sensor, se envía por puerto serie y se inicia el timeout de espera. Si se ha recibido un ACK se para el timeout de espera, se resetean las variables y se espera a la siguiente petición del Master. Si se ha recibido un NACK se retransmite la trama anterior y se reinicia el timeout de espera. En la Figura 21 se aclara en formato gráfico el flujograma de la función **loop()**.

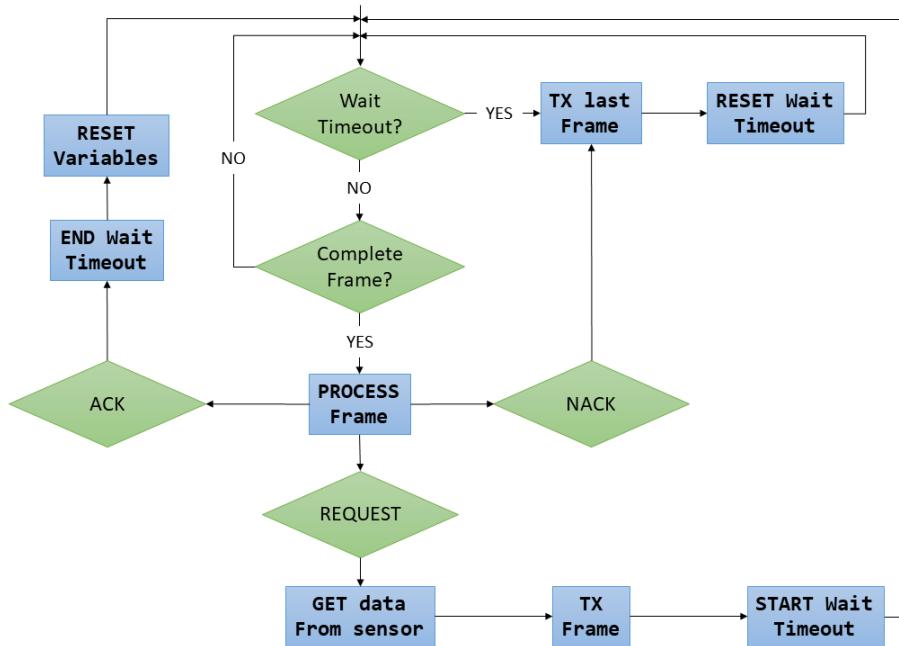


Figura 21: flujograma de la función **loop()**

- **serialEvent():** salta por interrupciones cada vez que hay un nuevo byte en el buffer de entrada del puerto serie (nunca interrumpe la ejecución de la función **loop()**). Se encarga de leer los bytes de la trama entrante uno a uno. Aquí se hacen ya algunas comprobaciones en la trama. Primero se comprueba el timeout interbyte, si ha saltado reseteamos todas las variables, sino continuamos. En segundo lugar, leemos el byte en cuestión y:
 - Si es el primer byte que recibimos con éxito, comprobamos que la cabecera coincide e iniciamos el timeout interbyte, sino reseteamos todas las variables y paramos el timeout.
 - Si es el segundo byte que recibimos con éxito, guardamos en una variable la longitud que esperamos recibir.

- Si es el tercer byte que recibimos con éxito, comprobamos que el frame es para mi y no otro Slave. Si no lo es reseteamos todas las variables y paramos el timeout.
- Cuando llegamos al último byte, indicamos al sistema que tenemos una trama completa, paramos el timeout y reseteamos todas las variables.

En la Figura 22 se aclara en formato gráfico el flujograma de la función `serialEvent()`.

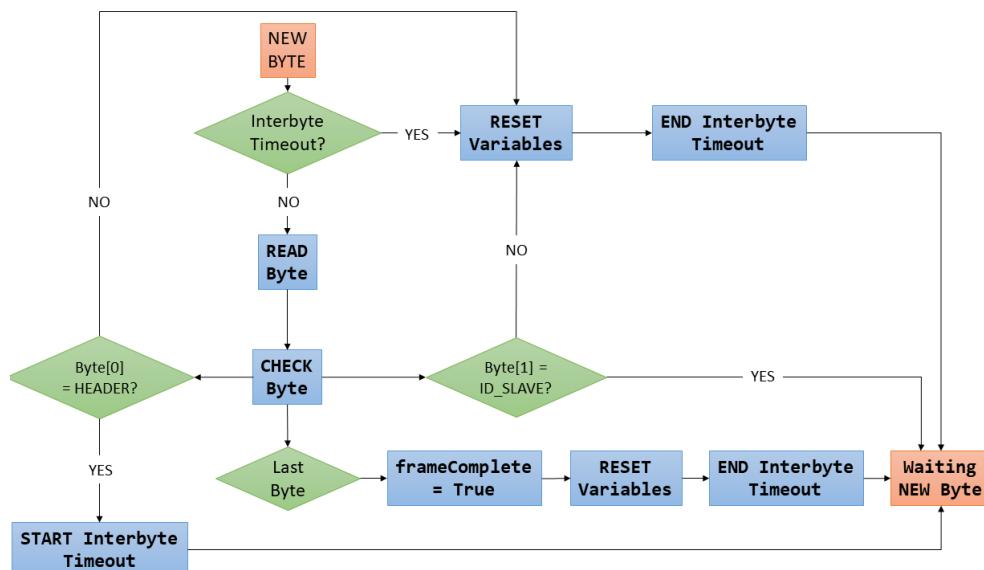


Figura 22: flujograma de la función `serialEvent()`

3.4.4. Adquisición y visualización de datos del sensor

El montaje experimental del bloque Arduino se muestra en la Figura 23.

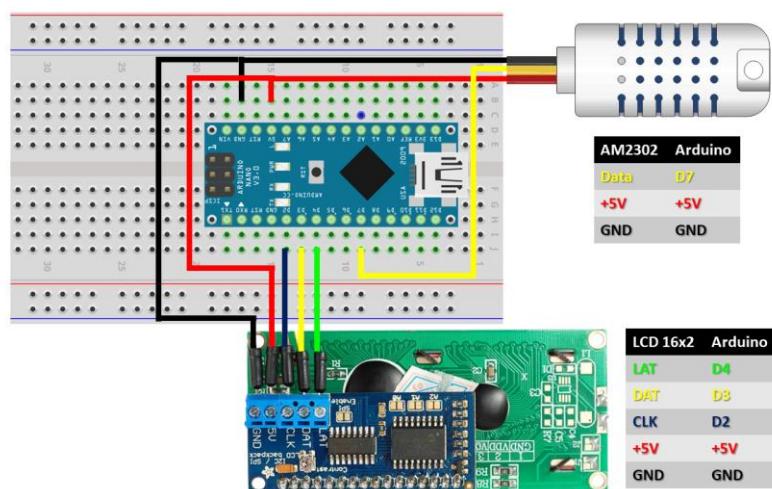


Figura 23: conexionado del sensor auxiliar

Cada vez que se solicite la adquisición de datos, el sensor pone en el bus 40 bits: 16 bits de RH, 16 bits de temperatura y 8 bits de CKS. La librería **DHT.h** nos permite obtener datos del sensor de manera fácil, pero proporciona al programa los datos en variables tipo **float** con solo un decimal significativo. Para evitar enviar 4 bytes por dato y modificar la librería que opera con el sensor, se ha optado por aplicar un factor x10 y almacenarlos en variables tipo **int**, de esta manera cada dato solo ocupará 2 bytes. Así hay menos bits que transferir por el canal.

Finalmente los datos son entregados al LCD mediante la librería **LiquidCrystal.h** para poder comprobar de manera visual la veracidad de los datos recibidos en el otro extremo del canal. El LCD muestra los datos en el formato que aparece en la Figura 24.



Figura 24: formato de los datos mostrados en el LCD

3.5. Bloque Raspberry

Este bloque se encarga de realizar dos principales funciones:

1. Comunicaciones alámbricas: recibir los datos por UART y almacenarlos en un fichero temporalmente.
2. Comunicaciones inalámbricas: conectarse con la APP Android y ejecutar las peticiones listadas en el apartado 3.2.

El diagrama de bloques que se ha implementado queda reflejado en la Figura 25.

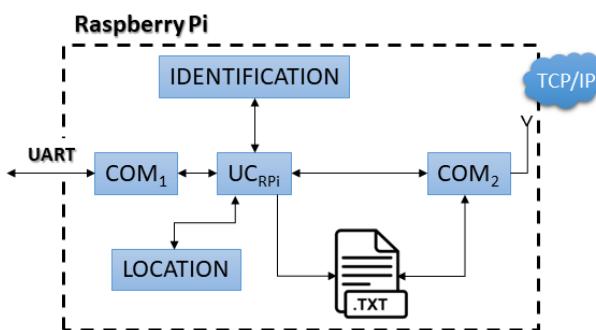


Figura 25: diagrama de bloques del bloque Raspberry

3.5.1. Placa Raspberry Pi Zero W

Las especificaciones necesarias de la placa Raspberry Pi Zero W que vamos a utilizar son las siguientes:

- Puerto serie UART para comunicarse con Arduino (por GPIO o USB).
- Comunicaciones Wi-Fi para conectarse con la APP.
- Al menos un puerto GPIO y el GND para conectar el LED localizador.
- Memoria de almacenamiento.

En la Figura 26 podemos ver los puertos de entrada salida de la cabecera GPIO de la placa.

Raspberry Pi Zero W GPIO Header	
Pin#	NAME
01	3.3v DC Power
03	GPIO02 (SDA1 , I ² C)
05	GPIO03 (SCL1 , I ² C)
07	GPIO04 (GPIO_GCLK)
09	Ground
11	GPIO17 (GPIO_GEN0)
13	GPIO27 (GPIO_GEN2)
15	GPIO22 (GPIO_GEN3)
17	3.3v DC Power
19	GPIO10 (SPI_MOSI)
21	GPIO09 (SPI_MISO)
23	GPIO11 (SPI_CLK)
25	Ground
27	ID_SD (I ² C ID EEPROM)
29	GPIO05
31	GPIO06
33	GPIO13
35	GPIO19
37	GPIO26
39	Ground
(TXD0) GPIO14	
(RXD0) GPIO15	
(GPIO_GEN1) GPIO18	
Ground	
(GPIO_GEN4) GPIO23	
(GPIO_GEN5) GPIO24	
Ground	
(GPIO_GEN6) GPIO25	
(SPI_CE0_N) GPIO08	
(SPI_CE1_N) GPIO07	
(I ² C ID EEPROM) ID_SC	
Ground	
GPIO12	
Ground	
GPIO16	
GPIO20	
Ground	
GPIO21	

Figura 26: pinout de la placa Raspberry Pi Zero W

Como hemos mencionado en el apartado 3.5, el bloque debe almacenar temporalmente los datos. En Raspberry la memoria interna depende de la tarjeta SD en la que grabes el Sistema Operativo. En nuestro caso hemos usado la última versión de Raspbian con escritorio en una µSD de 8GB. Nos quedan libres aproximadamente 3,5 - 4GB para almacenamiento de datos. Para hacernos una idea de si va a ser suficiente, vamos a hacer unos cálculos, sabiendo de antemano que un fichero que contiene 1000 medidas ocupa aproximadamente unos 75KB.

$$(1) \quad 3,5 \text{ GB} \cdot \frac{1024 \text{ MB}}{1 \text{ GB}} \cdot \frac{1024 \text{ KB}}{1 \text{ MB}} \cdot \frac{1000 \text{ medidas}}{75 \text{ KB}} \cong 49M \text{ de medidas}$$

$$(2) \quad 365 \text{ días} \cdot \frac{24 \text{ h}}{1 \text{ día}} \cdot \frac{60 \text{ min}}{1 \text{ h}} \cdot \frac{1 \text{ medida}}{1 \text{ min}} = 525.600 \text{ medidas}$$

En (1) hemos calculado la cantidad de medidas que cabrían en 3,5GB y en (2) hemos calculado la cantidad de medidas que generaría el sistema en un año. Se puede observar claramente que el espacio de almacenamiento no resulta ser un problema.

3.5.2. Software de comunicaciones alámbricas

En este apartado vamos a explicar más en detalle el software diseñado para la parte de comunicaciones alámbricas del bloque. Al igual que en Arduino, el código implementa el protocolo de comunicaciones definido en el apartado 3.3. Para dar robustez al código y poder manejar de forma agradable los objetos que almacenan temporalmente las tramas, se ha decidido separar el código en dos partes: el script `main_wired_rpi.py` y la clase `myFrame.py`, adjuntados en los Anexos II y III respectivamente.

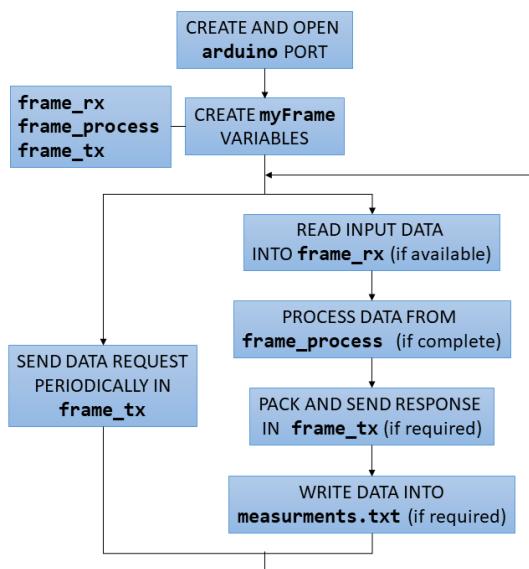


Figura 27: flujo de funcionamiento de `main_wired_rpi.py`

En el flujograma de la Figura 27 se puede observar que el funcionamiento es similar al del software implementado en Arduino. Salvo que Raspberry al ser la placa MASTER

se encarga de solicitar periódicamente los datos a Arduino (SLAVE). Además también almacena los datos que va recibiendo en un fichero.

En la clase `myFrame.py` se han implementado todos los métodos y las variables necesarias para nuestras necesidades. Los métodos y variables se listan en la Tabla 3.

MÉTODO	DESCRIPCIÓN	VARIABLES
<code>__init__(size)</code>	Inicializa el objeto <code>frame</code> al tamaño <code>size</code> con todos los elementos con valor 0	<code>frame</code> : almacena la trama
<code>__len__()</code>	Devuelve el número de elementos del objeto <code>frame</code>	<code>send_flag</code> : indica que hay una trama por enviar y de que trama se trata
<code>copyFrame(dest_frame)</code>	Copia ítem a ítem el objeto <code>frame</code> en el objeto <code>dest_frame</code>	
<code>getFrame()</code>	Devuelve el objeto <code>frame</code>	<code>frameComplete</code> : indica si tenemos un frame completo
<code>putItem(itemVal, index)</code>	Fija el valor <code>itemVal</code> en la posición <code>index</code> del objeto <code>frame</code>	
<code>getItem(index)</code>	Devuelve el ítem de la posición <code>index</code> del objeto <code>frame</code>	<code>write_in_file</code> : indica si es necesario escribir una medida en el fichero de medidas
<code>getTemp()</code>	Devuelve la temperatura registrada en la variable <code>temp</code>	<code>frameLen</code> : almacena la longitud del frame entrante
<code>getHum()</code>	Devuelve el % de humedad registrada en la variable <code>hum</code>	
<code>setTemp(t)</code>	Fija el valor <code>t</code> en la variable <code>temp</code>	<code>temp, hum</code> : almacena los datos recibidos
<code>setHum(h)</code>	Fija el valor <code>h</code> en la variable <code>hum</code>	
<code>restartFrame()</code>	Fija a 0 el valor de todos los ítems del objeto <code>frame</code>	<code>bytes_rx</code> : contabiliza el número de bytes leídos del puerto
<code>printFrame()</code>	Muestra por línea de comandos todos los ítems del objeto <code>frame</code>	
<code>computeCKS()</code>	Devuelve el CKS calculado a partir de los ítems del objeto <code>frame</code>	<code>nack_rtx</code> : contabiliza el número de NACK transmitidos en caso de saltar el timeout de espera
<code>packFrame(type)</code>	Fija el valor de los ítems del objeto <code>frame</code> en función del <code>type</code> de trama	<code>escapeTimeFrame</code> : fija el tiempo máximo del timeout interbyte
<code>readIntoFrame(serial)</code>	Si hay un byte disponible en el puerto <code>serial</code> definido, lo lee y lo almacena en el objeto <code>frame</code>	<code>escapeWaitMeasure</code> : fija el tiempo máximo del timeout de espera
<code>processFrame(frame_rx)</code>	Procesa <code>frame_rx</code> y actúa en consecuencia	

Tabla 3: métodos y variables de la clase `myFrame.py`

Por último, vamos a describir el funcionamiento detallado de los dos métodos principales: `readIntoFrame(serial)` y `processFrame(frame_rx)`.

- `readIntoFrame(serial)`: funciona exactamente igual que la función `serialEvent()` de Arduino explicada en el apartado 3.4.3 salvo que el MASTER no comprueba el id_slave de la trama. En este caso, si salta el timeout interframe, fuerzo al sistema que envíe un NACK al SLAVE poniendo el flag `frameComplete` a `True`. El flujo gráfico resultante se muestra en la Figura 28.

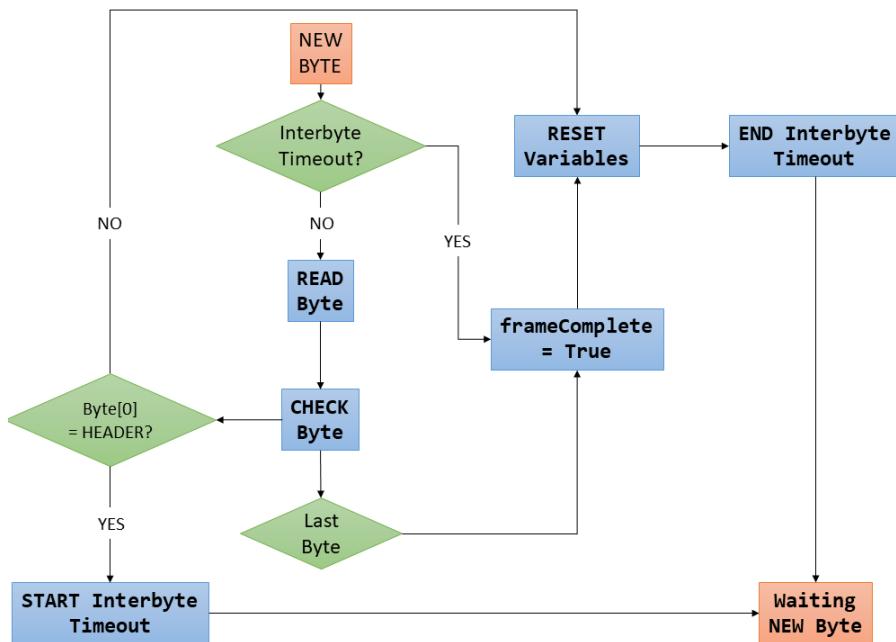


Figura 28: flujo gráfico de la función `readIntoFrame()`

- `processFrame(frame_rx)`: esta función sería la equivalente al `loop()` de Arduino explicada en el apartado 3.4.3. En primer lugar comprueba el timeout de espera, si ha saltado transmite un NACK, de acuerdo con la política de retransmisiones del apartado 3.3.3.2, y resetea el timeout de espera, sino continua. En segundo lugar, y si ya dispone el sistema de una trama completa, se comprueba el CKS de la misma. Si es incorrecto se resetean todas las variables y se transmite un NACK. Si los CKS coinciden se pasa a procesar la trama. En el caso del MASTER sólo se realizan acciones si la trama es una trama de datos. En este caso, se guardarán en las variables correspondientes los datos recibidos, se indicará al script principal (`main_wired_rpi.py`) que deben ser escritos en el fichero, que se debe enviar un ACK, y por último se resetearán todas las variables pertinentes. El flujo gráfico de esta función se relata en la Figura 29.

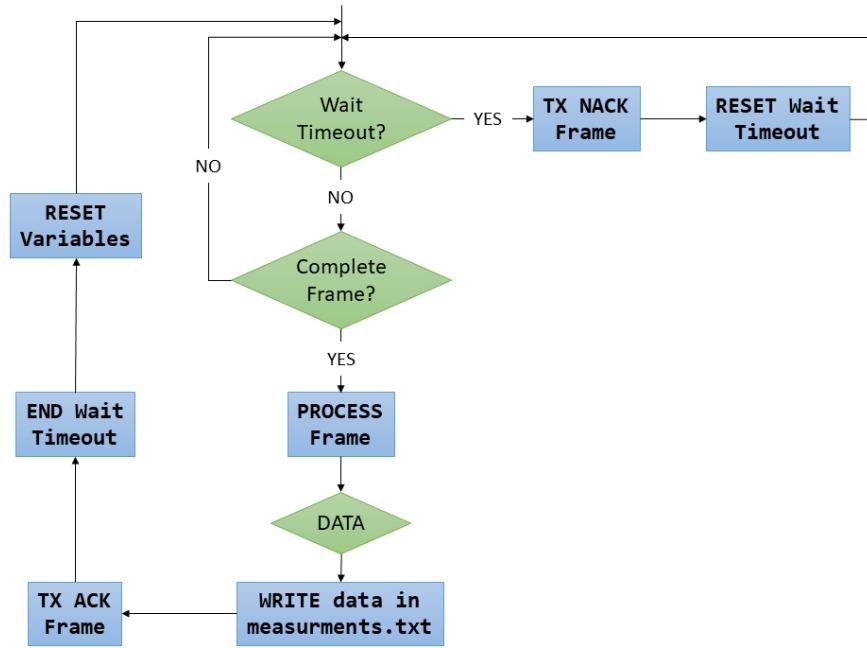


Figura 29: flujograma de la función `processFrame()`

3.5.3. Software de comunicaciones inalámbricas

Se ha diseñado el software en base a una arquitectura Servidor – Cliente, como se ilustra en la Figura 30. Este tipo de arquitecturas se encargan de relacionar y repartir las tareas entre dos procesos que se ejecutan en máquinas separadas. En nuestro caso, el cliente (consumidor de servicios) será la APP Android y el servidor (proveedor de servicios) será el bloque Raspberry en conjunto con el bloque Arduino.

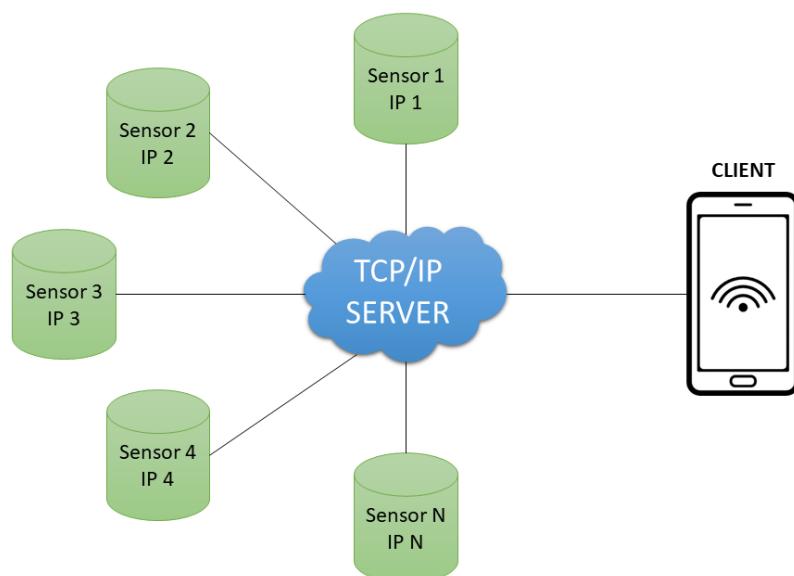


Figura 30: arquitectura servidor - cliente

Como decíamos, el script `main_wireless_rpi.py` está diseñado en base a una arquitectura Servidor – Cliente. El código está adjuntado en el Anexo **¡Error! No se encuentra el origen de la referencia..** La conexión entre extremos se establece por Wi-Fi a través de sockets. En nuestro caso, como se desea poder administrar múltiples sensores (servidores) con una única APP, es el propio Smartphone el que genera un punto de acceso al que se conectar los diferentes sensores a elección del usuario. Para que Raspberry se conecte de manera automática al punto de acceso es necesario añadir las credenciales de la red de la siguiente forma:

1. Ejecutar `sudo nano /etc/wpa_supplicant/wpa_supplicant.conf`
2. Añadir al final del fichero el texto mostrado en la Figura 31.

```
network={  
    ssid="Test Wifi Network"  
    psk="SecretPassword"  
}
```

Figura 31: configuración de red Wi-Fi en Raspberry

Raspberry se mantiene a la espera hasta que puede conectarse a la red que le hemos prefijado. Una vez conectada, espera recibir peticiones por parte de la APP (Cliente), hasta entonces espera. Hay múltiples tareas que se ejecutan desde el servidor a petición del cliente, todas ellas funcionan en base al flujo de la Figura 32. Son tareas independientes para cada sensor y son las siguientes:

1. Asignación de ID al establecer la conexión.
2. Parpadeo del diodo LED para poder localizarlo visualmente en la planta.
3. Inicio/Parada del proceso de medidas.
4. Transferencia de datos almacenados.

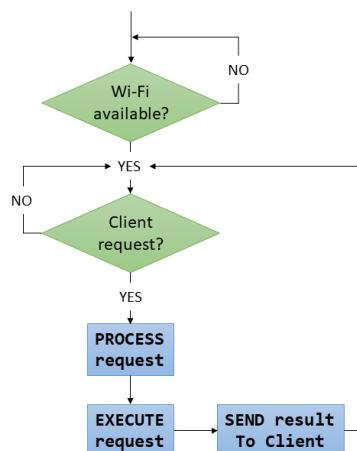


Figura 32: flujo de trabajo del servidor

3.5.3.1. Intercambio inicial de mensajes para la identificación del sistema

Cuando se establece conexión entre APP y sensor, la APP siempre entrega un ID al sensor, que si no tiene ya el ID fijado, almacena y escribe en el fichero **id.txt**. El fichero únicamente contiene el valor del ID del sensor en formato texto. De este modo, si se interrumpiese la alimentación de Raspberry, al arrancar el sistema mantendrá su ID. Una vez fijado el ID, no procesa el ID que le intenta dar la APP cada vez que se conecta pero si responde a la APP informando de si está o no realizando el proceso de medidas.

3.5.3.2. Localización del sistema en la planta

El objetivo de poder localizar el sensor de manera visual mediante el parpadeo de un LED es debido a que el usuario puede haber instalado el sensor pero no haberlo registrado en la APP. Es por eso que, si en el momento de registrar un sensor no tenemos manera de saber cuál es, se ha añadido una función parpadeo. Para esta función simplemente hemos conectado un LED al pin 12 de la GPIO de la placa, como podemos contrastar entre la Figura 26 y Figura 33. Cuando el usuario solicita localizar el sensor desde la APP, el LED se pone a parpadear a una cadencia de 0,5 segundos. A nivel de software, la tarea se reduce a hacer un toggle en el puerto asociado al polo positivo del LED.

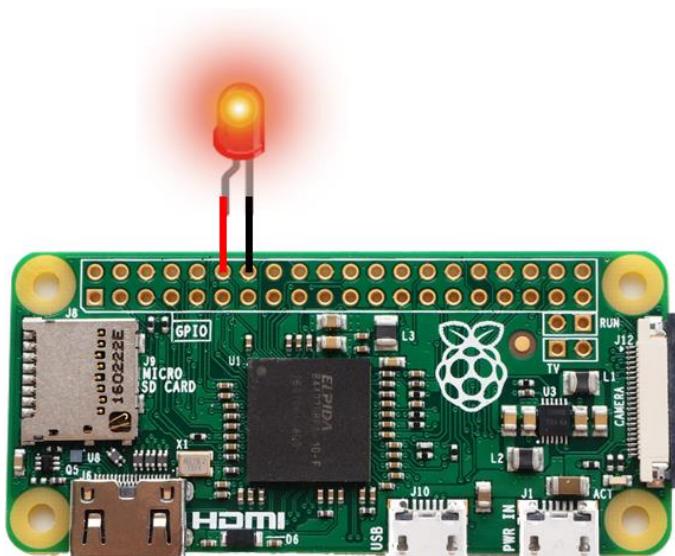


Figura 33: montaje LED de localización

3.5.3.3. Inicio/Parada del proceso de mediciones

Como hemos dicho en apartados anteriores, el bloque Arduino debe ir tomando medidas periódicamente. Pero no necesariamente tiene que iniciar el proceso de medidas desde el primer momento. Es por eso que se ha decidido dejar que el usuario elija cuando iniciar y parar el proceso de manera independiente a través de la APP. Para esta tarea se utiliza la clase **subprocess** de Python, que nos permite comenzar o terminar de ejecutar otro script. Cuando el usuario solicita iniciar o parar el proceso, el script `main_wireless_rpi.py` inicia o para automáticamente el script `main_wired_rpi.py`.

3.5.3.4. Transferencia de datos vía socket

Llegados a este punto estamos delante de la tarea más importante de esta parte del sensor: el envío de los datos almacenados. Como se muestra en la Figura 27, los datos se almacenan en el fichero `measurements.txt`. Cuando el usuario solicita la descarga de información desde la APP el script `main_wireless_rpi.py` realiza las siguientes acciones:

1. Copiar el contenido de `measurements.txt` en `temp.txt`.
2. Vaciar el fichero `measurements.txt` para no enviar medidas repetidas en las siguientes solicitudes y además poder seguir escribiendo medidas.
3. Notificar a la APP del número de líneas que contiene `temp.txt`.
4. Transferir línea a línea los datos almacenados.
5. Eliminar el fichero `temp.txt`.

En este caso no ha sido necesario implementar mecanismos de detección y control de errores ya que las comunicaciones vía socket utilizan el protocolo TCP/IP y es éste el que se encarga de gestionarlo internamente.

3.6. Bloque Android APP

3.6.1. Fundamentos y pasos iniciales

Antes de profundizar en la propia APP, vamos a introducir algunos fundamentos del diseño de aplicaciones Android. Una aplicación se compone al menos de tres ficheros: **MainActivity.java**, **activity_main.xml** y **AndroidManifest.xml**. En el primero se define la lógica de la APP y en el segundo se define la interfaz gráfica. Cada par adicional que se añada al proyecto supondrá una vista más a nuestra aplicación. El último resulta ser el fichero de configuración de la APP. En él aplicaremos las configuraciones básicas de la aplicación y declararemos todas las vistas de las que conste nuestra aplicación.

Cada una de las vistas de la aplicación tiene un ciclo de vida que viene definido por Android. En la Figura 34 podemos ver el ciclo y las funciones que lo componen. Como es lógico pensar, cada desarrollador puede sobreescrivir (Override) el contenido de dichas funciones predefinidas. Es importante tener en cuenta que las variables que se usen en el código pueden no tener los valores esperados en ciertos momentos ya que pueden reiniciarse o no actualizarse debido a la navegación entre Activity's y a sus respectivos ciclos de vida.

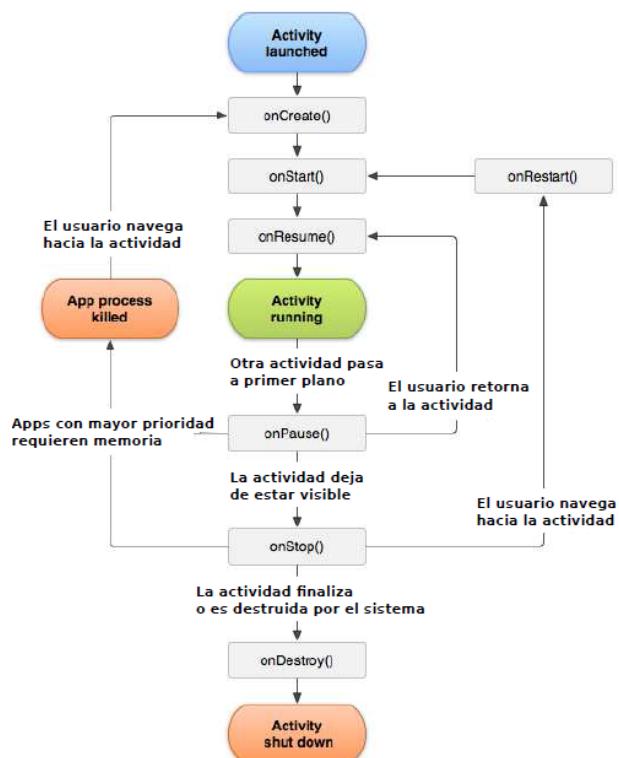


Figura 34: ciclo de vida de un Activity

Para que la APP pueda funcionar conjuntamente con el resto del sistema será necesario configurar el Punto de Acceso de nuestro dispositivo con las mismas credenciales que hemos prefijado en Raspberry, como en el ejemplo de la Figura 31.

A efectos prácticos este es el único bloque con el que va a interactuar el usuario final. En él podrá realizar la petición de todas las tareas explicadas en el apartado 3.5.3 además de las acciones específicas que actúan sobre la base de datos y la presentación gráfica de las medidas almacenadas. Podemos observar el diagrama de bloques en la Figura 35.

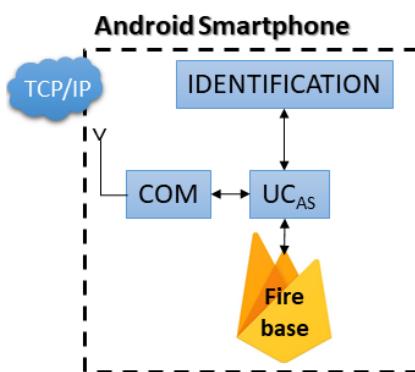


Figura 35: diagrama de bloques del bloque Android APP

3.6.2. ¿Por qué Firebase?

En este apartado vamos a entrar en detalle de cuáles son los aspectos que nos han hecho decantarnos por Firebase, las principales características y funcionalidades de Firebase las encontramos en <https://firebase.google.com/docs/database/>. Vamos a hacer una lista con las principales características que eran fundamentales para la realización de este proyecto:

- Está alojada en la nube y permite almacenar y sincronizar datos entre usuarios en tiempo real.
- Los usuarios pueden acceder a sus datos desde cualquier dispositivo vía web.
- Se incluye en los SDK para dispositivos móviles y web, de manera que puedes crear APPs sin necesidad de usar servidores.
- Si el usuario se desconecta, los SDK de Realtime Database usan la caché local del dispositivo para publicar y almacenar cambios. Cuando el dispositivo vuelve a conectarse, los datos locales se sincronizan de manera automática.

- Integra además Firebase Authentication para brindar autenticación intuitiva y sencilla.

3.6.3. Adecuación del entorno para trabajar con Firebase

Para poder trabajar con Firebase es necesario seguir un proceso de configuración del proyecto en Android Studio. En primer lugar, hay que registrar nuestra APP en <https://console.firebaseio.google.com>, descargar el fichero `google-services.json` y copiarlo a la carpeta local del módulo del proyecto.

Finalmente es necesario añadir a los ficheros `Module/build.gradle` y `Project/build.gradle` las partes destacadas de la Figura 36 y la Figura 37.

```
buildscript {  
    // ...  
    dependencies {  
        // ...  
        classpath 'com.google.gms:google-services:3.2.0' // google-services plugin  
    }  
  
    allprojects {  
        // ...  
        repositories {  
            // ...  
            maven {  
                url "https://maven.google.com" // Google's Maven repository  
            }  
        }  
    }  
}
```

Figura 36: plantilla Project/bulid.gradle

```
apply plugin: 'com.android.application'  
  
android {  
    // ...  
}  
  
dependencies {  
    // ...  
    compile 'com.google.firebaseio.firebaseio:firebase-database:11.8.0'  
  
    // Getting a "Could not find" error? Make sure you have  
    // added the Google maven respository to your root build.gradle  
}  
  
// ADD THIS AT THE BOTTOM  
apply plugin: 'com.google.gms.google-services'
```

Figura 37: plantilla Module/bulid.gradle

3.6.4. Funciones y características principales de la APP

En este apartado vamos a explicar las funciones principales y como la APP gestiona las diferentes peticiones. No vamos a entrar en como es la interfaz gráfica ni como se gestionan las diferentes vistas de la APP.

Los sensores se conectan vía Wi-Fi con el Smartphone, por lo tanto, cada uno de ellos representará una dirección IP en el sistema. Para que la aplicación nos permita conectarnos al sensor que nosotros escojamos, se ha implementado un escáner de direcciones IP. Para ello hemos hecho uso de la clase **AsyncTask** de JAVA. Esta clase se encarga de repartir sus tareas entre el hilo principal de la aplicación y un hilo en segundo plano. La implementación de esta funcionalidad se encuentra en el Anexo VI. En la Figura 38 se muestra el ciclo de vida de la clase y se especifican las funciones que utiliza.

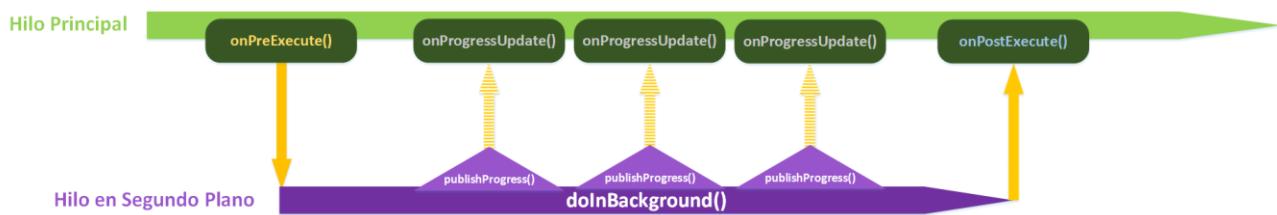


Figura 38: ciclo de vida de la clase *AsyncTask*

En nuestro caso concreto, las tareas a realizar por cada una de estas funciones son:

- **onPreExecute()**: pone a punto los elementos gráficos de la vista.
- **doInBackground()**: intenta conectarse recursivamente a todas las posibles direcciones IP de dentro de su red. Si tiene éxito realiza un **publishProgress("dirIP")**.
- **onProgressUpdate()**: se ejecuta cuando desde **doInBackground()** se realiza un **publishProgress**. Se encarga de añadir la dirección IP en cuestión a una lista de elementos seleccionables.
- **onPostExecute()**: resetea todos los elementos gráficos de la vista salvo la lista de IP's.

Una vez seleccionado un elemento de la lista, la APP accederá al Activity donde se realizan las peticiones al Servidor. Todas las peticiones se ejecutan en hilos separados utilizando la clase **Thread** de JAVA. Mientras no se destruya el Activity que contiene los diferentes Threads se estarán ejecutando acorde al flujo de la Figura 39.

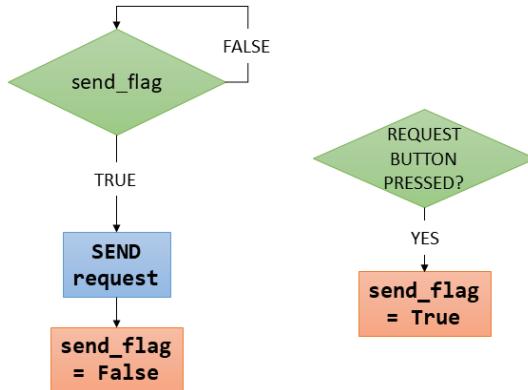


Figura 39: flujograma genérico de una petición

En concreto hablaremos de cuatro peticiones: localización, inicio/parada de medidiones, eliminación del sensor y descarga de datos (todas ellas en el Anexo IX). Las tres primeras peticiones no realizan ninguna función especial, simplemente envían la petición y es el Servidor que al procesarla realiza la acción pertinente:

- Si se le pide localizar, el Servidor pone en marcha el LED.
- Si se le pide inicio/parada de medidas, el Servidor inicia o para el script `main_wired_rpi.py`.
- Si se le pide eliminar el sensor, el Servidor resetearía el valor del ID y también pararía el script `main_wired_rpi.py` en caso de que se estuviera ejecutando.

La petición de descarga de datos es un caso especial y la función que realiza se encuentra repartida en dos Threads diferentes. El primer Thread se comporta como los que hemos mencionado antes, y como característica adicional, en este caso recibe respuesta del Servidor: el número de líneas del fichero que se dispone a recibir. Una vez recibido el número de líneas activa un flag que permite al segundo Thread recibir todas y cada una de las líneas del fichero. Una vez acabada la recepción de datos ambos Threads actualizan sus flags para volver al estado inicial.

Para acabar, hay una serie de variables que gracias a la clase **SharedPreferences** de JAVA nos permiten que su valor quede guardado en la memoria del dispositivo para así poder cargarlo al iniciar la aplicación. Necesitábamos que ciertas informaciones acerca de cada sensor quedasen almacenadas para garantizar el correcto funcionamiento de la aplicación y la gestión de esta de todos los sensores. Estas informaciones se han almacenado en objetos del tipo **HashMap** que se componen de tuplas clave – valor.

Los objetos que almacenamos son los siguientes:

- `HashMap<String, Integer>` **sensors** para almacenar pares `ipaddress – id_sensor`. Cuando se conecta al sensor por primera vez se almacena el par de valores.
- `HashMap<String, Boolean>` **regLog** para almacenar pares `ipaddress – flag_sensorRegistrado`. Cuando se conecta al sensor por primera vez se almacena el par de valores. El flag indica si está registrado.
- `HashMap<String, Boolean>` **regID** para almacenar pares `id – flag_id_utilizado`. Gestiona los ID que la aplicación va entregando a los sensores que se conectan. El flag indica si valor `id` está en uso.
- `HashMap<String, String>` **regNAME** para almacenar pares `ipaddress – nombre_sensor`. Cuando se registra el sensor se almacena el par de valores. Se utiliza para indicar el nombre del sensor en la lista de IP's.

Como es lógico pensar, cuando el sensor es borrado los pares de valores asociados a él de cada una de las listas anteriores son eliminados.

3.6.5. Manejo de Firebase desde la APP

En este apartado nos vamos a centrar exclusivamente en el manejo que hace la aplicación sobre Firebase. En primer lugar, vamos a introducir cuál es la estructura de datos que utiliza. Los datos están estructurados de forma similar a cómo se organiza la información en un fichero JSON, es decir, en forma de árbol donde cada nodo puede contener un valor o bien contener nodos hijo, que a su vez podrán contener valores o tener nuevos nodos hijo, y así sucesivamente hasta un máximo de 32 niveles de profundidad. Por otro lado la capacidad de almacenamiento es de 1GB en su versión gratuita, pero lo discutiremos más adelante.

Como comentábamos en el apartado 3.6.2, el soporte a Firebase está incluido en los SDK de Android. Esto va a hacer que la subida y descarga de datos de Firebase se pueda manejar fácilmente gracias a las clases **FirebaseDatabase** y **DatabaseReference** de JAVA. Con la primera obtendremos la instancia a la base de datos y con la segunda obtendremos la referencia para poder acceder a ella, ya sea para cargar o descargar datos. Para subir datos disponemos de los métodos explicados en la Figura 40.

set	Escribe o reemplaza datos en una ruta de acceso definida, como <code>messages/users/<username></code>
update	Actualiza algunas de las claves de una ruta de acceso definida sin reemplazar todos los datos
push	Agrega a una lista de datos en la base de datos. Cada vez que envías un nodo nuevo a una lista, tu base de datos genera una clave única, como por ejemplo, <code>messages/users/<unique-user-id>/<username></code>
transaction	Usa transacciones cuando trabajes con datos complejos que podrían dañarse con las actualizaciones simultáneas

Figura 40: guardado de datos en Firebase

Para obtener datos disponemos de Receptores Asíncronos, que no son más que interfaces de escucha de eventos que asociaremos a la referencia deseada de la base de datos. Concretamente se utiliza la interfaz `ChildEventListener`. Esta interfaz implementa cinco métodos que se explican en la Figura 41.

<code>onChildAdded()</code>	Recupera listas de elementos o detecta elementos agregados a una lista. Esta devolución de llamada se activa una vez por cada elemento secundario existente y otra vez cuando se agrega un elemento secundario nuevo a la ruta de acceso especificada. La <code>DataSnapshot</code> pasada al agente de escucha contiene los datos del nuevo elemento secundario.
<code>onChildChanged()</code>	Detecta cambios en los elementos de una lista. Este evento se activa cada vez que un nodo secundario se modifica, incluidas todas las modificaciones en los descendientes del nodo secundario. La <code>DataSnapshot</code> pasada al agente de escucha de eventos contiene los datos actualizados del elemento secundario.
<code>onChildRemoved()</code>	Detecta cuando se quitan elementos de una lista. La <code>DataSnapshot</code> pasada a la devolución de llamada del evento contiene los datos del elemento secundario eliminado.
<code>onChildMoved()</code>	Detecta cambios en el orden de los elementos de una lista ordenada. Este evento se activa cada vez que una actualización activa la devolución de llamada <code>onChildChanged()</code> que cambia el orden del elemento secundario. Se usa con datos que se ordenan con <code>orderByChild</code> o <code>orderByValue</code> .

Figura 41: métodos interfaz `ChildEventListener`

En nuestra aplicación existen cinco tareas que actúan sobre Firebase: descarga de datos (Anexo IX), registro de sensor (Anexo X), presentación gráfica de datos (Anexo XI), eliminación de datos (Anexo IX) y eliminación de sensor (Anexo IX). En el primero caso, utilizamos la función `push()` explicada en la Figura 40 para subir los datos. A medida que se van recibiendo las líneas del fichero de datos se van escribiendo en la base de datos, temperaturas en un nodo y humedades en otro. De la misma manera, pero en este caso con la función `set()` también explicada en la Figura 40, en el

momento del registro se añaden las referencias siguientes a la base de datos: ID, dirección IP, nombre del vino, referencia del vino y fecha de creación. En la Figura 42 se puede ver como queda Firebase después de registrar el sensor “tinto”.

```

https://wineapp-edb91.firebaseio.com/
+ Sensors
  + Sensor_1
    + Creation: "Fri Apr 27 16:22:46 GMT+02:00 2020"
    + ID: "1"
    + IP address: "192.168.43.183"
    + Name: "tinto"
    + Reference: "33"
  
```

Figura 42: estado de Firebase después del registro del sensor "tinto"

Para la presentación gráfica de los datos es donde entra en juego la interficie **ChildEventListener**. Utilizamos el método **onChildAdded()** para obtener todos los elementos de los nodos “Temperaturas” y “Humedades” y así almacenarlos en listas. Una vez tenemos la lista de temperaturas y humedades hacemos uso de la clase **LineChart** para hacer la gráfica de los datos. Cabe destacar que esta clase no es nativa de JAVA, ha sido descargada de <https://github.com/PhilJay/MPAndroidChart>.

En las tareas de eliminación de datos y del sensor lo que hacemos es borrar información de Firebase. Para ello utilizamos la función **removeValue()** que, aplicada a una referencia concreta, se encarga de eliminarla. Para la opción de eliminar datos las referencias serían “Temperaturas” y “Humedades”. Para la opción de eliminar el sensor la referencia sería “Sensor_X” y se borraría todo lo relativo a ese sensor en la base de datos.

Una vez ya hemos hablado de las opciones de eliminación de datos y del sensor vamos a discutir la cuestión del almacenamiento de Firebase, tal como hemos comentado antes. Vamos a volver a hacer números usando algunos resultados del apartado 3.5.1,

$$(1) \quad 1 \text{ GB} \cdot \frac{1024 \text{ MB}}{1 \text{ GB}} \cdot \frac{1024 \text{ KB}}{1 \text{ MB}} \cdot \frac{1000 \text{ medidas}}{75 \text{ KB}} \cdot \frac{1 \text{ año}}{526600 \text{ medidas}} \cong 26,6 \text{ años}$$

Como vemos en (1) podríamos almacenar todos los datos generados por un sensor durante 26,6 años. Si volvemos a la motivación original del proyecto, explicada en el apartado 1, y teniendo en cuenta que cuando el proceso de fermentación maloláctica acabe el sensor ya no necesitará seguir midiendo, se procederá a su eliminación para liberar espacio en Firebase y poder instalarse posteriormente en otra cuba de vino.

Para acabar, la aplicación también dispone de unas opciones especiales destinadas a los sensores ya registrados. Estas opciones las hemos llamado “Offline Options” ya que solo realizan consultas a Firebase y por lo tanto no necesitan estar conectadas por Wi-Fi al sensor. En este caso al usuario se le permite: obtener información del sensor, graficar los datos que están almacenados hasta el momento en Firebase y eliminar los datos del sensor. El código de esta parte se encuentra en los Anexos VII y VIII. El Anexo VIII hace referencia a la vista que muestra la lista de sensores registrados para que elijamos sobre cual actuar. El Anexo VII es la vista con las opciones antes mencionadas.

4. Puesta en marcha y resultados

Como podemos comprobar en el diagrama de Gantt del Anexo XIII el desarrollo del proyecto ha seguido el siguiente orden:

- I. Puesta a punto del sensor AM2302 con Arduino, adquisición de datos del sensor y visualización a través del LCD.
- II. Configuración de la red Wi-Fi en Raspberry y el punto de acceso en el dispositivo portátil además del escaner de IP's de la aplicación.
- III. Comunicaciones por cable entre Arduino y Raspberry.
- IV. Desarrollo principal de la APP Android.

El sistema completo con el que se han ido haciendo los test aparece en la Figura 43:

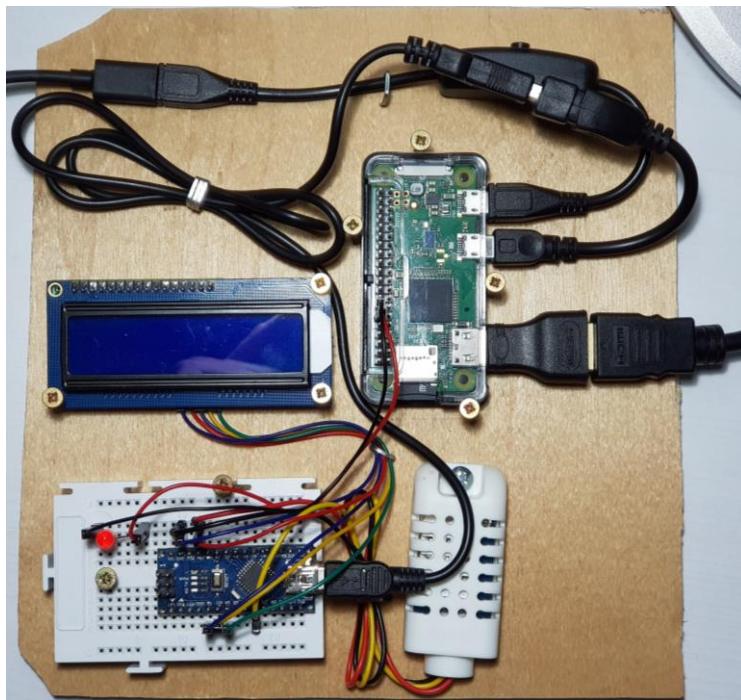


Figura 43: montaje final del sistema

Al mismo tiempo que se desarrollaba cada una de las partes del sistema se ha procedido a probar de manera aislada cada parte. En primer lugar, se escribió el código necesario para adquirir y visualizar en el LCD los datos del sensor. Para ello el programa iba obteniendo de manera periódica los datos y enviándolos al LCD mediante Arduino. Gracias al LCD se podía comprobar que se estaban obteniendo los datos correctamente. Una vez validada esta parte se procedió a desarrollar y validar la siguiente.

En segundo lugar, se configuraron las credenciales de la red predefinida en Raspberry y en el punto de acceso en el Smartphone y se implementó el cuerpo del código `main_wireless_rpi.py`. Acto seguido se empezó a construir el cuerpo principal de la APP. Una vez hecho se conectó Raspberry a la red y al realizar el escaneado se pudo ver como la aplicación detectaba la dirección IP. Para comprobar que efectivamente la dirección IP correspondía a la Raspberry se ejecutó el comando `sudo ifconfig`, que nos proporciona información de las direcciones IP de las interfacies de conexión de red.

En tercer lugar, se escribió el código necesario para ajustarse al protocolo descrito en el apartado 3.3 en el extremo de Arduino. Gracias al previo conocimiento sobre el uso de este tipo de comunicaciones en Raspberry, a medida que se iba desarrollando el código en Arduino se iban haciendo test. Los test se realizaban mediante un script sencillo en Raspberry, en la Figura 44, que enviaba una trama aislada para así comprobar que el código en el extremo de Arduino se estaba comportando como nosotros queríamos.

```

pi@raspberrypi: ~
pi@raspberrypi: ~
Archivo Editar Pestañas Ayuda
GNU nano 2.2.6 Fichero: send_frame.py

import serial
import time

arduino = serial.Serial(
    port = '/dev/ttyS0',
    baudrate = 9600,
    parity = serial.PARITY_NONE,
    stopbits = serial.STOPBITS_ONE,
    bytesize = serial.EIGHTBITS,
    # READ TIMEOUT in seconds (allow float)
    #timeout = 0.5
    # WRITE TIMEOUT in seconds (allow float)
    #writeTimeout = 0.5
)
data = bytearray([0xAA, 0x03, 0x01, 0x03, 0xAB, 0x00, 0x00, 0x00, 0x00])

while 1:
    arduino.write(data)
    # wait 10s until next send
    time.sleep(10)
  
```

Figura 44: script para pruebas serial en Arduino

Una vez validado el extremo de Arduino se procedió a desarrollar el código en el extremo de Raspberry. La metodología que se siguió fue la misma. Para validar este extremo se escribió un pequeño programa en Arduino que enviaba también una sola trama. El código se muestra en la Figura 45.

```

#define BAUDRATE 9600

byte frame[9];

void setup()
{
  Serial.begin(BAUDRATE, SERIAL_8N1);

  frame[0] = 0xAA;
  frame[1] = 0x07;
  frame[2] = 0x01;
  frame[3] = 0x02;
  frame[4] = 0x96;
  frame[5] = 0x00;
  frame[6] = 0x8A;
  frame[7] = 0x02;
  frame[8] = 0xB0;
}

void loop() // run over and over
{
  partial = millis();
  if(time == 0)
  {
    time = millis();
  }
  else if(time == 10000){
    Serial.write(frame, sizeof(frame));
    time = 0;
  }
}

```

Figura 45: código para pruebas serial en Raspberry

Una vez subido el programa `main_wired_arduino.ino` a Arduino, ponemos a ejecutar el programa `main_wired_rpi.py` en Raspberry. El resultado aparece en la Figura 46.

```

pi@raspberrypi: ~ $ sudo python main_wired_rpi.py

FRAME REQUEST SENT: AA 03 01 03 AB 00 00 00 00
CORRECT HEADER DETECTED - FRAME TIMEOUT STARTED
MASTER NOT NEED TO CHECK ID SLAVE
FRAME RECEIVED: AA 07 01 02 BE 00 FB 02 E9
FRAME PROCESS: AA 07 01 02 BE 00 FB 02 E9
THE LAST INPUT FRAME HAS CORRECT CKS - LET'S GO TO PROCESS...
Temerature: 190
Humidity: 763
Writting in measurments file...
FRAME ACK SENT: AA 03 01 04 AC 00 00 00 00

```

Figura 46: proceso de medidas visto desde Raspberry

Además de comprobar que el dato se ha capturado correctamente con el log del terminal de Raspberry, mostrado en la Figura 46, también es importante que el contenido del fichero **measurements.txt**, en la Figura 47, se haya escrito correctamente.

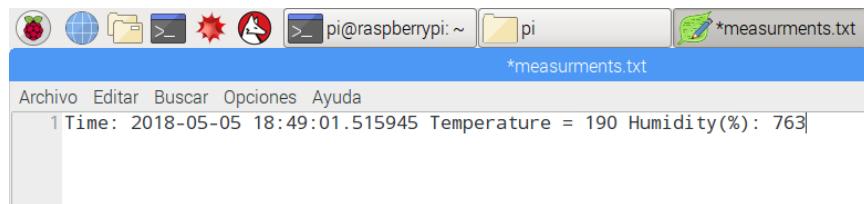


Figura 47: estructura fichero measurements.txt

Para comprobar que no hay diferencias entre el dato capturado por Arduino y el dato escrito en el fichero por Raspberry disponemos del LCD. Como se puede observar en la Figura 24 mostrada anteriormente, el dato coincide.

Por último, se desarrolló todo lo que es la lógica de la aplicación para gestionar los sensores. En este caso se fue probando cada funcionalidad de las que hemos hablado en el apartado 3.6 de forma aislada. Una vez funcionaba correctamente la función específica se procedía a implementar la siguiente. El funcionamiento de la APP completa se especifica a continuación.

En la pantalla principal de la APP se realiza la búsqueda de sensores (direcciones IP), la elección de a cuál queremos conectarnos y las Opciones Offline. La lista de IPs estará vacía hasta que hagamos click en "START SCANNING". Mientras busca sensores se irá completando una barra de progreso y a su vez se irá llenando la lista con las IPs que estén conectadas al punto de acceso generado por el Smartphone. La captura de la lista de IPs una vez terminada la búsqueda se muestra en la Figura 48.

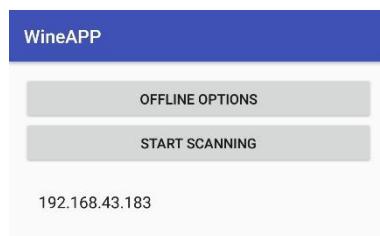


Figura 48: pantalla inicial de la APP

Por el contrario, si ya tenemos sensores registrados, la lista de IPs aparecerá con los ítems registrados por defecto, tal como aparece en la Figura 49.

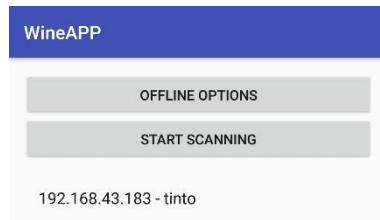


Figura 49: pantalla inicial con IPs registradas

Estas opciones sólo estarán disponibles para aquellos sensores ya registrados. Las opciones que puede ejecutar el usuario a través de este menú no necesitan que el sensor se encuentre con la conexión al punto de acceso activa, por lo tanto pueden ejecutarse en cualquier momento. Son acciones que actúan sobre las referencias a la base de datos de cada sensor registrado. Si hacemos click en el botón “OFFLINE OPTIONS” del menú de la Figura 48 accederemos al siguiente Activity.



Figura 50: sensores registrados pero no activos

Como podemos ver en la Figura 50, en la lista de IPs aparecen únicamente los sensores registrados. Si hacemos click en uno de ellos accederemos el menú de la Figura 51.

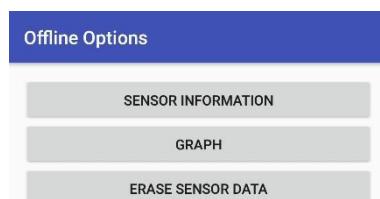


Figura 51: pantalla Offline Options

Si hacemos click en “SENSOR INFORMATION” la aplicación obtiene la información de Firebase y la muestra en la lista que aparece debajo de los botones de la Figura 52.



Figura 52: información del sensor en Offline Options

La APP también nos da la posibilidad de visualizar gráficamente los datos almacenados en Firebase por el sensor “tinto” si hacemos click en “GRAPH”. Al hacer click se accede a un nuevo Activity que explicaremos más adelante.

Por último podemos eliminar los datos almacenados en Firebase dentro del sensor “tinto”. Si hacemos click en “ERASE SENSOR DATA” se abre una pantalla de confirmación para realizar o cancelar la acción. Tenemos una captura en la Figura 53.

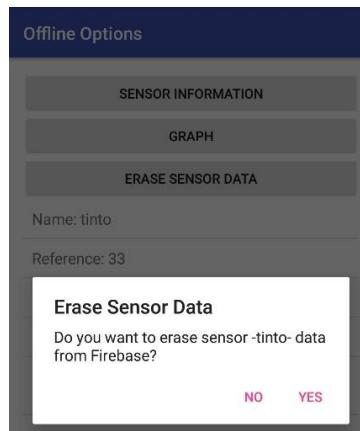


Figura 53: pantalla de borrado en Offline Options

Podríamos referirnos a este Activity como el menú principal de la APP, ya que es aquí donde podemos realizar todas las tareas relativas a un sensor. En la Figura 54 podemos ver el listado de acciones que podemos realizar. Además también se muestra la información de la que disponemos del sensor en ese momento.

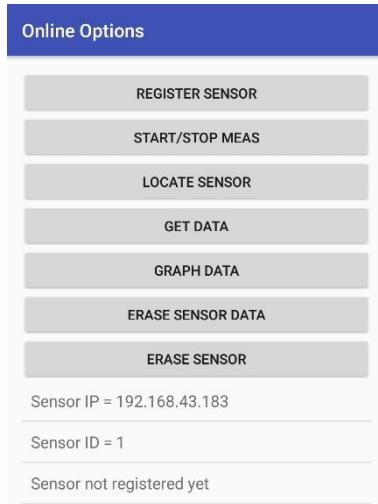


Figura 54: pantalla Online Options

Si hacemos click en el botón “REGISTER SENSOR” accederemos a un nuevo Activity. En este Activity se ejecutará el registro del sensor en Firebase y en nuestra APP.

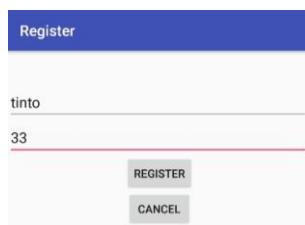


Figura 55: ventana de registro

La Figura 55 muestra la ventana de registro donde deberemos introducir obligatoriamente Nombre y Referencia del sensor antes de registrar. Si hacemos click en “REGISTER” aparecerá en Firebase una entrada asociada al sensor como la de la Figura 42.

Como podemos ver dentro de la rama “Sensors” se añade “Sensor_1”, dónde el número que identifica al sensor siempre será su ID actual. Además no solo guardará Nombre y Referencia sino que también guardará de forma automática:

- Fecha de creación
- ID
- Dirección IP

Una vez registrado el sensor esta función quedará inaccesible hasta que eliminemos el sensor. En la interfaz gráfica se mostrará como en la Figura 56 además de con un mensaje flotante que aparecerá si pulsamos nuevamente el botón “REGISTER SENSOR”.



Figura 56: pantalla Online Options de un sensor registrado

Si hacemos click en el botón “START/STOP MEAS” informaremos al Servidor que haga lo contrario de lo que está haciendo actualmente, es decir: si el sensor había iniciado anteriormente el proceso de medidas lo parará, sino lo comenzará. El propio botón nos indicará de manera visual si está midiendo ya que se coloreará de color verde en caso afirmativo.

Si hacemos click en el botón “LOCATE SENSOR” daremos aviso al servidor que está ejecutándose en Raspberry de que tiene que poner a parpadear el LED. La vista de Online Options se actualizará como la de la Figura 57 para avisarnos de que el LED está parpadeando. Si volvemos a hacer click el LED se apagará y la vista de la APP volverá a ser como la de la Figura 54. Esta funcionalidad es accesible en todo momento.

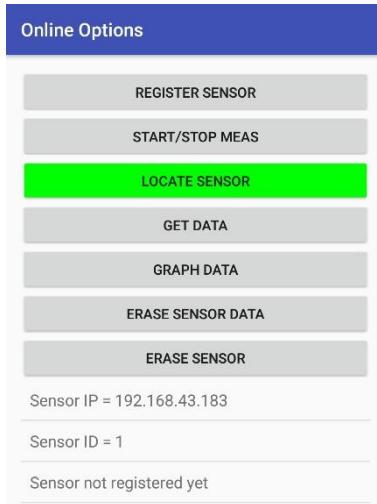


Figura 57: pantalla Online Options con localización activada

Si hacemos click en el botón “GET DATA” solicitaremos al sensor que nos envíe los datos que ha almacenado hasta el momento. Primero obtendremos el número de líneas que debemos leer, después las leeremos una a una y finalmente se añadirán a Firebase en el sensor correspondiente. A modo provisional, para poder comprobar desde la misma APP que los datos se han obtenido correctamente, se añaden a la lista que veíamos en la Figura 54, dónde se muestra también la información del sensor.

Si hacemos click en el botón “GRAPH DATA” accederemos al mismo Activity que comentábamos en las Opciones Offline. Esta función no está accesible hasta que el sensor no esté registrado, de lo contrario la APP nos avisará con un mensaje flotante de que primero debemos registrarla.

Si hacemos click en el botón “ERASE SENSOR DATA” se nos abrirá una ventana de confirmación como la que aparece en la Figura 53. En caso afirmativo los datos referentes a ese sensor en Firebase se borrarán. Esta función no está accesible hasta que el sensor no esté registrado, de lo contrario la APP nos avisará con un mensaje flotante de que primero debemos registrarla.

Si hacemos click en “ERASE SENSOR” se nos abrirá también una ventana de confirmación como la mostrada en la Figura 58. En caso afirmativo se eliminarán todas las referencias al sensor activo en Firebase y además se notificará al cliente que el sensor ha sido eliminado para que pare el proceso de medidas en caso de que este ejecutándose y para que inicialice su ID. Esta función no está accesible hasta que el sensor no esté registrado, de lo contrario la APP nos avisará con un mensaje flotante de que primero debemos registrarla.

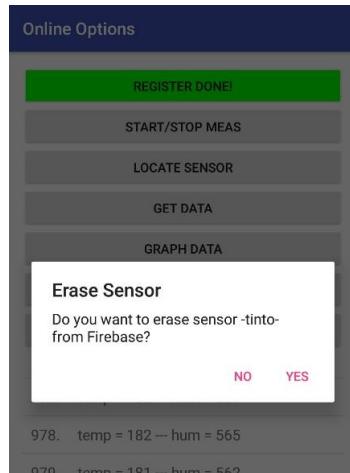


Figura 58: ventana Erase Sensor

También podremos visualizar en forma de gráfica los datos que Firebase tenga almacenados referentes al sensor al que estamos conectados. Como se puede ver en la Figura 59, inicialmente no aparecerá ninguna gráfica. Al hacer el primer click en el botón “SHOW DATA” nos mostrará la gráfica de temperaturas (color rojo). Si volvemos a hacer click, nos mostrará la gráfica de RH (color azul).

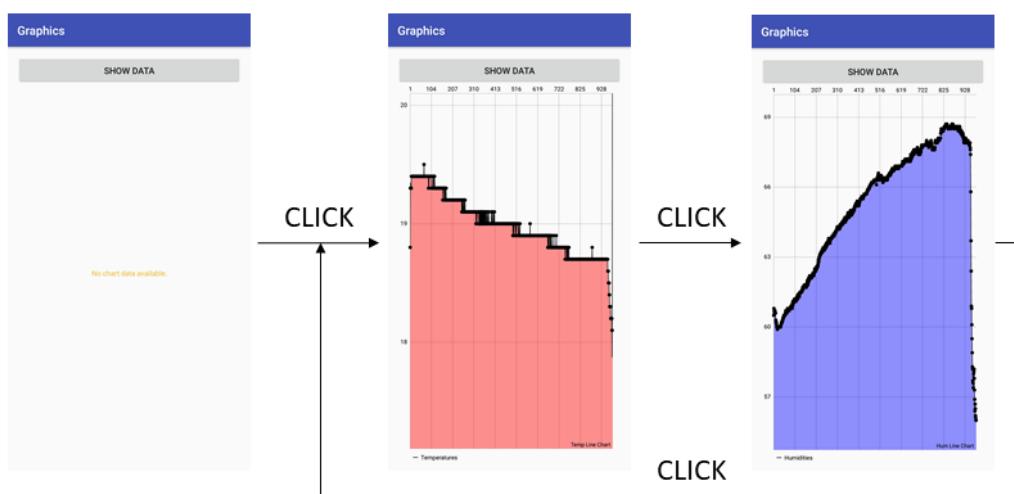


Figura 59: pantalla de gráficas

Por último, se realizó un test de alcance inalámbrico para averiguar si estábamos dentro de los objetivos o no. Para ello se puso a funcionar el sistema en la calle con una powerbank y dejando el conjunto de la Figura 43 fijo nos dispusimos a alejarnos con el Smartphone. El momento de pérdida de conexión determinó el alcance, que resultaron ser aproximadamente 30m.

Adicionalmente, hemos comprobado que las especificaciones fijadas para el protocolo de comunicaciones alámbricas son las deseadas (apartado 3.3). Para ello hemos capturado la trama de solicitud de datos con un osciloscopio. En la Figura 60 podemos ver que los tiempos de trama son los esperados.

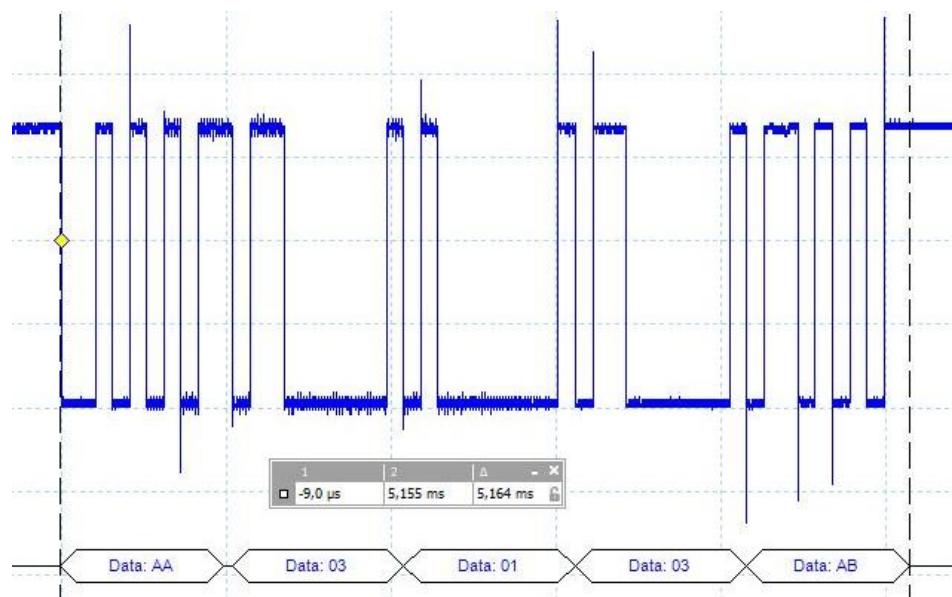


Figura 60: trama de solicitud a través de PicoScope

5. Presupuesto

A continuación se van a especificar los costes de todos los materiales utilizados para el desarrollo del proyecto. Actualmente se han comprado materiales para montar dos sensores independientes. El desglose aparece en la Tabla 4.

MATERIAL	PRECIO	CANTIDAD
Raspberry Pi Zero W Pack	29,2 €	2
Arduino NANO	2,25 €	2
AM2302	12,7 €	2
LCD 16x2	23,04 €	2
Protoboard	3,4 €	2
Cables H-H	3,38 €	2
Cables H-M	3,29 €	2
Diodo LED	0,25 €	2
TOTAL	155,02 €	

Tabla 4: listado de materiales

Por último, los costes totales de desarrollo del proyecto se adjuntan en la Tabla 5.

HORAS INGENIERO	COSTE	TOTAL
13 semanas x 40h/semana	40 €/h	20.800 €
PRESUPUESTO FINAL		
Materiales		155,02 €
Coste horas		20.800 €
TOTAL		20.955,02 €

Tabla 5: presupuesto final

6. Conclusiones y futuros pasos:

En este proyecto se ha desarrollado un sistema para dotar de conectividad inalámbrica un sensor ultrásónico, que se encarga de medir el nivel de fermentación maloláctica del vino, y además poder gestionarlo mediante un dispositivo externo. En primera instancia y para asegurar el cumplimiento de los objetivos en el tiempo previsto, el desarrollo se ha hecho con un sensor de temperatura y humedad con el fin de imitar la salida digital del sensor de fermentación.

A partir de las necesidades iniciales del sistema y de cada uno de los bloques funcionales que lo componen se escogió un hardware adecuado para cada uno de ellos. En nuestro caso escogimos una placa Arduino NANO para la adquisición de datos del sensor auxiliar y una Raspberry Pi Zero W para la comunicación por UART con Arduino y para la comunicación por Wi-Fi con la APP Android. Las dos placas escogidas trabajan a +5V con lo cual estamos dentro de las especificaciones iniciales.

El primer bloque se encarga de adquirir los datos del sensor y transmitirlos por UART a Raspberry. Para ello se decidió implementar un protocolo de comunicaciones específico para dar robustez a las comunicaciones por cable del bloque. Como se ha explicado en el apartado de Resultados se han realizado una serie de test para verificar que el bloque cumple con las especificaciones iniciales del sistema.

El segundo bloque se encarga de recibir los datos de Arduino, almacenarlos y esperar conectarse con la APP para ejecutar las tareas requeridas por el usuario. Para ello se hace uso del protocolo alámbrico de comunicaciones que comentábamos y del protocolo TCP/IP para la comunicación inalámbrica. Nuevamente, como se ha explicado en el apartado de Resultados, se han realizado los test necesarios para verificar que las dos funcionalidades del bloque se ajustan a las especificaciones iniciales del bloque. El proyecto marcaba como especificación que fuera posible almacenar las medidas generadas durante al menos un mes, hemos visto que cumplimos esta especificación holgadamente. Además también se requería un alcance mínimo de la red Wi-Fi de 10m, haciendo un test de alcance hemos obtenido un alcance de aproximadamente 30m.

El tercer bloque se encarga de gestionar el sensor a través de la aplicación Android diseñada. Con ella el usuario podrá realizar todas las acciones que requerían las especificaciones del bloque. Se han realizado múltiples test durante el proceso de desarrollo de la aplicación con el fin de probar de manera independiente todas y cada una de las acciones. Además, la especificación inicial de que el usuario pueda manejar hasta 10 sensores la hemos superado con creces, ya que al ir asociados a direcciones IP podremos gestionar tantos sensores como direcciones IP disponibles haya en la red generada por el dispositivo portátil. Por lo tanto podremos gestionar hasta 253 sensores.

Finalmente, se ha realizado un test de todo el sistema dónde hemos podido ver que los tres bloques juntos son capaces de funcionar correctamente y ofrecer al usuario una interfaz amigable con la que manejar la red de sensores. En el apartado anterior hemos visto que el coste del sistema es, sin costes de mano de obra, de 77,5€, por tanto cumplimos la especificación inicial de coste máximo de 100€.

Cabe destacar que finalmente no ha sido posible realizar el estudio de alimentar el sistema con baterías. Esta es realmente una necesidad del sistema, ya que como hemos argumentado desde el principio, una de las motivaciones del proyecto era evitar cablear instalaciones en entornos industriales debido a su dificultad y poca practicidad. Dicho esto, queda como tarea pendiente y posible mejora.

Una última mejora que cabría mencionar es la posibilidad de poder evolucionar el sistema para que fuera completamente autónomo y ni siquiera el usuario necesitase ir a la bodega para descargar los datos de cada sensor. El sistema actual ya está diseñado de tal forma que mejoras de este tipo no supondrían una gran carga de trabajo.

Bibliografía:

<https://learn.adafruit.com/dht>

<https://learn.adafruit.com/i2c-spi-lcd-backpack/overview>

<https://www.arduino.cc/reference/en/language/functions/communication/serial/>

<https://docs.python.org/2.7/>

https://pythonhosted.org/pyserial/pyserial_api.html

<https://firebase.google.com/docs/database/>

<https://stackoverflow.com/>

Anexos:

I. main_wired_arduino.ino

```

/*
- PROYECTO FINAL DE GRADO REALIZADO POR OSCAR LIÑAN LOPEZ.
- GRADO EN CIENCIAS Y TECNOLOGIAS DE LAS TELECOMUNICACIONES, ETSETB.
- SISTEMA DE RECOGIDA, ALMACENAMIENTO Y ENVIO INALAMBRICO DE INFORMACION DE
SENSORES
*/
#include <Wire.h>
#include <LiquidCrystal.h>
#include <DHT.h>

#define DHTPIN 7          // what digital pin we connect the sensor data wire
#define DHTTYPE DHT22    // DHT 22  (AM2302, AM2321)
#define BAUDRATE 9600    // communication baudrate

// initialize LCD for SPI. Data pin is #3, Clock is #2 and Latch is #4
LiquidCrystal lcd(3, 2, 4);

// Initialize DHT sensor
DHT dht(DHTPIN, DHTTYPE);

// Sensor data variables
int hum, temp;
byte hum_high, hum_low, temp_high, temp_low;
boolean readOK;

// Control variables
boolean frameComplete;
byte bytes_received, header, slave_id, lastFrame, frameLen, measure_rtx,
cks_calculated;

// Frame variables - max 9 bytes by frame
byte frame_rx[9], frame_process[9], frame_tx[9], empty_frame [9];

/*
FRAME STRUCTURE:
frame[0]      = header
frame[1]      = length (id's + data + cks)
frame[2]      = id_slave
frame[3]      = id_frame

```

```
frame[4..x] = data      |
                  |--> with x<9
frame[x+1] = checksum |
*/



// Frame Time in millis
unsigned long frame_time;



// Frame Timeout variables
unsigned long startTimeFrame;           // value in millis when serial receives
                                         right header
unsigned long escapeTimeFrame;          // value in millis when time out
unsigned long durationTimeFrame;        // value in millis of current time


// ACK-NACK wait Timeout variables
unsigned long startWaitACK;            // value in millis when serial receives
                                         right header
unsigned long escapeWaitACK;           // value in millis when time out
unsigned long durationWaitACK;         // value in millis of current time


void setup()
{
    // initialize variables
    readOK = false;

    // Control variables
    frameComplete = false;
    bytes_received = 0;
    header = 0xAA;
    slave_id = 0x01;
    lastFrame = 0x00;
    frameLen = 0x00;
    measure_rtx = 0;
    cks_calculated = 0;

    // start serial
    Serial.begin(BAUDRATE, SERIAL_8N1);

    // set up the LCD's number of rows and columns
    lcd.begin(16, 2);
```

```
// initialize LCD
lcd.setBacklight(HIGH);
lcd.setCursor(0, 0);

// start sensor communication
dht.begin();

//frame_time = (sizeof(frame_rx) * 8 * 1000) / BAUDRATE;
frame_time = sizeof(frame_rx)*8;
frame_time = frame_time*1000;
frame_time = round((frame_time/BAUDRATE)+1);
escapeTimeFrame = frame_time * 2;           // 16ms
escapeWaitACK = 2 * frame_time * 3;          // 48ms
}

void loop()
{
    // ACK/NACK timeout control
    if(startWaitACK != 0)
    {
        durationWaitACK = millis() - startWaitACK;
    }
    // if wait ACK/NACK timeout
    if(durationWaitACK >= escapeWaitACK)
    {
        measure_rtx++;
        // check number of rtx
        if(measure_rtx < 3)
        {
            // rtx measure to RPi and restart RPi response timeout
            startWaitACK = millis();
            durationWaitACK = 0;
            // send the sensor data by serial
            Serial.write(frame_tx, sizeof(frame_tx));
        }
        // too much rtx done, waiting new request
        else
        {
            // error flag in next tx frame??
        }
    }
}
```

```
// complete frame available to process
if (frameComplete)
{
    // copy the frame into another array to avoid lose rx data and clear rx frame
    memcpy(frame_process, frame_rx, sizeof(frame_rx[0])*sizeof(frame_rx));
    memcpy(frame_rx, empty_frame, sizeof(empty_frame[0])*sizeof(empty_frame));

    // check frame CKS
    cks_calculated = XORChecksum8(frame_process, (frameLen + 2));

    // process the input frame received correctly
    if(cks_calculated == 0x00)
    {
        switch(frame_process[3])
        {
            case 0x01: // CONFIG frame
                // config with frame_rx[3]
                break;
            case 0x02: // MEDIDA frame
                // frame not precessed by Arduino
                break;

            case 0x03: // SOLICITUD frame
                // Reading temperature or humidity takes about 250 milliseconds
                while(readOK == false)
                {
                    // read % humidity
                    hum = dht.readHumidity() * 10;
                    // Read temperature as Celsius
                    temp = dht.readTemperature() * 10;
                    // Check if any reads failed and exit to try again
                    if (isnan(hum) || isnan(temp))
                    {
                        lcd.setCursor(0, 0);
                        lcd.print("Failed to read ");
                        lcd.setCursor(0, 1);
                        lcd.print("from DHT sensor!");
                        readOK = false;
                    }
                    else readOK = true;
                }
        }
    }
}
```

```
showInLCD(temp, hum);

// send the sensor data by serial
packFrame(header, 0x07, slave_id, 0x02);
Serial.write(frame_tx, sizeof(frame_tx));

// Start RPi response timeout
startWaitACK = millis();
break;

case 0x04: // ACK frame
if(lastFrame == 0x03) // last received was MEASURE request
{
    // last frame sent received by raspberry correctly ---> reset measure
    // response timeout and rtx variables
    durationWaitACK = 0;
    startWaitACK = 0;
    measure_rtx = 0;
}
break;

case 0x05: // NACK frame
// last frame sent received by raspberry incorrectly --->
// RTX the last frame sent
if(lastFrame == 0x03) // last received was MEASURE request
{
    // count measure frame RTX
    measure_rtx++;
    if(measure_rtx < 3)
    {
        // Restart RPi response timeout
        startWaitACK = millis();
        durationWaitACK = 0;
        // send the sensor data by serial
        Serial.write(frame_tx, sizeof(frame_tx));
    }
}
else
{
    // initialize all variable values
    lastFrame = 0x00;
    frameLen = 0x00;
```

```
        frameComplete = false;
        readOK = false;
        memcpy(frame_process, empty_frame,
               sizeof(empty_frame[0])*sizeof(empty_frame));
    }
    break;

default:
    // initialize all variable values
    lastFrame = 0x00;
    frameLen = 0x00;
    frameComplete = false;
    readOK = false;
    memcpy(frame_process, empty_frame,
           sizeof(empty_frame[0])*sizeof(empty_frame));
    break;
}

// after valid frame process, save last frame id received and reset variable
values
lastFrame = frame_process[3];
frameLen = 0x00;
frameComplete = false;
readOK = false;
memcpy(frame_process, empty_frame,
       sizeof(empty_frame[0])*sizeof(empty_frame));
}

// frame with errors ---> rejected
else
{
    // initialize all variable values
    lastFrame = 0x00;
    frameLen = 0x00;
    frameComplete = false;
    readOK = false;
    memcpy(frame_process, empty_frame,
           sizeof(empty_frame[0])*sizeof(empty_frame));
}
}
```

```
/*
SerialEvent occurs whenever a new data comes in the
hardware serial RX. This routine is run between each
time loop() runs, so using delay inside loop can delay
response. Multiple bytes of data may be available.

*/
void serialEvent() {
    if(startTimeFrame != 0)
    {
        durationTimeFrame = millis() - startTimeFrame;
    }

    // if no timeout
    if(durationTimeFrame <= escapeTimeFrame)
    {
        // get and store the new byte
        byte inByte = (byte)Serial.read();
        frame_rx[bytes_received] = inByte;
        bytes_received++;

        if(bytes_received == 1)
        {
            // check header
            if(inByte == header)
            {
                // start frame timeout
                startTimeFrame = millis();
            }
            else {
                // this is an invalid frame ---> rejected
                memcpy(frame_rx,empty_frame, sizeof(empty_frame[0])*sizeof(empty_frame));
                bytes_received = 0;
                durationTimeFrame = 0;
                startTimeFrame = 0;
            }
        }
        else if(bytes_received == 2)
        {
            //get frame length
            frameLen = inByte;
        }
    }
}
```

```
else if(bytes_received == 3)
{
    //check ID slave
    if(inByte != slave_id)
    {
        // this is an invalid frame ---> rejected
        memcpy(frame_rx, empty_frame, sizeof(empty_frame[0])*sizeof(empty_frame));
        bytes_received = 0;
        durationTimeFrame = 0;
        startTimeFrame = 0;
    }
}

else if(bytes_received == frameLen + 2)
{
    frameComplete = true;
    bytes_received = 0;
    durationTimeFrame = 0;
    startTimeFrame = 0;
}
}

// if timeout
else
{
    // this is an invalid frame ---> rejected
    memcpy(frame_rx,empty_frame, sizeof(empty_frame[0])*sizeof(empty_frame));
    bytes_received = 0;
    durationTimeFrame = 0;
    startTimeFrame = 0;
}
}

byte XORChecksum8(const byte *data, byte len) {
    byte value = 0;
    for (byte i = 0; i < len; i++)
    {
        value = (byte)(value ^ data[i]);
    }
    return value;
}
```

```
void packFrame(byte head, byteflen, byteids, byteidf)
{
    temp_low = temp & 0xFF;
    temp_high = (temp >> 8) & 0xFF;
    hum_low = hum & 0xFF;
    hum_high = (hum >> 8) & 0xFF;

    frame_tx[0] = head;
    frame_tx[1] =flen;
    frame_tx[2] =ids;
    frame_tx[3] =idf;
    frame_tx[4] = temp_low;
    frame_tx[5] = temp_high;
    frame_tx[6] = hum_low;
    frame_tx[7] = hum_high;
    frame_tx[8] = XORChecksum8(frame_tx, sizeof(frame_tx) - 1);
}

void showInLCD(int t, int h)
{
    lcd.setCursor(0, 0);
    lcd.print("Temp: ");
    lcd.setCursor(6, 0);
    lcd.print(t);
    lcd.setCursor(10, 0);
    lcd.print("*C");
    lcd.setCursor(0, 1);
    lcd.print("Hum: ");
    lcd.setCursor(6, 1);
    lcd.print(h);
    lcd.setCursor(9, 1);
    lcd.print(" %");
}
```

II. main_wired_rpi.py

```
import time, serial, os
from myFrame import myFrame

def writeInFile(str):
    print '\nWritting in measurements file...'
    file = open("measurements.txt", "a")
    file.write(str + "\n")
    file.close()

def main():
    # serial port configuration
    arduino = serial.Serial(
        port = '/dev/ttyUSB0',
        baudrate = 9600,
        parity = serial.PARITY_NONE,
        stopbits = serial.STOPBITS_ONE,
        bytesize = serial.EIGHTBITS,
        # READ & WRITE TIMEOUT in seconds (allow float)
        #timeout = 0.5
        #writeTimeout = 0.5
    )
    # funcion to get current time in milliseconds
    millis = lambda: int(round(time.time() * 1000))

    # period measurement request every minute
    period = 60000

    # periodic data request variables
    partial = 0
    current = 0

    frame_tx = myFrame(9)
    frame_rx = myFrame(9)
    frame_process = myFrame(9)

    while 1:
        # send periodic request
        partial = millis()
```

```
if current == 0:
    current = millis()
elif (partial - current) > period:
    frame_tx.send_flag = 3
    current = 0

# read serial incoming data
frame_rx.readIntoFrame(arduino)

# process frame
frame_process.processFrame(frame_rx, frame_tx)

# writting data to file
if frame_process.write_in_file == True:
    writeInFile(frame_process.str_to_write)
    frame_process.write_in_file = False

# send frame if necessary
if frame_tx.send_flag == 3:
    frame_tx.packTxFrame('REQUEST')
    arduino.write(frame_tx.getFrame())
    frame_tx.send_flag = 0
elif frame_tx.send_flag == 4:
    frame_tx.packTxFrame('ACK')
    arduino.write(frame_tx.getFrame())
    frame_tx.send_flag = 0
elif frame_tx.send_flag == 5:
    frame_tx.packTxFrame('NACK')
    arduino.write(frame_tx.getFrame())
    frame_tx.send_flag = 0

if __name__ == '__main__':
    main()
```

III. myFrame.py

```
import time, datetime, os

# funcion to get current time in milliseconds
millis = lambda: int(round(time.time() * 1000))

baudrate = 9600

class myFrame():
    frame = bytearray()

    # control variables
    send_flag = 0
    frameComplete = False
    write_in_file = False
    str_to_write = ''
    frameLen = 0

    # data variables
    temp = 0
    hum = 0

    # read variables
    bytes_rx = 0
    durationTimeFrame = 0
    startTimeFrame = 0
    escapeTimeFrame = 0

    # process variables
    durationWaitMeasure = 0
    startWaitMeasure = 0
    nack_rtx = 0
    escapeWaitMeasure = 0

    def __init__(self, size):
        self.frame = bytearray(size)
        self.escapeTimeFrame = ((size*9*1000)/baudrate)*1.2
        self.escapeWaitMeasure = (2*(size*9*1000)/baudrate)*1.2
```

```
def __len__(self):
    return len(self.frame)

def copyFrame(self, dest):
    for i in range(0, len(self)):
        dest[i] = self.frame[i]

def getFrame(self):
    return self.frame

def putItem(self, byteVal, index):
    self.frame[index] = byteVal

def getItem(self, index):
    return self.frame[index]

def getTemp(self):
    return self.temp

def getHum(self):
    return self.hum

def setTemp(self, temp):
    self.temp = temp

def setHum(self, hum):
    self.hum = hum

def restartFrame(self):
    for i in range (0, len(self)):
        self.frame[i] = 0x00
def printFrame(self):
    return ' '.join(["%02X" % ord(x) for x in str(self.frame)]) 

def computeCKS(self):
    cks = 0
    for i in range(0, (self.frame[1] + 2)):
        cks = cks ^ self.frame[i]
    return cks
```

```

def packTxFrame(self, ftype):
    self.restartFrame()
    self.frame[0] = 0xAA
    self.frame[1] = 0x03
    self.frame[2] = 0x01
    if ftype == 'REQUEST':
        self.frame[3] = 0x03
    elif ftype == 'ACK':
        self.frame[3] = 0x04
    elif ftype == 'NACK':
        self.frame[3] = 0x05
    self.frame[4] = self.computeCKS()
    print('\nFRAME ' + ftype + ' SENT: ' + self.printFrame())

def readIntoFrame(self, serial):
    # upload durationTimeFrame value
    if self.startTimeFrame != 0:
        self.durationTimeFrame = millis() - self.startTimeFrame

    # if no timeout
    if self.durationTimeFrame <= self.escapeTimeFrame:
        if serial.inWaiting() > 0:
            # restart frame
            if self.bytes_rx == 0 and self.frameComplete == False:
                self.restartFrame()
            # read and save the new byte
            inByte = serial.read(1)
            self.frame[self.bytes_rx] = inByte
            self.bytes_rx += 1

            # check header
            if self.bytes_rx == 1:
                # valid frame
                if inByte == '\xAA':
                    # start rx frame timeout
                    #startTimeFrame = millis()
                    print '\nCORRECT HEADER DETECTED - FRAME TIMEOUT
STARTED\n'

```

```
        else:
            # invalid frame ---> restart variables
            self.bytes_rx = 0
            self.durationTimeFrame = 0
            self.startTimeFrame = 0
            print 'INCORRECT HEADER DETECTED - NOT A VALID
            FRAME\n'

        # get frame length
        elif self.bytes_rx == 2:
            self.frameLen = ord(inByte)

        elif self.bytes_rx == 3:
            # master not need to check id slave
            print 'MASTER NOT NEED TO CHECK ID SLAVE\n'

        # all frame read
        elif self.bytes_rx == (self.frameLen + 2):
            self.frameComplete = True
            self.bytes_rx = 0
            self.durationTimeFrame = 0
            self.startTimeFrame = 0
            print('FRAME RECEIVED: ' + self.printFrame() + '\n')

    # if timeout
    else:
        self.bytes_rx = 0
        self.durationTimeFrame = 0
        self.startTimeFrame = 0
        self.frameComplete = True # to force a NACK sending
        self.ready_to_write = True
        self.str_to_write = "FRAME TIMEOUT ENDS - ERROR IN RX FRAME"

def processFrame(self, frame_rx, frame_tx):
    # MEASURE timeout control
    if self.startWaitMeasure != 0:
        self.durationWaitMeasure = millis() - self.startWaitMeasure
```

```
# MEASURE timeout ends ---> RTX Measure Request
if self.durationWaitMeasure > self.escapeWaitMeasure:
    print "WAIT MEASURE TIMEOUT ENDS - SEND NEW ACK REQUIRED"
    self.nack_rtx += 1

# check number of RTX
if self.nack_rtx < 4:
    # restart wait timeout
    self.startWaitMeasure = millis()
    self.durationWaitMeasure = 0
    # flag NACK frame ---> send_flag = 5
    frame_tx.send_flag = 5
    # store error in file
    self.ready_to_write = True
    self.str_to_write = "FRAME NOT RECEIVED BEFORE TIMEOUT
- RTX REQUIRED"

else:
    print "TOO MUCH RTX DONE. WAITING TO SEND NEXT REQUEST"
    # store error in file
    self.ready_to_write = True
    self.str_to_write = "TOO MUCH RTX DONE - WAITING TO
SEND NEXT REQUEST"
    # restart timeout control variables
    self.durationWaitMeasure = 0
    self.startWaitMeasure = 0
    self.nack_rtx = 0

if frame_rx.frameComplete == True:
    # copy the frame into another array to avoid lose rx data and
    # clear rx frame
    frame_rx.copyFrame(self.getFrame())
    frame_rx.restartFrame()
    print("\nFRAME PROCESS: " + self.printFrame() + '\n')

    # check frame CKS
    if self.computeCKS() == 0:
        print "\nTHE LAST INPUT FRAME HAS CORRECT CKS - LET'S GO TO
PROCESS..."
```

```
# CONFIG frame
if self.frame[3] == 1:
    print "CONFIG FRAME IS NOT PROCESSED BY MASTER"

# MEASURE frame ---> like ACK for MASTER
elif self.frame[3] == 2:
    # restart timeout control variables
    self.durationWaitMeasure = 0
    self.startWaitMeasure = 0
    self.nack_rtx = 0
    # decode received data and write it into a txt file
    self.setTemp((self.frame[5] << 8) + self.frame[4])
    print("\nTemerature: " + str(self.temp))
    self.setHum((self.frame[7] << 8) + self.frame[6])
    print("Humidity: " + str(self.hum))
    # flag to send ACK to slave ---> send_flag = 4
    frame_tx.send_flag = 4
    self.write_in_file = True
    self.str_to_write = ("Time: " +
        str(datetime.datetime.now()) + " Temperature = " +
        str(self.temp) + " Humidity(%): " + str(self.hum))

# REQUEST frame
elif self.frame[3] == 3:
    print "REQUEST FRAME IS NOT PROCESSED BY MASTER"

# ACK frame
elif self.frame[3] == 4:
    print "ACK FRAME IS NOT PROCESSED BY MASTER"

# NACK frame
elif self.frame[3] == 5:
    print "NACK FRAME IS NOT PROCESSED BY MASTER"

# frame with errors
else:
    # flag to send NACK to slave ---> send_flag = 5
    frame_tx.send_flag = 5
    # store error log
    self.write_in_file = True
```

```
self.str_to_write = "FRAME RECEIVED WITH ERRORS - RTX
REQUIRED"

# restart variables to wait next frame
frame_rx.frameComplete = False
self.durationWaitMeasure = 0
self.startWaitMeasure = 0
self.nack_rtx = 0

# restart variables to wait next frame
frame_rx.frameComplete = False
self.durationWaitMeasure = 0
self.startWaitMeasure = 0
self.nack_rtx = 0
```

IV. main_wireless_rpi.py

```
import sys, socket, select, urllib2, time, os, subprocess
import RPi.GPIO as GPIO
from shutil import copyfile

HOST = ''
SOCKET_LIST = []
RECV_BUFFER = 4096
PORT = 9009

# gpio pin setup
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)

# function to get current time in milliseconds
millis = lambda: int(round(time.time() * 1000))

global blink_flag, current, start, download_flag, meas_flag
download_flag = False
blink_flag = False
meas_flag = False
current = 0
start = 0

def chat_server():

    global ID, blink_flag, download_flag, meas_flag
    file = open("id.txt", "r")
    ID = int(file.readline())
    file.close()

    try:
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print 'Socket created.'
    except socket.error, msg:
        print 'Failed to create socket. Error code: ' + str(msg[0]) + \
              ' Message ' + msg[1] + '.'
        sys.exit()

    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
try:
    server_socket.bind((HOST, PORT))
except socket.error, msg:
    print 'Bind failed. Error code: ' + str(msg[0]) + ' Message ' +
          msg[1] + '.'
    sys.exit()
print 'Socket bind complete.'

server_socket.listen(10)

# add server socket object to the list of readable connections
SOCKET_LIST.append(server_socket)

print "Server started on port " + str(PORT) + '.'
print "Server started with ID = " + str(ID) + '.\n'

while 1:
    # check blink
    if blink_flag == True:
        blink()
    else:
        nblink()

    # send temp.txt
    if download_flag == True:
        line = file_tx.readline()
        if line == "":
            file_tx.close()
            download_flag = False
        else:
            broadcast(server_socket, sock, line + "\n")

    # get the list sockets which are ready to be read through select
    # 4th arg, time_out = 0 : poll and never block
    ready_to_read,ready_to_write,in_error =
        select.select(SOCKET_LIST,[],[],0)

    for sock in ready_to_read:
        # a new connection request received
        if sock == server_socket:
            conn, addr = server_socket.accept()
```

```
SOCKET_LIST.append(conn)
print "Client (%s, %s) connected.\n" % addr

# a message from a client, not a new connection
else:
    # process data received from client,
    try:
        # receiving data from the socket.
        data = sock.recv(RECV_BUFFER)
        if data:
            print("From Android: " + data)
            # processing data
            if data[:3] == "ID=":
                print "Received ID.\n"
                new_id = int(data[3:])
                print "Old ID ---> " + str(ID) + "."
                print "New ID ---> " + str(new_id) + ".\n"
                if ID == 0:
                    ID = new_id
                    file = open("id.txt", "w")
                    file.write(str(ID))
                    file.close()
                    broadcast(server_socket,sock,"ID fixed\n")
                    print "To (%s, %s): ID fixed!" % addr
                    print "The new ID for the sensor is ---> "
                    + str(ID) + ".\n"
                else:
                    if meas_flag:
                        broadcast(server_socket, sock, "MEAS\n")
                    else:
                        broadcast(server_socket, sock, "NMEAS\n")

                    print "To (%s, %s): I already have ID!" % addr
                    print("This sensor already has ID. The ID is
---> " + str(ID) + ".\n")
            else:
                if data[:4] == "MEAS":
                    start_meas = subprocess.Popen(['python',
                      'main_wired_rpi.py'])
                    meas_flag = True
```

```
        elif data[:5] == "NMEAS":
            start_meas.terminate()
            meas_flag = False
        elif data[:5] == "BLINK":
            blink_flag = True
        elif data[:6] == "NBLINK":
            blink_flag = False
        elif data[:5] == "ERASE":
            ID = 0
            file = open("id.txt", "w")
            file.write(str(ID))
            file.close()
        elif data[:8] == "DOWNLOAD":
            copyfile("/home/pi/measurments.txt",
                     "/home/pi/temp.txt")
            file_lines = 0
            file_tx = open("/home/pi/temp.txt")
            for line in file_tx.xreadlines():
                file_lines += 1
            file_tx.close()
            broadcast(server_socket, sock, "lines = " +
                      str(file_lines) + "\n")
            download_flag = True
            file_tx = open("/home/pi/temp.txt", "r")
        else:
            print "Received unknown data."
    else:
        # remove the socket that's broken
        if sock in SOCKET_LIST:
            SOCKET_LIST.remove(sock)

        # at this stage, no data means probably the connection
        # has been closed
        print(str(addr) + " is offline.")

    # exception
except:
    continue

server_socket.close()
```

```
def blink():
    global start, current
    current = millis()
    if start == 0:
        start = millis()
        GPIO.output(12, True)
    else:
        if (current - start) > 250 and (current - start) < 500:
            GPIO.output(12, False)
        elif (current - start) > 500:
            start = 0

def nblink():
    global start, current
    GPIO.output(12, False)
    current = 0
    start = 0

# answer messages to connected client
def broadcast(server_socket, sock, message):
    for socket in SOCKET_LIST:
        # send the message only to peer
        if socket == sock:
            try :
                socket.send(message)
            except :
                # broken socket connection
                socket.close()
                # broken socket, remove it
                if socket in SOCKET_LIST:
                    SOCKET_LIST.remove(socket)

    if __name__ == "__main__":
        sys.exit(chat_server())
```

V. AndroidManifest.xml

```

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

    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".OnlineOptionsActivity"
            android:label="Online Options"
            android:theme="@style/AppTheme"
            android:screenOrientation="portrait">
        </activity>

        <activity android:name=".RegisterTaskActivity"
            android:label="Register" android:theme="@style/AppTheme"
            android:screenOrientation="portrait">
        </activity>

        <activity android:name=".GraphicsTaskActivity"
            android:label="Graphics" android:theme="@style/AppTheme"
            android:screenOrientation="portrait">
        </activity>

        <activity android:name=".RegisteredSensorsActivity"
            android:label="Registered Sensors"
            android:theme="@style/AppTheme"
            android:screenOrientation="portrait">
        </activity>

        <activity android:name=".OfflineOptionsActivity"
            android:label="Offline Options"
            android:theme="@style/AppTheme"
            android:screenOrientation="portrait">
        </activity>
    </application>
</manifest>

```

VI. MainActivity.java – activity_main.xml

```

package com.example.scarlinpez.wineapp;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.os.AsyncTask;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.ProgressBar;
import android.widget.Toast;
import org.json.JSONObject;
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class MainActivity extends AppCompatActivity {

    Button btnScan;
    ProgressBar simpleProgressBar;
    ListView listViewIp;

    ArrayList<String> ipList;
    ArrayAdapter<String> adapter;
    String str_ip = "";

    // inter-activities variables
    String ip, name;

    // To Load regLog and regNAME
    SharedPreferences preferencesREG, preferencesNAME;

    // To save regLog and regNAME
    SharedPreferences.Editor editorREG, editorNAME;

    // map ip_address - flag_registered
    HashMap<String, Boolean> regLog = new HashMap<String, Boolean>();
    String key;
    Boolean value;

    // map ip_address - flag_registered
    HashMap<String, String> regNAME = new HashMap<String, String>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Layout components
        btnScan = findViewById(R.id.scan);
    }
}

```

```

listViewIp = findViewById(R.id.ipList);
simpleProgressBar = findViewById(R.id.simpleProgressBar);
simpleProgressBar.setVisibility(View.INVISIBLE);

// List of sensors
ipList = new ArrayList();
adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1,
    android.R.id.text1, ipList);
listViewIp.setAdapter(adapter);

// To Load regLog map values
regLog = loadMapBool(preferencesREG, "regLog", "log_map");

// To Load regLog map values
regNAME = loadMapStr(preferencesNAME, "regNAME", "name_map");

// add registered sensors to list view
if(regLog.isEmpty() == false){
    for (Map.Entry<String, Boolean> entry : regLog.entrySet()) {
        key = entry.getKey();
        value = entry.getValue();
        if(value == true){
            ipList.add(key + " - " + regNAME.get(key));
            adapter.notifyDataSetChanged();
        }
    }
}

// get access when click in item outside ScanIpTask
listViewIp.setClickable(true);
listViewIp.setOnItemClickListener(
    new AdapterView.OnItemClickListener() {

        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1,
            int position, long arg3) {

            Object o = listViewIp.getItemAtPosition(position);
            String[] parts = o.toString().split(" ");
            str_ip = parts[0];
            Intent intent =
                OnlineOptionsActivity.makeIntent(MainActivity.this);
            intent.putExtra("IP_AD",str_ip);
            startActivityForResult(intent, 1015);
        }
    });
}

// button click task
btnScan.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        new ScanIpTask().execute();
    }
});

public void clickOffline(View view) {
    if(regNAME.isEmpty() == true){
        Toast.makeText(MainActivity.this, "No sensor registered for now.
            Try scanning.", Toast.LENGTH_LONG).show();
    }
}

```

```

        else{
            Intent intent =
                RegisteredSensorsActivity.makeIntent(MainActivity.this);
            startActivity(intent);
        }
    }

@Override
protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    switch (requestCode) {
        case 1015:
            if (resultCode == Activity.RESULT_OK) {
                String wName = data.getStringExtra("wName");
                if(wName != null && wName.equals("") == false &&
                    wName.contains(":") == false){
                    ipList.set(ipList.indexOf(ip), ip + " - " + wName);
                    adapter.notifyDataSetChanged();
                }
                else if(wName != null && wName.contains(":") == true){
                    if(ipList.indexOf(ip+" - "+wName.split(" ")[1])!=-1){
                        ipList.set(ipList.indexOf(ip + " - " +
                            wName.split(" ")[1]), ip);
                        adapter.notifyDataSetChanged();
                    }
                }
                // Reload regNAME map values
                regNAME = loadMapStr(preferencesNAME, "regNAME",
                    "name_map");
            }
            break;
    }
}

private class ScanIpTask extends AsyncTask<Void, String, Void> implements
    com.example.scarlinlpez.wineapp.ScanIpTask {

    // Scan IP 192.168.43.2~192.168.43.254

    static final String subnet = "192.168.43.";
    static final int lower = 2;
    static final int upper = 254;
    static final int timeout = 250;

    @Override
    protected void onPreExecute() {
        btnScan.setClickable(false);
        btnScan.setText("SCANNING IP's... ");
        simpleProgressBar.getProgressDrawable().setColorFilter(Color.BLUE,
            android.graphics.PorterDuff.Mode.SRC_IN);
        simpleProgressBar.setMax(upper - 2);
        simpleProgressBar.setVisibility(View.VISIBLE);
    }

    @Override
    protected Void doInBackground(Void... params) {
        for (int i = lower; i <= upper; i++) {
            String host = subnet + i;
            simpleProgressBar.setProgress(i - 2);
            try {
                InetAddress inetAddress = InetAddress.getByName(host);
                if (inetAddress.isReachable(timeout)){
                    publishProgress(inetAddress.toString());
                }
            }
        }
    }
}

```

```

        }
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
return null;
}

@Override
protected void onProgressUpdate(String... values) {
    ip = values[0].replace("/", "");
    if(ipList.indexOf(ip) == -1 && regNAME.containsKey(ip) == false){
        ipList.add(ip);
        adapter.notifyDataSetChanged();
    }
}

@Override
protected void onPostExecute(Void aVoid) {
    btnScan.setText("Start scanning");
    btnScan.setClickable(true);
    simpleProgressBar.setVisibility(View.INVISIBLE);
    Toast.makeText(MainActivity.this, "Scanning done.",
        Toast.LENGTH_LONG).show();
}
}

private HashMap<String,Boolean> loadMapBool(SharedPreferences preferences,
    String name, String key){
    HashMap<String,Boolean> outputMap = new HashMap<String,Boolean>();
    preferences = getApplicationContext().getSharedPreferences(name,
        Context.MODE_PRIVATE);
    try{
        if (preferences != null){
            String jsonString = preferences.getString(key, (new
                JSONObject()).toString());
            JSONObject jsonObject = new JSONObject(jsonString);
            Iterator<String> keysItr = jsonObject.keys();
            while(keysItr.hasNext()) {
                String k = keysItr.next();
                Boolean value = (Boolean) jsonObject.get(k);
                outputMap.put(k, value);
            }
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return outputMap;
}

private HashMap<String,String> loadMapStr(SharedPreferences preferences,
    String name, String key){
    HashMap<String,String> outputMap = new HashMap<String,String>();
    preferences = getApplicationContext().getSharedPreferences(name,
        Context.MODE_PRIVATE);
    try{
        if (preferences != null){
            String jsonString = preferences.getString(key, (new
                JSONObject()).toString());
            JSONObject jsonObject = new JSONObject(jsonString);
            Iterator<String> keysItr = jsonObject.keys();
            while(keysItr.hasNext()) {

```

```
        String k = keysItr.next();
        String value = (String) jsonObject.get(k);
        outputMap.put(k, value);
    }
}
}catch(Exception e){
    e.printStackTrace();
}
return outputMap;
}

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/offline"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickOffline"
        android:text="Offline Options"/>

    <Button
        android:id="@+id/scan"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start scanning"/>

    <ProgressBar
        android:id="@+id/simpleProgressBar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>

    <ListView
        android:id="@+id/iplist"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

VII. OfflineOptionsActivity.java – activity_offline.xml

```

package com.example.scarlinpez.wineapp;

import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.Toast;
import com.google.firebaseio.database.ChildEventListener;
import com.google.firebaseio.database.DataSnapshot;
import com.google.firebaseio.database.DatabaseError;
import com.google.firebaseio.database.DatabaseReference;
import com.google.firebaseio.database.FirebaseDatabase;
import org.json.JSONObject;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

public class OfflineOptionsActivity extends AppCompatActivity{

    // APP Buttons
    Button info, graphData, eraseDataSensor;

    // Manage data from DB
    FirebaseDatabase database;
    DatabaseReference myRef;

    // To show sensor info
    ListView infoListView;
    ArrayList<String> infoList;
    ArrayAdapter<String> listAdapter;

    // map ip_address - id
    HashMap<String, Integer> sensors = new HashMap<String, Integer>();

    // To store/load sensors
    SharedPreferences preferencesSEN;

    // getting data from RegisteredSensorsActivity
    Intent intent;
    String ipadress;

    // sensor data
    String name, reference, id, ip, creation;
    String kname, kreference, kid, kip, kcreation;

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

        // getting data from MainActivity
        intent = getIntent();
        ipadress = intent.getStringExtra("IP_AD");
    }
}

```

```

// Layout components
info = findViewById(R.id.info);
graphData = findViewById(R.id.graphData2);
eraseDataSensor = findViewById(R.id.eraseDataSensor2);

// To Load sensors map values
sensors = loadMapInt(preferencesSEN, "sensors", "sen_map");

// database instance to get/set data
database = FirebaseDatabase.getInstance();
myRef = database.getReference("Sensors").child("Sensor_" +
    sensors.get(ipaddress).toString());

// setup list view for sensor information
infoListView = findViewById( R.id.infoList);
infoList = new ArrayList();
listAdapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, android.R.id.text1, infoList);
infoListView.setAdapter(listAdapter);

myRef.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String s) {
        String key = dataSnapshot.getKey();
        if(key.equals("Creation")){
            kcreation = key;
            creation = dataSnapshot.getValue(String.class);
        }else if(key.equals("ID")){
            kid = key;
            id = dataSnapshot.getValue(String.class);
        }else if(key.equals("IP address")){
            kip = key;
            ip = dataSnapshot.getValue(String.class);
        }else if(key.equals("Name")){
            kname = key;
            name = dataSnapshot.getValue(String.class);
        }else if(key.equals("Reference")){
            kreference = key;
            reference = dataSnapshot.getValue(String.class);
        }
    }
    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String s) {
    }
    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {
    }
    @Override
    public void onChildMoved(DataSnapshot dataSnapshot, String s) {
    }
    @Override
    public void onCancelled(DatabaseError databaseError) {
    });
});

public void clickInfo(View view) {
    listAdapter.add(kname + ": " + name);
    listAdapter.notifyDataSetChanged();

    listAdapter.add(kreference + ": " + reference);
    listAdapter.notifyDataSetChanged();
}

```

```

listAdapter.add(kip + ":" + ip);
listAdapter.notifyDataSetChanged();

listAdapter.add(kid + ":" + id);
listAdapter.notifyDataSetChanged();

listAdapter.add(kcreation + ":" + creation);
listAdapter.notifyDataSetChanged();
}

public void clickGraph(View view) {
    Integer id_s = sensors.get(ipadress);
    Intent intentGraph =
        GraphicsTaskActivity.makeIntent(OfflineOptionsActivity.this);
    intentGraph.putExtra("ID_S", id_s);
    startActivity(intentGraph);
}

public void clickEraseData(View view) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    final String sensor = "Sensor_" + sensors.get(ipadress).toString();

    builder.setMessage("Do you want to erase sensor - " + name + "- data
from Firebase?") .setTitle("Erase Sensor Data")
    .setPositiveButton("Yes", new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                // Erase data in DB
                myRef = database.getReference("Sensors")
                    .child(sensor).child("Temperatures");
                myRef.removeValue();
                myRef = database.getReference("Sensors")
                    .child(sensor).child("Humidities");
                myRef.removeValue();
                Toast.makeText(OfflineOptionsActivity.this, "Sensor
data erased.", Toast.LENGTH_SHORT).show();
                dialog.cancel();
            }
        })
    .setNegativeButton("No", new
        DialogInterface.OnClickListener(){
            public void onClick(DialogInterface dialog, int id) {
                dialog.cancel();
            }
        });
    builder.create().show();
}

public static Intent makeIntent(Context context) {
    return new Intent(context, OfflineOptionsActivity.class);
}

private HashMap<String, Integer> loadMapInt(SharedPreferences preferences,
    String name, String key){
    HashMap<String, Integer> outputMap = new HashMap<String, Integer>();
    preferences = getApplicationContext().getSharedPreferences(name,
        Context.MODE_PRIVATE);
    try{
        if (preferences != null){
            String jsonString = preferences.getString(key, (new
                JSONObject()).toString());
            JSONObject jsonObject = new JSONObject(jsonString);
            Iterator<String> keysItr = jsonObject.keys();

```

```
        while(keysItr.hasNext()) {
            String k = keysItr.next();
            Integer value = (Integer) jsonObject.get(k);
            outputMap.put(k, value);
        }
    }catch(Exception e){
        e.printStackTrace();
}
return outputMap;
}

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".OfflineOptionsActivity">

    <Button
        android:id="@+id/info"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickInfo"
        android:text="Sensor Information"/>

    <Button
        android:id="@+id/graphData2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickGraph"
        android:text="Charts"/>

    <Button
        android:id="@+id/eraseDataSensor2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickEraseData"
        android:text="Erase Sensor Data"/>

    <ListView
        android:id="@+id/infoList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="50dp" />

</LinearLayout>
```

VIII. RegisteredSensorsActivity.java – activity_sensors.xml

```
package com.example.scarlinlpez.wineapp;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import org.json.JSONObject;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class RegisteredSensorsActivity extends AppCompatActivity {

    // To show sensor info
    ListView sensorsListView;
    ArrayList<String> sensorsList;
    ArrayAdapter<String> sensorsAdapter;

    // map ip_address - flag_registered
    HashMap<String, Boolean> regLog = new HashMap<String, Boolean>();
    String key;
    Boolean value;

    // map ip_address - flag_registered
    HashMap<String, String> regNAME = new HashMap<String, String>();

    // To Load regLog
    SharedPreferences preferencesNAME;

    // To Load regLog
    SharedPreferences preferencesREG;

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

        // setup list view for sensors
        sensorsListView = findViewById( R.id.sensorsList );
        sensorsList = new ArrayList();
        sensorsAdapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, android.R.id.text1, sensorsList);
        sensorsListView.setAdapter(sensorsAdapter);

        // To Load regLog map values
        regLog = loadMapBool(preferencesREG, "regLog", "log_map");

        // To Load regLog map values
        regNAME = loadMapStr(preferencesNAME, "regNAME", "name_map");
        // add registered sensors to list view
        if(regLog.isEmpty() == false){
            for (Map.Entry<String, Boolean> entry : regLog.entrySet()) {
                key = entry.getKey();
                value = entry.getValue();
                if(value == true){


```

```

        sensorsAdapter.add(key + " - " + regNAME.get(key));
        sensorsAdapter.notifyDataSetChanged();
    }
}

// get access when click in item outside ScanIpTask
sensorsListView.setOnItemClickListener(new
    AdapterView.OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> arg0, View arg1,
        int position, long arg3) {
        Object o = sensorsListView.getItemAtPosition(position);
        String[] parts = o.toString().split(" ");
        String str_ip = parts[0];
        Intent intent = OfflineOptionsActivity
            .makeIntent(RegisteredSensorsActivity.this);
        intent.putExtra("IP_AD",str_ip);
        startActivity(intent);
    }
});

private HashMap<String,Boolean> loadMapBool(SharedPreferences preferences,
    String name, String key){
    HashMap<String,Boolean> outputMap = new HashMap<String,Boolean>();
    preferences = getApplicationContext().getSharedPreferences(name,
        Context.MODE_PRIVATE);
    try{
        if (preferences != null){
            String jsonString = preferences.getString(key, (new
                JSONObject()).toString());
            JSONObject jsonObject = new JSONObject(jsonString);
            Iterator<String> keysItr = jsonObject.keys();
            while(keysItr.hasNext()) {
                String k = keysItr.next();
                Boolean value = (Boolean) jsonObject.get(k);
                outputMap.put(k, value);
            }
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return outputMap;
}

private HashMap<String,String> loadMapStr(SharedPreferences preferences,
    String name, String key){
    HashMap<String,String> outputMap = new HashMap<String,String>();
    preferences = getApplicationContext().getSharedPreferences(name,
        Context.MODE_PRIVATE);
    try{
        if (preferences != null){
            String jsonString = preferences.getString(key, (new
                JSONObject()).toString());
            JSONObject jsonObject = new JSONObject(jsonString);
            Iterator<String> keysItr = jsonObject.keys();
            while(keysItr.hasNext()) {
                String k = keysItr.next();
                String value = (String) jsonObject.get(k);
                outputMap.put(k, value);
            }
        }
    }
}

```

```
        }catch(Exception e){
            e.printStackTrace();
        }
        return outputMap;
    }

    public static Intent makeIntent(Context context) {
        return new Intent(context, RegisteredSensorsActivity.class);
    }
}

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".RegisteredSensorsActivity">

    <ListView
        android:id="@+id/sensorsList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="50dp" />

</LinearLayout>
```

IX. OnlineOptionsActivity.java – activity_online.xml

```

package com.example.scarlinpez.wineapp;

import android.app.Activity;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.graphics.PorterDuff;
import android.os.Bundle;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.Toast;
import com.google.firebaseio.database.DatabaseReference;
import com.google.firebaseio.database.FirebaseDatabase;
import org.json.JSONObject;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketAddress;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

public class OnlineOptionsActivity extends AppCompatActivity{

    // Network variables
    String ipadress;
    Intent intent;
    SocketAddress server;
    Socket client, client2;
    Integer port = 9009, id_sensor;

    // Network I/O data variables
    OutputStream outtoserver;
    OutputStreamWriter osw;
    BufferedWriter bw;
    BufferedReader br;
    BufferedInputStream is;
    BufferedReader br2;
    BufferedInputStream is2;

    // task threads
    Thread locateThread, idThread, measThread, eraseThread, downloadThread,
        downloadThread2;

    // Thread control variables
    String msgLocate, msgMeas;
    Boolean send_data_locate, send_data_meas, erase, download, lines_received;
}

```

```
Integer num, lines;

// APP Buttons
Button register, meas, locate, getData, graphData, eraseDataSensor,
eraseSensor;

// To store/Load sensors, regLog, regID, regName maps
SharedPreferences preferencesSEN, preferencesREG, preferencesID,
preferencesNAME;
SharedPreferences.Editor editorSEN, editorREG, editorID, editorNAME;

// map ip_address - id
HashMap<String, Integer> sensors = new HashMap<String, Integer>();

// map ip_address - flag_registered
HashMap<String, Boolean> regLog = new HashMap<String, Boolean>();

// map id - flag_used
HashMap<String, Boolean> regID = new HashMap<String, Boolean>();

// map ipaddress - sensor_name
HashMap<String, String> regNAME = new HashMap<String, String>();

// Manage data from DB
 FirebaseDatabase database;
 DatabaseReference myRef;

// inter-activities variables
String wName, flag;

// to show data - info
ListView mainListView;
ArrayList<String> infodataList;
ArrayAdapter<String> infodataAdapter;

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

    // getting data from MainActivity
    intent = getIntent();
    ipadress = intent.getStringExtra("IP_AD");

    // Layout components
    register = findViewById(R.id.register);
    locate = findViewById(R.id.Locate);
    getData = findViewById(R.id.getData);
    graphData = findViewById(R.id.graphData);
    eraseDataSensor = findViewById(R.id.eraseDataSensor);
    eraseSensor = findViewById(R.id.eraseSensor);
    meas = findViewById(R.id.meas);

    // SERVER AND SOCKET
    server = new InetSocketAddress(ipadress, port);
    client = new Socket();
    client2 = new Socket();

    // to manage Locate task
    msgLocate = "NBLINK";
    send_data_locate = false;

    // to manage meas task
    msgMeas = "NMEAS";
```

```
send_data_meas = false;

// to manage erase sensor task
erase = false;

// to manage download file task
download = false;
lines_received = false;
num = 0;
lines = 0;

// database instance to get/set data
database = FirebaseDatabase.getInstance();

// To Load sensors map values
sensors = loadMapInt(preferencesSEN, "sensors", "sen_map");

// To Load regLog map values
regLog = loadMapBool(preferencesREG, "regLog", "log_map");

// To Load regID map values
regID = loadMapBool(preferencesID, "regID", "id_map");

// To Load regNAME map values
regNAME = loadMapStr(preferencesNAME, "regNAME", "name_map");

// List of data-information
mainListView = (ListView) findViewById(R.id.infodataList);
infodataList = new ArrayList();
infodataAdapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, android.R.id.text1,
    infodataList);
mainListView.setAdapter(infodataAdapter);

// check if ip registered and update button
if(regLog.isEmpty() == false){
    if(regLog.containsKey(ipadress) == true &&
        regLog.get(ipadress) == true){
        register.setText("Register Done!");
        register.getBackground().setColorFilter(Color.GREEN,
            PorterDuff.Mode.SRC_ATOP);
    }
    else{
        register.setText("Register");
        register.getBackground().clearColorFilter();
    }
}

// ID sync thread
idThread = new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            //server = new InetSocketAddress(ipadress,port);
            client.connect(server);
            OnlineOptionsActivity.this.runOnUiThread(new Runnable() {
                public void run() {
                    if(client.isConnected()) {
                        Toast.makeText(OnlineOptionsActivity.this,
                            "Connected!", Toast.LENGTH_SHORT).show();
                    }
                }
            });
        }
    }
});
```

```
// in-out socket data
outtoserver = client.getOutputStream();
osw = new OutputStreamWriter(outtoserver);
bw = new BufferedWriter(osw);

//get and send id value
if(regID.isEmpty()){
    // initialize default values
    for(Integer i = 1; i <= 30; i++){
        regID.put(i.toString(), false);
    }
}
id_sensor = getID();

if(id_sensor != 0){
    bw.write("ID=" + String.valueOf(id_sensor) + "\n");
    bw.flush();

    is = new BufferedInputStream(client.getInputStream());
    br = new BufferedReader(new InputStreamReader(is));

    try{
        for();{
            final String message = br.readLine();
            if(message.length() != 0){
                OnlineOptionsActivity.this
                    .runOnUiThread(new Runnable() {
                        public void run() {
                            if(message.contentEquals("MEAS")){
                                msgMeas = "MEAS";
                                meas.setBackground()
                                    .setColorFilter(Color.GREEN,
                                PorterDuff.Mode.SRC_ATOP);
                            }

                            else if
                                (message.contentEquals("NMEAS")){
                                msgMeas = "NMEAS";
                                meas.setBackground()
                                    .clearColorFilter();
                            }
                            else if(message.contentEquals("ID
                                fixed!")){
                                meas.setBackground()
                                    .clearColorFilter();
                                // Store in dictionary pair of
                                ip_address - id
                                sensors.put(ipadress,
                                id_sensor);
                                saveMapInt(sensors,
                                preferencesSEN,
                                editorSEN,
                                "sensors", "sen_map");
                                // update ip flag in regid
                                regID.put(
                                id_sensor.toString(), true);
                                saveMapBool(regID,
                                preferencesID,
                                editorID,
                                "regID", "id_map");
                                // Store in dictionary pair of
                                ip_address - flag_registered
                                regLog.put(ipadress, false);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

        // display some sensor information
        infodataList.add("Sensor IP = " +
                          ipaddress);
        infodataAdapter
            .notifyDataSetChanged();
        if(regNAME.containsKey(ipaddress)){
            infodataList.add("Sensor Name
                            = " + regNAME
                            .get(ipaddress));
            infodataAdapter
                .notifyDataSetChanged();
        }
        infodataList.add("Sensor ID = " +
                         sensors.get(ipaddress));
        infodataAdapter
            .notifyDataSetChanged();
        if(regLog.containsKey(ipaddress)){
            if(regLog.get(ipaddress) ==
               true){
                infodataList.add("Sensor
                                registered");
                infodataAdapter
                    .notifyDataSetChanged();
            }
            else{
                infodataList.add("Sensor
                                not registered yet");
                infodataAdapter
                    .notifyDataSetChanged();
            }
        }
    });
}
catch(IOException e)
{
    e.printStackTrace();
}
}
else{
    Toast.makeText(OnlineOptionsActivity.this, "No more
        available ID's. Please erase any registered sensor.",
        Toast.LENGTH_SHORT).show();
    onBackPressed();
}
}
catch (UnknownHostException e2) {
    e2.printStackTrace();
}
catch (IOException e1) {
    e1.printStackTrace();
    Log.d("Time out", "Time");
}
}
);
idThread.start();

```

```

// Meas button thread
measThread = new Thread(new Runnable() {
    @Override
    public void run(){
        while(!measThread.isInterrupted()){
            if(send_data_meas){
                try {
                    // get output stream from socket
                    outtoserver = client.getOutputStream();
                    // write in output stream
                    osw = new OutputStreamWriter(outtoserver);
                    // write in output buffer and socket
                    bw = new BufferedWriter(osw);
                    bw.write(msgMeas + "\n");
                    bw.flush();
                    send_data_meas = false;
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
});
measThread.start();

// Locate button thread
locateThread = new Thread(new Runnable() {
    @Override
    public void run(){
        while(!locateThread.isInterrupted()){
            if(send_data_locate){
                try {
                    // get output stream from socket
                    outtoserver = client.getOutputStream();
                    // write in output stream
                    osw = new OutputStreamWriter(outtoserver);
                    // write in output buffer
                    bw = new BufferedWriter(osw);
                    // write in socket
                    bw.write(msgLocate + "\n");
                    bw.flush();
                    send_data_locate = false;
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
});
locateThread.start();

// Send download flag and receive file length thread
downloadThread = new Thread(new Runnable() {
    @Override
    public void run(){
        try {
            client2.connect(server);
            try{
                for(;;){
                    while(!download);
                    // get output stream from socket
                    outtoserver = client2.getOutputStream();
                    // write in output stream
                    osw = new OutputStreamWriter(outtoserver);

```

```
// write in output buffer
bw = new BufferedWriter(osw);
// write in socket
bw.write("DOWNLOAD" + "\n");
bw.flush();
download = false;
is2 = new
    BufferedInputStream(client2.getInputStream());
br2 = new BufferedReader(new
    InputStreamReader(is2));
final String message = br2.readLine();
if(message.length() != 0){
    OnlineOptionsActivity.this.runOnUiThread(new
        Runnable() {
            public void run() {
                if(message.contains("lines =")){
                    String[] parts=message.split(" ");
                    lines =
                        Integer.parseInt(parts[2]);
                    infodataList.add("File lines = "
                        + parts[2]);
                    infodataAdapter
                        .notifyDataSetChanged();
                    lines_received = true;
                }
            }
        });
}
}
catch(IOException e)
{
    e.printStackTrace();
}
}
catch (UnknownHostException e2) {
    e2.printStackTrace();
} catch (IOException e1) {
    e1.printStackTrace();
}
}
});
downloadThread.start();

// Download data thread
downloadThread2 = new Thread(new Runnable() {
    @Override
    public void run(){
        try{
            while(!downloadThread2.isInterrupted()){
                while(!lines_received);
                is2 = new
                    BufferedInputStream(client2.getInputStream());
                br2 = new BufferedReader(new InputStreamReader(is2));
                try{
                    for(;;){
                        final String message = br2.readLine();
                        if(message.length() != 0){
                            lines--;
                            OnlineOptionsActivity.this
                                .runOnUiThread(new Runnable() {
                                    public void run() {
                                        String[] parts=message.split(" ");
                                        Integer temp =
```

```

        Integer.parseInt(parts[5]);
        Integer hum =
            Integer.parseInt(parts[7]);
        num++;
        infodataList.add(num.toString() +
            ".      temp = " + temp.toString()
            + " --- hum = " + hum.toString());
        infodataAdapter
            .notifyDataSetChanged();

        // Upload data to database
        String sensor = "Sensor_" +
            sensors.get(ipadress).toString();
        myRef = database
            .getReference("Sensors")
            .child(sensor);
        myRef.child("Temperatures")
            .push()
            .setValue(temp.toString());
        myRef.child("Humidities")
            .push()
            .setValue(hum.toString());
    }
}
if(lines == 0){
    lines_received = false;
    break;
}
}
}
}
catch(IOException e)
{
    e.printStackTrace();
}
}
} catch (IOException e) {
    e.printStackTrace();
}
}
}
downloadThread2.start();

// Erase Sensor button thread
eraseThread = new Thread(new Runnable() {
    @Override
    public void run(){
        while(!eraseThread.isInterrupted()){
            if(erase){
                try {
                    // get output stream from socket
                    outtoserver = client.getOutputStream();
                    // write in output stream
                    osw = new OutputStreamWriter(outtoserver);
                    // write in output buffer
                    bw = new BufferedWriter(osw);
                    // write in socket
                    bw.write("ERASE" + "\n");
                    bw.flush();
                    erase = false;
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
});

```

```

        }
    });
eraseThread.start();
}

public void clickLocate(View view){
    if(msgLocate.equals("BLINK")){
        locate.getBackground().clearColorFilter();
        msgLocate = "NBLINK";
    }
    else if(msgLocate.equals("NBLINK")){
        locate.getBackground().setColorFilter(Color.GREEN,
            PorterDuff.Mode.SRC_ATOP);
        msgLocate = "BLINK";
    }
    send_data_locate = true;
}

public void clickRegister(View view){
    if(regLog.get(ipadress) == true){
        // This ip is already registered
        Toast.makeText(OnlineOptionsActivity.this, "Sensor already
            registered.", Toast.LENGTH_SHORT).show();
    }
    else{
        // This ip is not registered yet
        Integer id_s = sensors.get(ipadress);
        Intent intentRegister =
            RegisterTaskActivity.makeIntent(OnlineOptionsActivity.this);
        intentRegister.putExtra("IP_AD", ipadress);
        intentRegister.putExtra("ID_S", id_s);
        startActivityForResult(intentRegister, 1014);
    }
}

public void clickMeas(View view) {
    if(regLog.get(ipadress) == true){
        if(msgMeas.equals("MEAS")){
            msgMeas = "NMEAS";
            meas.getBackground().clearColorFilter();
            Toast.makeText(OnlineOptionsActivity.this, "Measure process
                stopped.", Toast.LENGTH_SHORT).show();
        }
        else if(msgMeas.equals("NMEAS")){
            msgMeas = "MEAS";
            meas.getBackground().setColorFilter(Color.GREEN,
                PorterDuff.Mode.SRC_ATOP);
            Toast.makeText(OnlineOptionsActivity.this, "Measure process
                started.", Toast.LENGTH_SHORT).show();
        }
        send_data_meas = true;
    }else Toast.makeText(OnlineOptionsActivity.this, "Register sensor
        before start measures.", Toast.LENGTH_SHORT).show();
}

public void clickGet(View view) {
    if(regLog.get(ipadress) == true){
        download = true;
    }
    else Toast.makeText(OnlineOptionsActivity.this, "Register sensor
        before get data.", Toast.LENGTH_SHORT).show();
}

```

```
public void clickGraph(View view) {
    if(regLog.get(ipadress) == true){
        Integer id_s = sensors.get(ipadress);
        Intent intentGraph =
            GraphicsTaskActivity.makeIntent(OnlineOptionsActivity.this);
        intentGraph.putExtra("ID_S", id_s);
        startActivity(intentGraph);
    }
    else Toast.makeText(OnlineOptionsActivity.this, "Register sensor
        before graph data.", Toast.LENGTH_SHORT).show();
}

public void clickEraseData(View view) {

    if(regLog.get(ipadress) == true){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        final String sensor = "Sensor_"
            + sensors.get(ipadress).toString();

        builder.setMessage("Do you want to erase sensor -"
            + regNAME.get(ipadress) + "- data from Firebase?")
            .setTitle("Erase Sensor Data")
            .setPositiveButton("Yes", new
                DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        // Erase data in DB
                        myRef = database.getReference("Sensors")
                            .child(sensor).child("Temperatures");
                        myRef.removeValue();
                        myRef = database.getReference("Sensors")
                            .child(sensor).child("Humidities");
                        myRef.removeValue();
                        Toast.makeText(OnlineOptionsActivity.this, "Sensor
                            data erased.", Toast.LENGTH_SHORT).show();
                        dialog.cancel();
                    }
                })
            .setNegativeButton("No", new
                DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        dialog.cancel();
                    }
                });
        builder.create().show();
    }
    else Toast.makeText(OnlineOptionsActivity.this, "Sensor not registered
        yet.", Toast.LENGTH_SHORT).show();
}

public void clickEraseSensor(View view) {

    if(regLog.get(ipadress) == true){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        String sensor = "Sensor_" + sensors.get(ipadress).toString();

        builder.setMessage("Do you want to erase sensor -"
            + regNAME.get(ipadress) + "- from Firebase?")
            .setTitle("Erase Sensor")
            .setPositiveButton("Yes", new
                DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
```

```
// Erase data in DB
String reference = "Sensor_"
    + sensors.get(ipadress).toString();
myRef = database.getReference("Sensors")
    .child(reference);
myRef.removeValue();
// update usage of current sensor id
regID.put(sensors.get(ipadress)
    .toString(), false);
// Remove item from sensor, regLog and regNAME
// lists
sensors.remove(ipadress);
regLog.remove(ipadress);
wName = ":" + regNAME.get(ipadress);
regNAME.remove(ipadress);
// notify raspberry that may initialize ID
erase = true;
Toast.makeText(OnlineOptionsActivity.this, "Sensor
erased.", Toast.LENGTH_SHORT).show();
dialog.cancel();
onBackPressed();
}
})
.setNegativeButton("No", new
    DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });
builder.create().show();
}
else Toast.makeText(OnlineOptionsActivity.this, "Sensor not registered
yet.", Toast.LENGTH_SHORT).show();
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
switch(requestCode){
    case 1014:
        if(resultCode == Activity.RESULT_OK){
            flag = data.getStringExtra("flag");
            wName = data.getStringExtra("wName");
            if(flag.equals("true")){
                regLog.put(ipadress, true);
                saveMapBool(regLog, preferencesREG, editorREG,
                    "regLog", "log_map");
                regNAME.put(ipadress, wName);
                saveMapStr(regNAME, preferencesNAME, editorNAME,
                    "regNAME", "name_map");
                register.setText("Register Done!");
                register.getBackground().setColorFilter(Color.GREEN,
                    PorterDuff.Mode.SRC_ATOP);
            }
            else if(flag.equals("false")){
                regLog.put(ipadress, false);
                saveMapBool(regLog, preferencesREG, editorREG,
                    "regLog", "log_map");
            }
        }
    }
break;
}
```

```
// display some sensor data
infodataList.clear();
infodataList.add("Sensor IP = " + ipadress);
infodataAdapter.notifyDataSetChanged();
if(regNAME.containsKey(ipadress)){
    infodataList.add("Sensor Name = " + regNAME.get(ipadress));
    infodataAdapter.notifyDataSetChanged();
}
infodataList.add("Sensor ID = " + sensors.get(ipadress));
infodataAdapter.notifyDataSetChanged();
if(regLog.containsKey(ipadress)){
    if(regLog.get(ipadress) == true){
        infodataList.add("Sensor registered");
        infodataAdapter.notifyDataSetChanged();
    }
    else{
        infodataList.add("Sensor not registered yet");
        infodataAdapter.notifyDataSetChanged();
    }
}
}

private void saveMapBool(HashMap<String,Boolean> inputMap,
    SharedPreferences preferences, SharedPreferences.Editor editor,
    String name, String key){
    preferences = getApplicationContext().getSharedPreferences(name,
        Context.MODE_PRIVATE);
    if (preferences != null){
        JSONObject jsonObject = new JSONObject(inputMap);
        String jsonString = jsonObject.toString();
        editor = preferences.edit();
        editor.clear();
        editor.remove(key).apply();
        editor.putString(key, jsonString);
        editor.apply();
    }
}

private HashMap<String,Boolean> loadMapBool(SharedPreferences preferences,
    String name, String key){
    HashMap<String,Boolean> outputMap = new HashMap<String,Boolean>();
    preferences = getApplicationContext().getSharedPreferences(name,
        Context.MODE_PRIVATE);
    try{
        if (preferences != null){
            String jsonString = preferences.getString(key, (new
                JSONObject()).toString());
            JSONObject jsonObject = new JSONObject(jsonString);
            Iterator<String> keysItr = jsonObject.keys();
            while(keysItr.hasNext()) {
                String k = keysItr.next();
                Boolean value = (Boolean) jsonObject.get(k);
                outputMap.put(k, value);
            }
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return outputMap;
}
```

```
private void saveMapStr(HashMap<String, String> inputMap, SharedPreferences preferences, SharedPreferences.Editor editor, String name, String key){  
    preferences = getApplicationContext().getSharedPreferences(name,  
        Context.MODE_PRIVATE);  
    if (preferences != null){  
        JSONObject jsonObject = new JSONObject(inputMap);  
        String jsonString = jsonObject.toString();  
        editor = preferences.edit();  
        editor.clear();  
        editor.remove(key).apply();  
        editor.putString(key, jsonString);  
        editor.apply();  
    }  
}  
  
private HashMap<String, String> loadMapStr(SharedPreferences preferences,  
    String name, String key){  
    HashMap<String, String> outputMap = new HashMap<String, String>();  
    preferences = getApplicationContext().getSharedPreferences(name,  
        Context.MODE_PRIVATE);  
    try{  
        if (preferences != null){  
            String jsonString = preferences.getString(key, (new  
                JSONObject()).toString());  
            JSONObject jsonObject = new JSONObject(jsonString);  
            Iterator<String> keysItr = jsonObject.keys();  
            while(keysItr.hasNext()) {  
                String k = keysItr.next();  
                String value = (String) jsonObject.get(k);  
                outputMap.put(k, value);  
            }  
        }  
    }catch(Exception e){  
        e.printStackTrace();  
    }  
    return outputMap;  
}  
  
private void saveMapInt(HashMap<String, Integer> inputMap, SharedPreferences preferences, SharedPreferences.Editor editor, String name, String key){  
    preferences = getApplicationContext().getSharedPreferences(name,  
        Context.MODE_PRIVATE);  
    if (preferences != null){  
        JSONObject jsonObject = new JSONObject(inputMap);  
        String jsonString = jsonObject.toString();  
        editor = preferences.edit();  
        editor.clear();  
        editor.remove(key).apply();  
        editor.putString(key, jsonString);  
        editor.apply();  
    }  
}  
  
private HashMap<String, Integer> loadMapInt(SharedPreferences preferences,  
    String name, String key){  
    HashMap<String, Integer> outputMap = new HashMap<String, Integer>();  
    preferences = getApplicationContext().getSharedPreferences(name,  
        Context.MODE_PRIVATE);  
    try{  
        if (preferences != null){  
            String jsonString = preferences.getString(key, (new  
                JSONObject()).toString());  
            JSONObject jsonObject = new JSONObject(jsonString);  
            Iterator<String> keysItr = jsonObject.keys();  
        }  
    }  
}
```

```
        while(keysItr.hasNext()) {
            String k = keysItr.next();
            Integer value = (Integer) jsonObject.get(k);
            outputMap.put(k, value);
        }
    }catch(Exception e){
        e.printStackTrace();
}
return outputMap;
}

public Integer getID(){
    Integer id = 0;
    for(Integer i = 1; i <= 30; i++){
        if(regID.get(i.toString()) == false && id == 0){
            id = i;
        }
    }
    return id;
}

@Override
public void onBackPressed() {
    saveMapInt(sensors, preferencesSEN, editorSEN, "sensors", "sen_map");
    saveMapBool(regLog, preferencesREG, editorREG, "regLog", "log_map");
    saveMapBool(regID, preferencesID, editorID, "regID", "id_map");
    saveMapStr(regNAME, preferencesNAME, editorNAME, "regNAME",
               "name_map");

    Intent intentBack = new Intent();
    intentBack.putExtra("wName", wName);
    setResult(Activity.RESULT_OK, intentBack);

    idThread.interrupt();
    measThread.interrupt();
    locateThread.interrupt();
    eraseThread.interrupt();
    downloadThread.interrupt();
    downloadThread2.interrupt();
    try {
        client.close();
        client2.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    this.finish();
}

public static Intent makeIntent(Context context) {
    return new Intent(context, OnlineOptionsActivity.class);
}
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".OnlineOptionsActivity">
    <Button
        android:id="@+id/register"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickRegister"
        android:text="Register Sensor"/>
    <Button
        android:id="@+id/meas"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickMeas"
        android:text="Start/Stop Meas"/>
    <Button
        android:id="@+id/locate"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickLocate"
        android:text="Locate Sensor"/>
    <Button
        android:id="@+id/getData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickGet"
        android:text="Get Data"/>
    <Button
        android:id="@+id/graphData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickGraph"
        android:text="Charts"/>
    <Button
        android:id="@+id/eraseDataSensor"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickEraseData"
        android:text="Erase Sensor Data"/>
    <Button
        android:id="@+id/eraseSensor"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="clickEraseSensor"
        android:text="Erase Sensor"/>
    <ListView
        android:id="@+id/infodataList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="50dp" />
</LinearLayout>
```

X. RegisterTaskActivity.java – activity_register.xml

```

package com.example.scarlinpez.wineapp;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import com.google.firebaseio.database.DatabaseReference;
import com.google.firebaseio.database.FirebaseDatabase;
import java.util.Date;

public class RegisterTaskActivity extends AppCompatActivity{

    // Layout components of RegisterTaskActivity
    Button register, cancel;
    TextView wName, wReference;

    // To manage firebase database
    FirebaseDatabase database;
    DatabaseReference myRef;
    Date date;

    // To get input data from OnLineOptionsActivity
    Intent intent;
    Integer id_sensor;
    String ipadress;

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

        // Get data from OnLineOptionsActivity
        intent = getIntent();
        ipadress = intent.getStringExtra("IP_AD");
        id_sensor = intent.getIntExtra("ID_S", 0);

        // Initialize layout components
        register = findViewById(R.id.bRegister);
        cancel = findViewById(R.id.bCancel);
        wName = findViewById(R.id.wineName);
        wReference = findViewById(R.id.reference);

        // database instance to get/set data
        database = FirebaseDatabase.getInstance();
    }

    public void clickRegis(View v){
        if(wName.getText().toString().equals("") &
           wReference.getText().toString().equals ""){
            Toast.makeText(RegisterTaskActivity.this, "Enter Wine Name and
                Reference.", Toast.LENGTH_SHORT).show();
        }
        else if(wName.getText().toString().equals ""){
            Toast.makeText(RegisterTaskActivity.this, "Enter Wine Name.",
```

```
        Toast.LENGTH_SHORT).show();
    }
    else if(wReference.getText().toString().equals ""){
        Toast.makeText(RegisterTaskActivity.this, "Enter Wine Reference.",  

            Toast.LENGTH_SHORT).show();
    }
    else{
        date = new Date();
        String sensor = "Sensor_" + id_sensor.toString();

        myRef = database.getReference("Sensors").child(sensor);
        myRef.child("Name").setValue(wName.getText().toString());
        myRef.child("Reference")
            .setValue(wReference.getText().toString());
        myRef.child("IP address").setValue(ipadress);
        myRef.child("ID").setValue(id_sensor.toString());
        myRef.child("Creation").setValue(date.toString());

        Intent intent = new Intent();
        intent.putExtra("flag", "true");
        intent.putExtra("wName", wName.getText().toString());
        setResult(Activity.RESULT_OK, intent);

        Toast.makeText(RegisterTaskActivity.this, "Sensor registered.",  

            Toast.LENGTH_SHORT).show();

        this.finish();
    }
}

public void clickCancel(View v){
    Intent intent = new Intent();
    intent.putExtra("flag", "false");
    intent.putExtra("wName", "");
    setResult(Activity.RESULT_OK, intent);

    this.finish();
}

public static Intent makeIntent(Context context) {
    return new Intent(context, RegisterTaskActivity.class);
}
```

```
<?xml version = "1.0" encoding = "utf-8"?>
<RelativeLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    xmlns:tools = "http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height = "match_parent"
    tools:context = ".RegisterTaskActivity">
    <EditText
        android:id = "@+id/wineName"
        android:hint = "Enter wine name"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:focusable = "true"
        android:layout_marginTop = "46dp"
        android:layout_alignParentLeft = "true"
        android:layout_alignParentStart = "true"
        android:layout_alignParentRight = "true"
        android:layout_alignParentEnd = "true" />
    <EditText
        android:id="@+id/reference"
        android:hint="Enter wine reference"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_below="@+id/wineName"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignRight="@+id/wineName"
        android:layout_alignEnd="@+id/wineName" />
    <Button
        android:id="@+id/bRegister"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:onClick="clickRegis"
        android:text="Register"
        android:layout_below="@+id/reference" />
    <Button
        android:id="@+id/bCancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:onClick="clickCancel"
        android:text="Cancel"
        android:layout_below="@+id/bRegister" />
</RelativeLayout>
```

XI. GraphicsTaskActivity.java – activity_graphics.xml

```
package com.example.scarlinpez.wineapp;

import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.highlight.Highlight;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
import com.github.mikephil.charting.listener.ChartTouchListener;
import com.github.mikephil.charting.listener.OnChartGestureListener;
import com.github.mikephil.charting.listener.OnChartValueSelectedListener;
import com.google.firebaseio.database.ChildEventListener;
import com.google.firebaseio.database.DataSnapshot;
import com.google.firebaseio.database.DatabaseError;
import com.google.firebaseio.database.DatabaseReference;
import com.google.firebaseio.database.FirebaseDatabase;
import java.util.ArrayList;

public class GraphicsTaskActivity extends AppCompatActivity implements
OnChartGestureListener, OnChartValueSelectedListener{

    // To manage firebase database
    FirebaseDatabase database;
    DatabaseReference myRef;

    // to store and graph data values
    ArrayList<Float> temps;
    ArrayList<Float> hums;
    LineChart mChart;
    Boolean temps_enabled;

    // To get input data from OnLineOptionsActivity
    Intent intent;
    Integer id_sensor;

    Button data;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_graphics);
        mChart = findViewById(R.id.linechart);
        mChart.setOnChartGestureListener(this);
        mChart.setOnChartValueSelectedListener(this);
        mChart.setDrawGridBackground(false);
        data = findViewById(R.id.showData);

        // data storage
        temps = new ArrayList<Float>();
        hums = new ArrayList<Float>();
        temps_enabled = false;
    }
}
```

```
// Get ID from OnlineOptionsActivity
intent = getIntent();
id_sensor = intent.getIntExtra("ID_S", 0);

// database instance to get/set data
database = FirebaseDatabase.getInstance();
myRef = database.getReference("Sensors").child("Sensor_" +
    id_sensor.toString());

myRef.child("Temperatures").addChildEventListener(new
    ChildEventListener() {
        @Override
        public void onChildAdded(DataSnapshot dataSnapshot, String s) {
            String temp = dataSnapshot.getValue(String.class);
            temps.add(Float.parseFloat(temp)/10);
        }
        @Override
        public void onChildChanged(DataSnapshot dataSnapshot, String s) {
        }
        @Override
        public void onChildRemoved(DataSnapshot dataSnapshot) {
        }
        @Override
        public void onChildMoved(DataSnapshot dataSnapshot, String s) {
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {
        }
    });
myRef.child("Humidities").addChildEventListener(new
    ChildEventListener() {
        @Override
        public void onChildAdded(DataSnapshot dataSnapshot, String s) {
            String hum = dataSnapshot.getValue(String.class);
            hums.add(Float.parseFloat(hum)/10);
        }
        @Override
        public void onChildChanged(DataSnapshot dataSnapshot, String s) {
        }
        @Override
        public void onChildRemoved(DataSnapshot dataSnapshot) {
        }
        @Override
        public void onChildMoved(DataSnapshot dataSnapshot, String s) {
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {
        }
    });
}
@Override
public void onChartGestureStart(MotionEvent me,
    ChartTouchListener.ChartGesture lastPerformedGesture) {
    Toast.makeText(GraphicsTaskActivity.this, "START, x: " + me.getX()
        + ", y: " + me.getY(), Toast.LENGTH_LONG).show();
}

@Override
public void onChartGestureEnd(MotionEvent me,
    ChartTouchListener.ChartGesture lastPerformedGesture) {
    Toast.makeText(GraphicsTaskActivity.this, "END, lastGesture: "
        + lastPerformedGesture, Toast.LENGTH_LONG).show();
// un-highlight values after the gesture is finished and no single-tap
```

```
    if(lastPerformedGesture != ChartTouchListener.ChartGesture.SINGLE_TAP)
        // or highlightTouch(null) for callback to onNothingSelected(...)
        mChart.highlightValues(null);
}

@Override
public void onChartLongPressed(MotionEvent me) {
    Toast.makeText(GraphicsTaskActivity.this, "Chart longpressed.",
        Toast.LENGTH_LONG).show();
}

@Override
public void onChartDoubleTapped(MotionEvent me) {
    Toast.makeText(GraphicsTaskActivity.this, "Chart double-tapped.",
        Toast.LENGTH_LONG).show();
}

@Override
public void onChartSingleTapped(MotionEvent me) {
    Toast.makeText(GraphicsTaskActivity.this, "Chart single-tapped.",
        Toast.LENGTH_LONG).show();
}

@Override
public void onChartFling(MotionEvent me1, MotionEvent me2,
    float velocityX, float velocityY) {
    Toast.makeText(GraphicsTaskActivity.this, "Chart flinged. VeloX: "
        + velocityX + ", VeloY: " + velocityY, Toast.LENGTH_LONG).show();
}

@Override
public void onChartScale(MotionEvent me, float scaleX, float scaleY) {
    Toast.makeText(GraphicsTaskActivity.this, "ScaleX: " + scaleX
        + ", ScaleY: " + scaleY, Toast.LENGTH_LONG).show();
}

@Override
public void onChartTranslate(MotionEvent me, float dX, float dY) {
    Toast.makeText(GraphicsTaskActivity.this, "dX: " + dX + ", dY: " + dY,
        Toast.LENGTH_LONG).show();
}

@Override
public void onValueSelected(Entry entry, int i, Highlight highlight) {
    Toast.makeText(GraphicsTaskActivity.this, entry.toString())
        Toast.LENGTH_LONG).show();
    Toast.makeText(GraphicsTaskActivity.this, "low: "
        + mChart.getLowestVisibleXIndex() + ", high: "
        + mChart.getHighestVisibleXIndex(),
        Toast.LENGTH_LONG).show();
    Toast.makeText(GraphicsTaskActivity.this, "xmin: "
        + mChart.getXChartMin() + ", xmax: " + mChart.getXChartMax()
        + ", ymin: " + mChart.getYChartMin() + ", ymax: "
        + mChart.getYChartMax(), Toast.LENGTH_LONG).show();
}

@Override
public void onNothingSelected() {
    Toast.makeText(GraphicsTaskActivity.this, "Nothing selected.",
        Toast.LENGTH_LONG).show();
}
```

```

// This is used to store x-axis values temps
private ArrayList<String> setXAxisValuesTemp(){
    ArrayList<String> xVals = new ArrayList<String>();
    for(Integer x = 1; x <= temps.size(); x++){
        xVals.add(x.toString());
    }
    return xVals;
}

// This is used to store Y-axis values temps
private ArrayList<Entry> setYAxisValuesTemp(){
    ArrayList<Entry> yVals = new ArrayList<Entry>();
    for(Integer index = 0; index <= (temps.size() - 1); index++){
        yVals.add(new Entry(temps.get(index), index));
    }
    return yVals;
}

// This is used to store x-axis values temps
private ArrayList<String> setXAxisValuesHum(){
    ArrayList<String> xVals = new ArrayList<String>();
    for(Integer x = 1; x <= hums.size(); x++){
        xVals.add(x.toString());
    }
    return xVals;
}

// This is used to store Y-axis values temps
private ArrayList<Entry> setYAxisValuesHum(){
    ArrayList<Entry> yVals = new ArrayList<Entry>();
    for(Integer index = 0; index <= (hums.size() - 1); index++){
        yVals.add(new Entry(hums.get(index), index));
    }
    return yVals;
}

private void setData(String data_in) {
    ArrayList<String> xVals;
    ArrayList<Entry> yVals;
    LineDataSet set1;

    if(data_in.equals("Temp")){
        xVals = setXAxisValuesTemp();
        yVals = setYAxisValuesTemp();
    }else if(data_in.equals("Hum")){
        xVals = setXAxisValuesHum();
        yVals = setYAxisValuesHum();
    }else{
        xVals = null;
        yVals = null;
    }
    // create a dataset and give it a type
    if(data_in.equals("Temp")){
        set1 = new LineDataSet(yVals, "Temperatures");
        set1.setFillColor(Color.RED);
    }else if(data_in.equals("Hum")){
        set1 = new LineDataSet(yVals, "Humidities");
        set1.setFillColor(Color.BLUE);
    }else{
        set1 = null;
    }
    set1.setFillAlpha(110);
    set1.setDrawCubic(true);
    set1.setColor(Color.BLACK);
}

```

```
    set1.setCircleColor(Color.BLACK);
    set1.setLineWidth(1f);
    set1.setCircleRadius(3f);
    set1.setDrawCircleHole(false);
    set1.setValueTextSize(9f);
    set1.setDrawFilled(true);

    ArrayList<ILineDataSet> dataSets = new ArrayList<ILineDataSet>();
    dataSets.add(set1); // add the datasets

    // create a data object with the datasets
    LineData data = new LineData(xVals, dataSets);

    // set data
    mChart.setData(data);

    // no description text
    mChart.setDescription(data_in + " Line Chart");
    mChart.setNoDataTextDescription("You need to provide data for the
        chart.");
}

public void showData(View view) {
    // add data
    if(tempEnabled == true){
        setData("Hum");
        tempEnabled = false;
    }else{
        setData("Temp");
        tempEnabled = true;
    }

    // get the Legend (only possible after setting data)
    Legend l = mChart.getLegend();

    // modify the legend ...
    // l.setPosition(LegendPosition.LEFT_OF_CHART);
    l.setForm(Legend.LegendForm.LINE);

    // enable touch gestures
    mChart.setTouchEnabled(true);

    // enable scaling and dragging
    mChart.setDragEnabled(true);
    mChart.setScaleEnabled(true);

    mChart.getAxisRight().setEnabled(false);

    // refresh the drawing
    mChart.invalidate();
}

public static Intent makeIntent(Context context) {
    return new Intent(context, GraphicsTaskActivity.class);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".GraphicsTaskActivity">

    <Button
        android:id="@+id/showData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="showData"
        android:text="Show Data"/>

    <com.github.mikephil.charting.charts.LineChart
        android:id="@+id/linechart"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

XII. Plan de trabajo

Planificación y propuesta de diseño	WP ref: 1
Componente principal: Estudio y análisis	Hoja 1 de 6
Descripción: Se pretende estudiar y analizar las posibilidades acerca de la realización del proyecto.	Start date: 4/09/2017 End date: 24/09/2017
	Entregable:
Tarea 1.1: Estudio de especificaciones y objetivos del proyecto. Tarea 1.2: Propuesta de diferentes alternativas de diseño. Tarea 1.3: Elección del diseño final y elección del material a utilizar.	Propuesta final del sistema y materiales a utilizar.

Documentación	WP ref: 2
Componente principal: Síntesis y redacción	Hoja 2 de 6
Descripción: Redacción de los diferentes documentos del proyecto.	Start date: 26/09/2017 End date: 11/05/2018
	Entregable:
Tarea 2.1: Propuesta y Plan de Trabajo. Tarea 2.2: Critical Design. Tarea 2.3: Final Report. Tarea 2.4: Diapositivas y exposición oral.	Documentos del proyecto.



Sensor auxiliar	WP ref: 3
Componente principal: Integración hardware y diseño software	Hoja 3 de 6
Descripción: Mediante una placa Arduino NANO y un sensor AM2302 se pretende simular una salida digital similar a la del sensor de ultrasonidos.	Start date: 9/10/2017 End date: 7/01/17
	Entregable:
Tarea 3.1: Puesta en marcha del hardware y del entorno software. Tarea 3.2: Obtener datos del sensor por puerto digital de entrada. Tarea 3.3: Visualizar los datos recibidos del sensor en un LCD. Tarea 3.4: Comunicación UART para transferir los datos obtenidos del sensor.	Subsistema capaz de comunicar los datos recibidos del sensor por UART.

Red Wi-Fi para transmission de datos inalámbrica	WP ref: 4
Componente principal: Diseño software	Hoja 4 de 6
Descripción: Mediante una Raspberry Pi Zero W se establecerá conexión a una red Wi-Fi generada por un dispositivo inalámbrico para así poder hacer peticiones de descarga de datos.	Start date: 23/10/2017 End date: 11/02/2018
	Entregable:
Tarea 4.1: Puesta en marcha del hardware y del entorno software. Tarea 4.2: Recepción de datos por UART. Tarea 4.3: Configuración de red Wi-Fi para transferir datos en caso de petición.	Red Wi-Fi operativa para almacenar y transmitir información.



Programa/APP de descarga, almacenamiento y visualización de datos	WP ref: 5
Componente principal: Diseño software	Hoja 5 de 6
Descripción: Android APP que permitirá la petición de descarga de datos, la visualización de los mismos y gestionar el sistema.	Start date: 26/10/2017 End date: 16/04/2018
	Entregable:
Tarea 5.1: Puesta en marcha del entorno software. Tarea 5.2: Comunicación Wi-Fi con el sensor. Tarea 5.3: Recepción de datos por Wi-Fi. Tarea 5.4: Subida de datos a la base de datos Tarea 5.5: Visualización de los datos alojados en la base de datos.	APP intuitiva de gestión, descarga de datos y visualización de información.

Test del funcionamiento del sistema	WP ref: 6
Componente principal: Test software y hardware	Hoja 6 de 6
Descripción: Prueba del correcto funcionamiento del sistema diseñado.	Start date: 23/10/2017 End date: 16/04/2018
	Entregable:
Tarea 6.1: Test de cada uno de los subsistemas (WP3, WP4 y WP5) Tarea 6.2: Test general del sistema.	Subsistemas y sistema global en correcto funcionamiento.

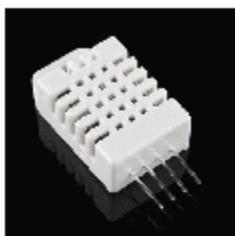
XIII. Diagrama de Gantt



#WP	Tasks	
1	Planteamiento y propuesta de diseño	12-18 FEB
1.1	Estudio de especificaciones y objetivos del proyecto	19-25 FEB
1.2	Propuesta de diferentes alternativas de diseño	20-4 MAR
1.3	Elección del diseño final y selección del material	5-11 MAR
2	Documentación	12-18 MAR
2.1	Redactar Propuesta de Proyecto	20-1 APR
2.1.1	Entrega de Propuesta de proyecto	2-8 APR
2.2	Redactar Critical Review	9-15 APR
2.2.1	Entrega Critical Review	16-22 APR
2.3	Redactar Final Report	23-29 APR
2.3.1	Entrega Final Report	30-6 MAY
2.4	Dispositivos y exposición oral	7-13 MAY
3	Sensor de presión	25-27 MAY
3.1	Puesta en marcha del hardware y del entorno software	1-7 JUN
3.2	Obtención de datos digitales del sensor	8-14 JUN
3.3	Visualización en LCD mediante SPI	15-21 JUN
3.4	Habilitar comunicación UART para transferir datos + protocolo de comunicaciones	22-28 JUN
4	Red WiFi para transmisión de datos	29-30 JUN
4.1	Puesta en marcha del hardware y del entorno software	1-7 JUL
4.2	Habilitar comunicación UART para recepción de datos + protocolo de comunicaciones	8-14 JUL
4.3	Configuración de red WiFi para solicitar peticiones	15-21 JUL
5	Apilado de datos, almacenamiento y visualización de datos	22-28 JUL
5.1	Puesta en marcha del entorno software	29-30 JUL
5.2	Recepción inalámbrica de datos	1-7 AÑO
5.3	Subida de datos a servidor remoto	8-14 AÑO
5.4	Visualización de datos del servidor	15-21 AÑO
6	Todos del funcionamiento del sistema	22-28 AÑO
6.1	Test de cada uno de los subsistemas	29-30 AÑO
6.2	Test general del sistema	1-7 AÑO

XIV. Hardware utilizado - Datasheets

Your specialist in innovating humidity & temperature sensors



Standard AM2302/DHT22



AM2302/DHT22 with big case and wires

Digital relative humidity & temperature sensor AM2302/DHT22

1. Feature & Application:

- *High precision
- *Capacitive type
- *Full range temperature compensated
- *Relative humidity and temperature measurement
- *Calibrated digital signal
- *Outstanding long-term stability
- *Extra components not needed
- *Long transmission distance, up to 100 meters
- *Low power consumption
- *4 pins packaged and fully interchangeable

2. Description:

AM2302 output calibrated digital signal. It applies exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability. Its sensing elements is connected with 8-bit single-chip computer.

Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of programme in OTP memory, when the sensor is detecting, it will cite coefficient from memory.

Small size & low consumption & long transmission distance(100m) enable AM2302 to be suited in all kinds of harsh application occasions. Single-row packaged with four pins, making the connection very convenient.

3. Technical Specification:

Model	AM2302	
Power supply	3.3-5.5V DC	
Output signal	digital signal via 1-wire bus	
Sensing element	Polymer humidity capacitor	
Operating range	humidity 0-100%RH;	temperature -40~80Celsius
Accuracy	humidity +2%RH(Max +5%RH);	temperature +0.5Celsius
Resolution or sensitivity	humidity 0.1%RH;	temperature 0.1Celsius
Repeatability	humidity +/-1%RH;	temperature +/-0.2Celsius
Humidity hysteresis	+0.3%RH	
Long-term Stability	+0.5%RH/year	
Interchangeability	fully interchangeable	