DESARROLLO Y DISEÑO DE UN DATA LOGGER INALÁMBRICO USANDO LA PLATAFORMA ANDROID PARA ADQUISICIÓN Y VISUALIZACIÓN DE DATOS

LUIS SEBASTIÁN FONSECA ACOSTA

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍAS
BOGOTÁ, D.C
2016

DESARROLLO Y DISEÑO DE UN DATA LOGGER INALÁMBRICO USANDO LA PLATAFORMA ANDROID PARA LA INTERPRETACIÓN DE DATOS

LUIS SEBASTIÁN FONSECA ACOSTA

Trabajo de grado para optar el título de Ingeniero Electrónico

Director de Tesis, Ing. Andrés Camilo Jiménez

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍAS
BOGOTÁ, D.C
2016

Nota de aceptación

CONTENIDO

Pág.

GLOSARIO 7
RESUMEN 9
INTRODUCCIÓN 10
OBJETIVO 11
MARCO TEÓRICO 12
Registradores de datos o data logger 12
Integridad de datos 13
Conectividad Bluetooth 14
ANTECEDENTES 16
PROBLEMA DE INVESTIGACIÓN 19
ALCANCE 19
LIMITACIONES 19
DISEÑO DE LA APLICACIÓN 23
1 CONEXIÓN BLUETOOTH Y DISEÑO DE INTERFAZ GRÁFICA 23
1.1 DISEÑO DE LA INTERFAZ GRÁFICA 23
1.2 ACTIVAR Y DESACTIVAR LA CONEXIÓN BLUETOOTH 25
1.3 BÚSQUEDA DE DISPOSITIVOS CERCANOS 26
1.4 MOSTRAR DISPOSITIVOS VINCULADOS 28
1.5 VERIFICAR ESTADO ACTUAL 29
1.6 CONECTAR A UN DISPOSITIVO 30
1.7 DESCONECTAR DE UN DISPOSITIVO 33
1.7.1 Desconexión voluntaria 33
1.7.2 Desconexión inesperada 34
2 RECEPCIÓN, LECTURA Y ENVIO DE DATOS 35
2.1 RECEPCIÓN DE DATOS 35
2.2 INTEGRIDAD DE LOS DATOS 36
2.2.1 Proceso de validación 37
2.2.2 Proceso de normalización 38
2.2.3 Proceso de identificación 38

2.3 ENVÍO DE DATOS 40
3 VISUALIZACIÓN DE DATOS 40
3.1 CREACIÓN DEL OBJETO DATO 41
3.2 VISUALIZACIÓN EXPLICITA DEL DATO 41
3.3 VISUALIZACIÓN GRÁFICA DEL DATO 42
3.3.1 Restablecimiento de la conexión 43
3.3.2 Configuración de la gráfica. 43
4 ALMACENAMIENTO DE DATOS 45
4.1 UBICANDO LA RUTA DE LA MEMORIA SD 45
4.2 CREACIÓN Y ESCRITURA DEL ARCHIVO DE TEXTO EN LA MEMORIA SD 46
4.3 LECTURA DE ARCHIVOS DE TEXTO CREADOS CON ANTERIORIDAD 47
5 PRUEBAS REALIZADAS Y RESULTADOS OBTENIDOS 50
5.1 INTERFAZ USADA 50
5.2 PRUEBAS REALIZADAS 52
Lectura de datos de radiación UV 52
Lectura de datos de temperatura 54
CONCLUSIONES 58
TRABAJOS FUTUROS 59
BIBLIOGRAFÍA 60
ANEXOS 61

LISTA DE DIAGRAMAS

Pág.

Diagrama 1. Unidades básicas de la aplicación 21
Diagrama 2. Funcionamiento básico del registrador de datos o data logger con un sistema de adquisición de datos 22
Diagrama 3. Funcionamiento básico del protocolo de ventana deslizante 37
Diagrama 4. Algoritmo de normalización para integridad de datos 39
Diagrama 5.Diagrama de clase de la aplicación 49

LISTA DE FIGURAS

Pág.

Figura 1. forma básica de un registrador de datos 12
Figura 2. Data logger DT82 16
Figura 3. Data logger tinyLOG 17
Figura 4. Data logger EL-WIFI-TH 18
Figura 5. Data logger MSR145WD 18
Figura 6. Primera interfaz visualizada por el usuario al iniciar la aplicación 23
Figura 7. Cuadro de dialogo ABRIR 24
Figura 8. Cuadro de dialogo GRAFICA o Ver Grafica en la opción ABRIR 24
Figura 9. Interfaz mostrada función de graficado 25
Figura 10. Cuadro de dialogo función Activar/Desactivar Bluetooth 26
Figura 11. Estado de búsqueda de Dispositivos 27
Figura 12. Lista de dispositivos encontrados 28
Figura 13. Aviso de Búsqueda no exitosa 28
Figura 14. Lista de dispositivos vinculados 29
Figura 15. Aviso Verificación de estado actual 30
Figura 16. Consulta al usuario para conexión 31
Figura 17. Conexión exitosa 32
Figura 18. Conexión fallida 33
Figura 19. Aviso de desconexión voluntaria del dispositivo 34
Figura 20. Aviso de desconexión inesperada del dispositivo 35
Figura 21. Visualización del objeto 40
Figura 22. Ejemplo de visualización explícita 42
Figura 23. Aviso de desconexión inesperada 43
Figura 24. Ejemplo de visualización gráfica 44
Figura 25. Lista de posibles sistemas de medida 45
Figura 26. Ejemplo de archivo creado 47
Figura 27. Cuadro de diálogo para Activar conexión Bluetooth 50

Figura 28. Lista de dispositivos disponibles 50
Figura 29. Cuadro de diálogo para conectar con otro dispositivo 51
Figura 30. Visualización de datos en tiempo real 51
Figura 31. Sistema de adquisición y envío de datos usado 52
Figura 32. Estado inicial y final del dispositivo móvil 53
Figura 33. Gráfica de los últimos 1000 datos registrados visualizados en la aplicación 53
Figura 34. Gráfica de los últimos 1000 datos registrados visualizados en Excel 54
Figura 35. Gráfica de todos los datos obtenidos 54
Figura 36. Sistema usado para la medición de la temperatura del agua 55
Figura 37. Inicio de la toma de datos de temperatura 55
Figura 38. Gráfica de los últimos 1000 datos de temperatura en la aplicación 56
Figura 39. Gráfica de los últimos 1000 datos de temperatura en Excel 56
Figura 40. Gráfica de todos los datos obtenidos 57

LISTA DE TABLAS

Pág.

Tabla 1. Comparación de la aplicación propuesta con algunos datalogger ya existentes 20
Tabla 2. Atributos otorgados a cada dato que se recibe 38
Tabla 3. Caracterización de los posibles tipos datos a registrar 45

GLOSARIO

Actividad: Son el conjunto de acciones en las que el usuario interviene directamente con la aplicación.

ArrayAdapter: Sistema que permite almacenar determinada cantidad de elementos en forma de lista para usarlos posteriormente en un listView.

ArrayList: Es una clase que permite almacenar datos sin la necesidad de especificar su tamaño.

Buffers: Es un espacio en la memoria reservado para almacenar determinado tipo de dato.

Ciclos de vida: Es el estado en el que se encuentra una aplicación, como puede ser visible, en pausa, detenida o destruida

Clases: La clase define las propiedades y métodos que contiene un objeto.

Hilos: Es un sistema que permite realizar varias tareas de manera simultánea.

Intent: Un intent define una acción se va a llevar a cabo, como lo puede ser iniciar una nueva actividad o un servicio.

Layout: Es aquella estructura gráfica que el usuario ve en la pantalla.

Librería: Es un conjunto de funciones, objetos y métodos permiten desarrollar determinadas tareas.

ListView: Sistema que usando un ArrayAdapter, permite visualizar los datos en forma de lista.

Método: Son todas aquellas acciones u operaciones que se pueden realizar con un objeto.

Objeto: Es un espacio en la memoria reservado para almacenar una entidad, esta puede contener diferentes propiedades y operaciones.

OnClick: Así se denomina la acción de tocar algún elemento mostrado en pantalla.

Sistema Operativo: Es un sistema informático que realiza la administración de los recursos del hardware de algún dispositivo.

Socket: Es un objeto que contiene los métodos y variables para la administración de la comunicación entre dispositivos.

Thread: Es un objeto que se encarga de realizar aquellas tareas que se llevan a cabo de manera simultánea.

RESUMEN

Este proyecto desarrolla una aplicación para dispositivos móviles el sistema operativo *Android,* usando como sistema de comunicación la tecnología *Bluetooth.* Esta aplicación es capaz de recibir los datos enviados por algún sistema llamado servidor, el cual cumple la tarea de capturar los datos de la variable a medir y servir como fuente de datos.

En esta aplicación el usuario puede tener control sobre las tareas conexión establecida para activarla o desactivarla, buscar dispositivos cercanos, obtener un historial de los dispositivos enlazados, verificar el estado actual, incluso de ser necesario es posible enviar datos o comandos al dispositivo servidor al que se encuentra enlazado.

La función que caracteriza a esta aplicación es la visualización de los datos obtenidos por el sistema de medición (Servidor) en tiempo real, puede verse de manera explícita en la pantalla como también existe la posibilidad de hacerlo a través de una gráfica, permitiendo al usuario realizar bajo su criterio una interpretación inmediata de los datos.

Otra función a destacar en esta aplicación es la de almacenar datos en un archivo de texto dentro de una memoria extraíble al finalizar la conexión, como lo puede ser una tarjeta SD, con la intención de obtener un registro de los datos obtenidos a lo largo del proceso.

Para el desarrollo de este proyecto se usó como lenguaje de programación *Java* y como entorno de desarrollo *Android Studio*. En la etapa de pruebas se usó como fuente de datos un sistema de telemetría para captura de radiación Ultra Violeta con el fin de obtener el comportamiento de esta variable durante determinados momentos del día.

- Captura de datos
- Procesamiento de datos
- Visualización de datos
- Almacenamiento de datos

INTRODUCCIÓN

En la actualidad los procesos de investigación y desarrollo se dan con más frecuencia en la población, esto debido al nacimiento de nuevas tecnologías, métodos de investigación e incluso fuentes que permiten fácil acceso a la información. Una etapa de vital importancia en estos procesos es la obtención y procesamiento de datos, debido a esto las herramientas que intervienen en esta etapa deben ser cada vez menos costosas, de más fácil acceso y más eficientes, para así serle útil a la mayor cantidad grupos de investigación y desarrollo.

Como solución a esta necesidad aparecieron los primeros *Registradores de datos* o también llamados *Data Logger*, estos sistemas se encargan de registrar los datos a través del tiempo y así mismo algunos son capaces de realizar tareas de análisis, algunos también se diseñan para propósitos generales, por lo tanto se pueden programar para diferentes tipos de medición, en este punto ya se puede hablar de registradores electrónicos.

Uno de los objetivos que busca cumplir este tipo de sistema, es la capacidad de capturar los datos la mayor cantidad de tiempo posible, sin verse afectado por aspectos relacionados a: fuentes de alimentación, factores climáticos o la supervisión constante de alguna persona, es decir que el registrador deber ser lo más autónomo posible.

Con la intención de cubrir algunas de las necesidades básicas anteriormente mencionadas se realizó este proyecto.

OBJETIVO

Desarrollar una aplicación para dispositivos móviles que cumpla las funciones básicas de un Registrador de datos o *Data Logger* usando el sistema *Bluetooth* como forma de comunicación.

- Desarrollar una interfaz gráfica para la visualización de datos.
- Implementar funciones dentro de la aplicación que permitan al usuario tener control básico sobre la conexión Bluetooth
- Validar el sistema capturando diferentes tipos de datos, para luego generar un archivo de texto.
- Implementar procesos y protocolos de comunicación que permitan mantener la integridad los datos.
- Desarrollar un sistema que permita el almacenamiento de datos

MARCO TEÓRICO

Registradores de datos o data logger

Las tareas más básicas de un registrador de datos son medir y almacenar parámetros físicos o eléctricos dentro de un intervalo de tiempo. Existen dispositivos capaces de medir y registrar datos, como algunos que no incluyen sistema de medición dentro del mismo dispositivo, también algunos cuentan con sistemas de visualización y análisis en pantallas.

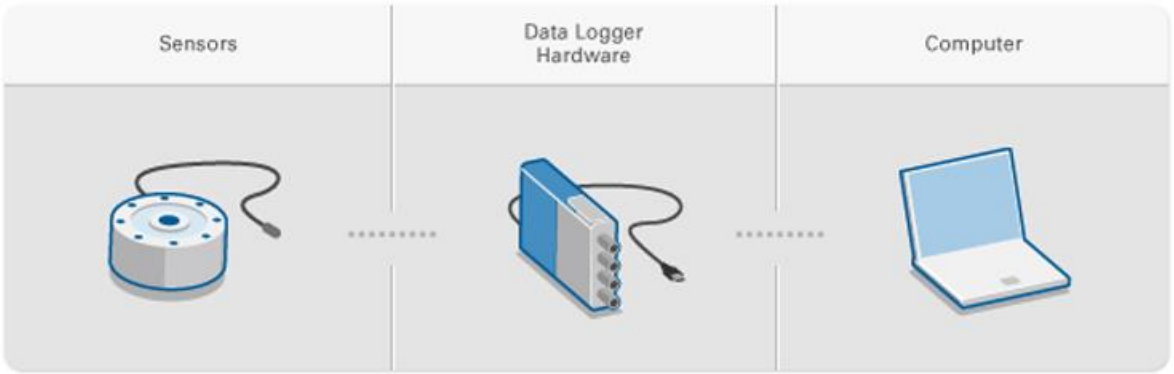
Algunos tipos de medida que comúnmente se registran en estos sistemas son:

- Temperatura
- Tensión eléctrica
- Corriente
- Presión
- Potencia
- Humedad
- Luz

Los registradores de datos pueden clasificarse en 2 tipos, autónomos o basados en PC.

Los de tipo autónomo son aquellos que tienen sistemas de almacenamiento interno y que por tanto no dependen de otros dispositivos para cumplir sus funciones de medición, conversión de señales y almacenamiento de datos.
Los de tipo basado en PC son sistemas que usan la capacidad de procesamiento y almacenamiento de un pc, este tipo puede ofrecer algunas ventajas como, visualización en tiempo real, análisis en línea, alta capacidad de almacenamiento de datos conectividad a la red. (NationalInstruments)

Figura 1. Forma básica de un registrador de datos



Fuente. Disponible en http://www.ni.com/data_logger/esa/whatis.htm imagen 8 noviembre de 2016

Integridad de datos

La integridad de datos se refiere a los valores reales que se almacenan y se utilizan en las estructuras de datos de la aplicación. La aplicación debe ejercer un control deliberado sobre todos los procesos que utilicen los datos para garantizar la corrección permanente de la información.

Es posible garantizar la integridad de los datos mediante la implementación escrupulosa de varios conceptos clave, como los que se incluyen a continuación
:

- Normalizar datos.
- Validar los datos.
- Diseño de datos

Normalización de datos:

La tarea de un diseñador de bases de datos consiste en estructurar los datos de forma que se eliminen duplicaciones innecesarias y se proporcione una ruta de búsqueda rápida para toda la información necesaria. El proceso de perfeccionar tablas, claves, columnas y relaciones para crear una base de datos eficaz se denomina normalización. La normalización no sólo es aplicable a archivos relacionales; también es una actividad de diseño común para archivos indizados.

Validación de datos:

La validación de datos garantiza la corrección y precisión de todos los valores de datos de la aplicación. La validación de datos de la aplicación puede diseñarse utilizando distintos enfoques: código de interfaz de usuario, código de aplicación o restricciones de bases de datos.

Diseño de datos:

El diseño de datos consiste en descubrir y la definir completamente de los procesos y características de los datos de la aplicación. El diseño de datos es un proceso de perfeccionamiento gradual que abarca desde la cuestión más elemental, "¿Qué datos requiere la aplicación?", hasta los procesos y estructuras de datos precisos que proporcionan dichos datos. Si el diseño de datos es bueno, el acceso a los datos de la aplicación será rápido y fácil de mantener, y podrá aceptar sin problemas las futuras mejoras de los datos.
El proceso de diseño de datos incluye la identificación de los mismos, la definición de tipos de datos y mecanismos de almacenamiento concretos, y la tarea de garantizar la integridad de los datos mediante el uso de reglas de empresa y otros mecanismos de exigencia en tiempo de ejecución. (MicrosofDeveloperNetwork).

Conectividad Bluetooth

Desarrollada en 1994 esta tecnología se propuso como alternativa inalámbrica a la clásica conexión con cables para el intercambio de datos, El nombre Bluetooth tiene como origen el nombre de un rey danés Harold Bluetooth el cual logró unir varias facciones de los actuales países de Noruega, Suecia y Dinamarca. De este mismo modo la tecnología Bluetooth tiene como propósito crear un estándar de comunicación abierto para permitir la interacción entre diversos tipos de sistemas y dispositivos.

Una de las aplicaciones más populares de esta tecnología ha sido la transmisión de audio, lo que permite crear sistemas manos libres en dispositivos móviles, automóviles y reproductores de música.

Para este tipo de aplicaciones se utiliza una versión Bluetooth llamada BR/EDR (tasa de bits/velocidad de datos mejorada), esta versión fue diseñada para enviar un flujo constante de datos de alta calidad, como lo puede ser archivos de audio, todo esto buscando mantener una eficiencia en el aspecto de energía.

Con el desarrollo de esta versión de bajo consumo de energía, los de desarrolladores de tecnología son capaces de usar pequeños sensores que requieren de baterías de tipo moneda, la cual se puede usar durante meses, e incluso años.

Otra consecuencia de esta versión de Bluetooth permite que se usen fuentes alternas de energía, como la solar o la cinética, y así tener dispositivos que operen un tiempo aún mayor.

Existe otra versión de la tecnología Bluetooth llamada GATT, la cual es extremadamente flexible desde la perspectiva de un desarrollador, lo cual quiere decir que puede usarse en prácticamente cualquier dispositivo móvil o PC. La unión de esta versión y la BR/EDR dan como resultado las actuales versiones de esta tecnología.

¿Cómo funciona la tecnología Bluetooth?

Un dispositivo con esta tecnología utiliza ondas de radio en lugar de cables para conectarse a otro dispositivo. Un dispositivo como un auricular o un reloj, contiene un pequeño chip con un sistema de procesamiento y un software, cuando dos sistemas Bluetooth quieren comunicarse, es necesario emparejarlos.

La comunicación entre estos dispositivos se hace mediante redes ad hoc de corto alcance, también conocidas como piconet. Una red de dispositivos Bluetooth puede variar de 2 a 8 dispositivos, cuando se establece la conexión un dispositivo toma la función de maestro, mientras los otros son esclavos, estos piconets

establecen la red de forma dinámica y automática en función de la proximidad de los dispositivos emparejados.

Ventajas de la tecnología Bluetooth

Posiblemente la mayor ventaja de esta tecnología, es que se encuentra en gran parte de los dispositivos que existen actualmente, algunas otras ventajas con:

- Bajo consumo de energía
- Fácil manejo
- Bajo costo
- Tecnología abierta para desarrolladores

Normas Bluetooth

- IEEE 802.15.1 define Bluetooth 1.x, que puede alcanzar velocidades de 1 Mbps;
- IEEE 802.15.2 recomienda prácticas para utilizar la banda de frecuencia de 2.4 GHz (la frecuencia también utilizada por WiFi). Sin embargo, este estándar todavía no se ha aprobado;
- IEEE 802.15.3 es un estándar que actualmente se está desarrollando, que ofrecerá velocidad de banda ancha (20 Mbps) con Bluetooth;
- IEEE 802.15.4 es un estándar que actualmente se está desarrollando para el uso con aplicaciones Bluetooth de baja velocidad.

Versiones de la tecnología Bluetooth

Bluetooth 1.0 - 1.2:

La versión 1.0 contiene errores de compatibilidad de hardware con cierto tipo de dispositivos, lo que conlleva a que algunos accesorios no funcionen de manera correcta. La versión 1.1 corrige muchos de estos errores, mediante el uso de un identificador de hardware personalizado. La versión 1.2 contiene además de lo anterior mencionado un aumento en la velocidad de conexión, mejores tasa de transmisión de datos.

Bluetooth 2.0 – 2.1:

En la versión 2.0 se incorporó una función que permite aumentar la velocidad con dispositivos EDR. En la versión 2.1 se agregó la característica Secure Simple Pairing, lo que mejoró aún más la velocidad de conexión y la seguridad en la transmisión de datos, además de adquirir y mostrar datos adicionales del dispositivo al que se está conectando.

Bluetooth 3.0:

La velocidad de transmisión que permite esta versión es de máximo 24 Mbps, usando sistemas de red inalámbrica 802.11, también se incluyó una tecnología conocida como MAC/PHY alternativa, que permite a los dispositivos Bluetooth cambiar el modo de datos.

Bluetooth 4.0:

En esta versión se implementaron los protocolos antiguos y el nuevo SA, con el fin de garantizar un alto nivel de compatibilidad, al igual que un protocolo que reduce el consumo de energía, pero con la condición que el dispositivo enlazado también cuente con la versión 4.0.

ANTECEDENTES

Basándose en algunos artículos consultados en la base de datos de la *IEEE* se evidencia que campos como la medicina, el estudio del clima y labores industriales, este tipo de dispositivos toman gran importancia para la recopilación y análisis de datos. (NationalInstruments)

Algunos proyectos como *"High sampling rate datalogger for the characterization of acceleration signals on the human body running"*, *y "GSM wireless datalogger of small hydro power generation system* (2)*,* buscan recolectar ya analizar datos en sus respectivos objetos de estudio, en el primer ejemplo se diseña un registrador básico, en el que cumple la única función de almacenar datos, mientras que en el segundo se usa un registrador DT82 el cual es un sistema que ofrece la empresa dataTaker, pero no ofrece ventajas en aspectos como tamaño, visualización de datos y consumo de energía, en ambos casos se puede ofrecer una alternativa, como una aplicación móvil que permita realizar estas tareas de almacenamiento y visualización, ofreciendo como ventaja la accesibilidad mediante cualquier dispositivo móvil.

Figura 2. Data logger DT82

Fuente. Disponible en http://www.datataker.com/DT82E.php imagen 28 de Julio 2016

Data acquisition and cloud storage system applied photovoltaic systems" es un proyecto que usa una aplicación móvil para almacenar los datos obtenidos en un sistema fotovoltaico en la nube, tomando como ejemplo este proyecto se puede ver que el uso de este tipo de registradores es totalmente viable para aplicaciones prácticas.

La necesidad de tener un sistema de este tipo que sea portable se puede ver en el proyecto *"Portable datalogger for intracranial pressure monitoring and intelligent diagnosis"* (4), un proyecto que propone un registrador de datos portable, el cual almacena los datos para luego poder visualizarlo en algún computador, aunque este registrador ofrece portabilidad, carece de algún sistema para la visualización de datos.

- Ejemplos de Data loggers inalámbricos usados en la actualidad

A continuación se mostrarán las ventajas y desventajas de algunos dispositivos usados en la actualidad, con el fin de evidenciar como se satisfacen o no algunas de las necesidades anterior mencionadas.

- tinyLOG

El tamaño de este dispositivo es posiblemente su mayor ventaja ya que lo hace útil en prácticamente cualquier espacio, incluye una batería de larga duración y es capaz de almacenar más de 130.000 datos. Al no contar un sistema de visualización propio, lo hace depender de dispositivos externos, por lo cual tampoco se puede visualizar datos en tiempo real, lo que representa una desventaja.

Figura 3. Data logger tinyLOG

Fuente. Disponible en http://www.coalesenses.com/index.php/products/solutions/environmental-monitoring/wireless-loggers/ imagen 28 de Julio 2016

- EL-WIFI-TH

Las principales ventajas de este dispositivo son el tamaño, duración de la batería, sistema de alarma para valores máximos o mínimos y la visualización del dato actual, aunque este dispositivo puede considerarse completo en cuanto a

rendimiento y sistemas de visualización carece de un sistema de almacenamiento propio ya que necesita de algún dispositivo externo que cuente con conexión WI-FI, para almacenar los datos obtenidos y obtener una gráfica de los mismos.

Figura 4. Data logger EL-WIFI-TH

Fuente. Disponible en http://www.lascarelectronics.com/temperaturedatalogger.php?datalogger=448#.imagen 28 de Julio 2016

- MSR145WD

Este dispositivo es posiblemente uno de los más completos que existe actualmente ya que cuenta con su propio sistema de almacenamiento y visualización diseñado para más de 1.000.00 de datos, batería de larga duración y capacidad para 5 tipos de medición, además de tener un sistema de monitoreo del dispositivo mediante una aplicación móvil con conexión *Bluetooth*, que también permite almacenar los datos obtenidos en un sistema en la nube.

Figura 5. Data logger MSR145WD

Fuente. Disponible en http://www.msr.ch/en/product/msr145wd.php .imagen 28 de Julio 2016

PROBLEMA DE INVESTIGACIÓN

Como se ha visto en algunos de los proyectos anteriormente citados, los registradores de datos han suplido varias necesidades como tamaño, capacidad de almacenamiento, visualización de datos, costo, tiempo de operación y accesibilidad.

¿Cómo se puede desarrollar un registrador de datos que cubra la mayor cantidad de necesidades?

Esta pregunta es la que se buscar responder con el desarrollo de este proyecto.

ALCANCE

En este caso se desarrolló la aplicación para una sistema de telemetría para captura y procesamiento de radiación Ultra Violeta, el cual tiene algunas características específicas, una de estas características es que cada dato que se envía debe terminar con un carácter definido de manera previa, el cual le indica a la aplicación cuando se ha terminado de enviar ese dato.

Otro aspecto a tener en cuenta es la magnitud física que se está midiendo, por lo cual se ajustaron los valores máximo y mínimo, unidades de medida y escalas usadas para la visualización de los datos en función de la radiación UV y sus unidades de medida, además que es posible realizar modificaciones a la aplicación para la lectura de datos provenientes de diferentes tipos de sensores.

LIMITACIONES

Factores como el consumo de energía, tiempo muestreo dependerán del dispositivo en el que se instale la aplicación. Para el caso factores como, capacidad de almacenamiento de los datos, y lugar de almacenamiento de datos se debe contar un sistema operativo inferior a la versión 4.4 (Kit Kat) para que sea posible almacenar los datos en la tarjeta SD, ya que en esta versión y las superiores la aplicaciones ya no tienen el permiso de escribir datos en memorias externas.

Este sistema es capaz de conectarse únicamente con un dispositivo, aunque es capaz de visualizar en la gráfica un máximo de dos tipos de datos.

También si se desea usar por largos periodos de tiempo se debe contar con un sistema de carga, ya que la constante visualización de los dotas afecta en gran forma el rendimiento de la batería en los dispositivos móviles.

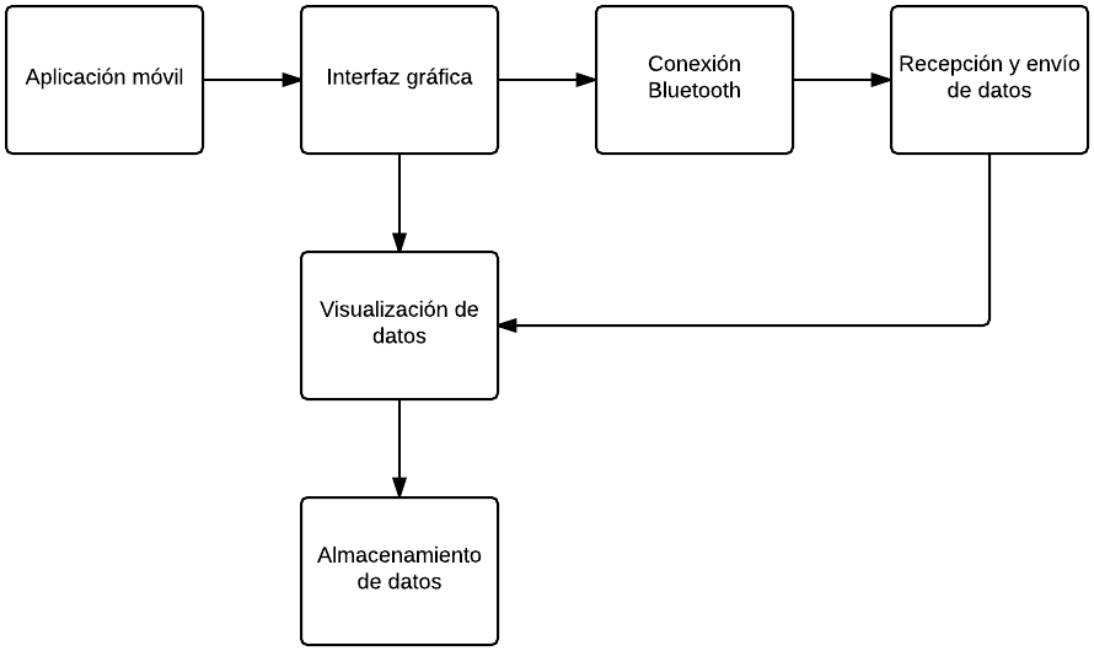
Tabla 1. Comparación de la aplicación propuesta con algunos datalogger ya existentes

Característica	Aplicación propuesta	dataTaker DT80	testo Saveris
Puerto Serial	NO	SI	SI
Puerto Ethernet	NO	SI	SI
Puerto USB	NO	SI	SI
Conectividad Wi-Fi	NO	NO	SI
Conectividad Bluetooth	SI	NO	NO
Requiere Software adicional	NO	SI	SI
Requiere Hardware adicional	SI	NO	NO
Formato de datos	Fecha/Hora/Dato/Unidad de medida		
Capacidad de almacenamiento memoria	Está en función de la capacidad de memoria externa del dispositivo	128MB	Capacidad del servicio en la nube
Frecuencia de muestreo	Máximo 5Hz	100Hz-10KHz	0.115mHz-0.2Hz
Batería	SI	SI	SI
Requiere sistema de alimentación externo	NO	NO	NO
Sistema de Alarma	NO	SI	SI
Magnitudes medibles	Temperatura, Radiación UV, Luz Visible	Temperatura, Humedad	Temperatura, Humedad
Rango de medición	Asignada por el usuario		Asignada por el usuario
Sistema de almacenamiento de datos	SI	SI	SI
Ajuste para valores máximo y mínimo	SI	NO	SI
Capacidad de transferencia datos a otro dispositivo	SI	SI	SI
Visualización gráfica	SI	SI	SI

Fuente. Fuente propia

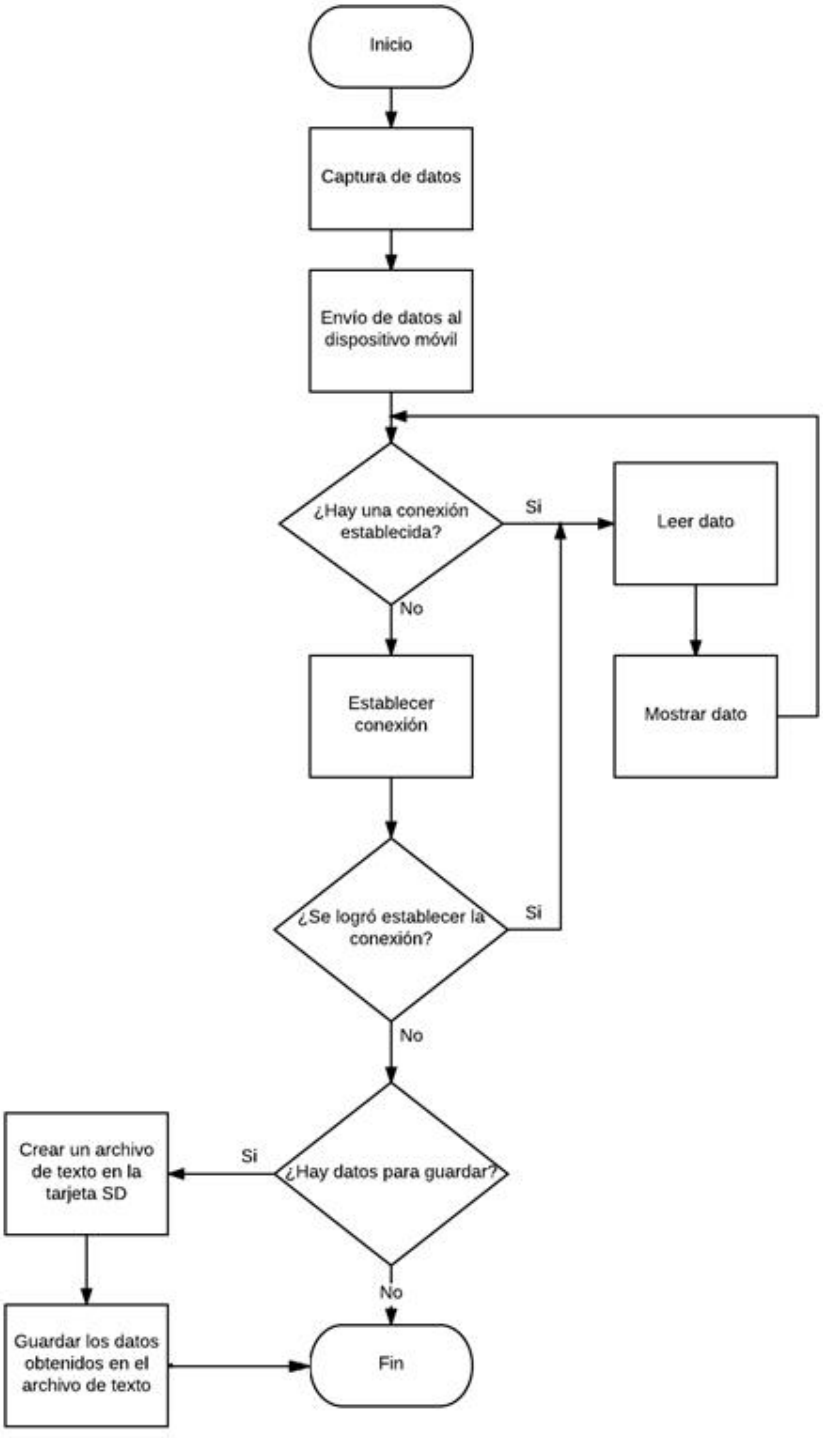
DISEÑO DE LA APLICACIÓN

Diagrama 1. Unidades básicas de la aplicación



Fuente. Fuente propia

Diagrama 2. Funcionamiento básico del registrador de datos o data logger con un sistema de adquisición de datos



Fuente. Fuente propia

DISEÑO DE LA APLICACIÓN

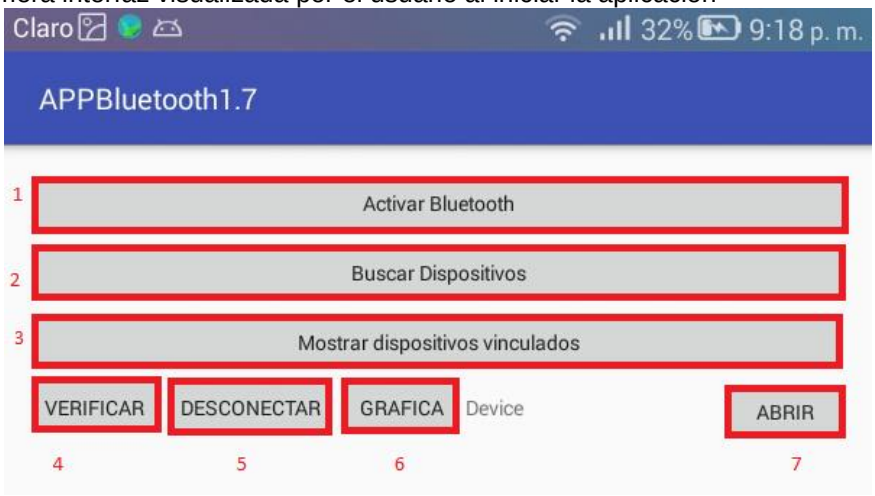
1 CONEXIÓN BLUETOOTH Y DISEÑO DE INTERFAZ GRÁFICA

La base del desarrollo de esta aplicación es la comunicación entre el dispositivo móvil y el sistema de captura y envío de datos, por lo cual se diseñó la interfaz gráfica mostrara en la sección 1.1.

1.1 DISEÑO DE LA INTERFAZ GRÁFICA

A continuación se muestra la interfaz que usó para las pruebas realizadas, y la función de cada una de las opciones mostradas.

Figura 6. Primera interfaz visualizada por el usuario al iniciar la aplicación



Fuente. Fuente propia

1. Activar Bluetooth: Permite al usuario activar o desactivar la conexión Bluetooth desde la aplicación, con el fin de evitar navegar hasta la sección de ajustes.

2. Buscar Dispositivos: Realiza una búsqueda de los dispositivos cercanos que estén disponibles para enlazar.

3. Mostrar dispositivos vinculados: Permite realizar una conexión con los dispositivos vinculados anteriormente, con el fin de evitar realizar una nueva búsqueda.

4. VERIFICAR: Le informa al usuario si el dispositivo se encuentra enlazado a otro, de ser así también informa el nombre del dispositivo vinculado.

5. DESCONECTAR: Finaliza la conexión con el dispositivo actual.

6. GRAFICA: Permite al usuario acceder a la función de graficado en tiempo real, de los datos que se están obteniendo.

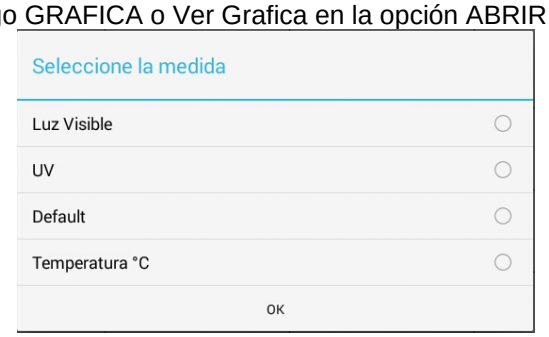
7. ABRIR: Permite al usuario abrir un archivo que contenga datos registrados con anterioridad.

Figura 7. Cuadro de dialogo ABRIR

Fuente. Fuente propia

Este cuadro de dialogo pide al usuario digitar la fecha en la que se obtuvieron los datos, ya que la aplicación asigna por nombre de archivo esta fecha. Al seleccionar la opción Aceptar se mostraran los datos de forma explícita al usuario, mientras que la opción Ver Grafica genera la gráfica correspondiente al archivo solicitado.

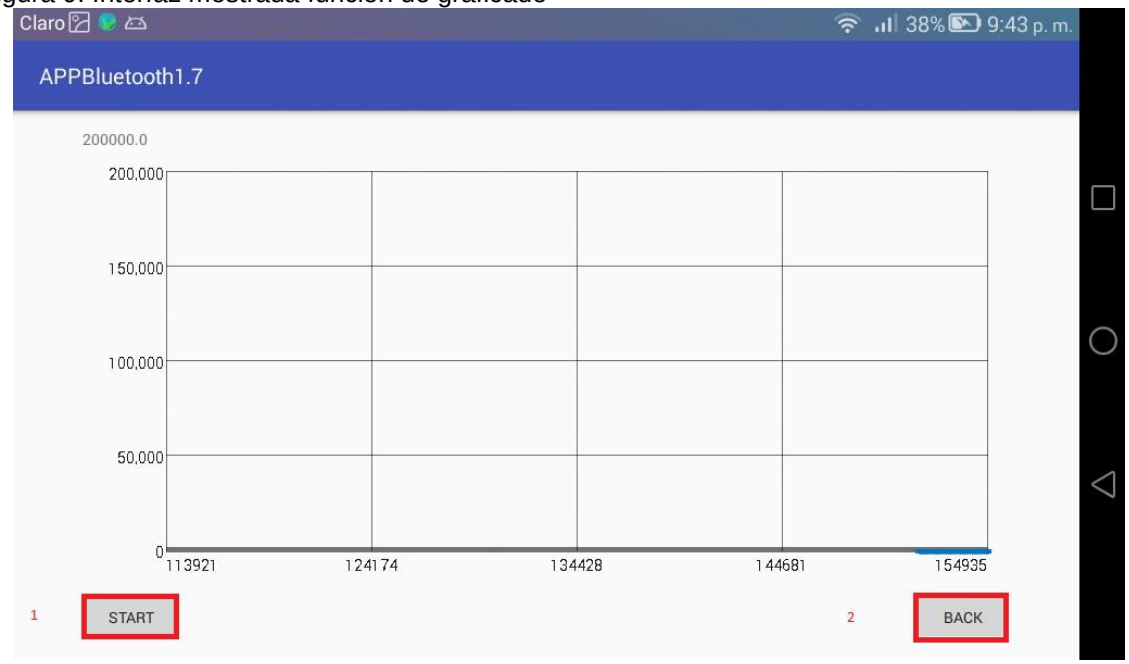
Figura 8. Cuadro de dialogo GRAFICA o Ver Grafica en la opción ABRIR



Fuente. Fuente propia

En la figura 7 se muestra el cuadro de dialogo que se genera al acceder a cualquiera de las opciones que ofrece la función de graficado, el usuario tiene la posibilidad de especificar la unidad de medida que va a ser leída, esto con el fin de ajustar de manera correcta la escala en la interfaz usada para la dicha gráfica.

Figura 9. Interfaz mostrada función de graficado



Fuente. Fuente propia

1. START: Esta función permite al usuario iniciar la función de graficado en tiempo real

2. BACK: Esta función permite al usuario volver a la interfaz inicial.

1.2 ACTIVAR Y DESACTIVAR LA CONEXIÓN BLUETOOTH

Para todas aquellas tareas relacionadas con la conexión *Bluetooth* se usaron constantes que determinan en qué estado se encuentra la conexión, los estados se definieron de la siguiente forma:

- *int STATE_BLUETOOTH*: Esta variable es la que guarda el estado actual de la conexión Bluetooth.
-
- *int BLUETOOTH_ON = 10:* Cuando la conexión Bluetooth se encuentra activa, se asigna este estado.
-
- *int BLUETOOTH_OFF = 11: C*uando la conexión Bluetooth se encuentra desactivada, se asigna este estado.

Es importante verificar la variable de estado en todas las actividades que requieran la conexión *Bluetooth* activa, ya que si no se hace esta verificación se pueden producir conflictos durante la ejecución de la aplicación.

La función más básica que se puede encontrar en una aplicación de este tipo es la de activar y desactivar la conexión Bluetooth, lo que nos va a permitir realizar esta función es la clase *BluetoothAdapter*, esta clase es una representación del dispositivo Bluetooth que tiene el dispositivo móvil, para que esta clase se vincule al dispositivo se debe crear un objeto de la clase *BluetoothAdapter*, que en este caso se creó con el nombre "BA", para luego usar el método "*getDefaultAdapter*()".Este método nos permite asignar al objeto BA el manejo del adaptador Bluetooth del dispositivo.

Una vez realizado el proceso anterior, se crea la función "*OnOff*", lo primero que se ve dentro de esta función es un condicional if, en el cual se evalúa la variable *STATE_BLUETOOTH* para conocer el estado de la conexión Bluetooth, de encontrarse en el estado *BLUETOOTH_OFF* se crea un cuadro de dialogo en que se le pregunta al usuario si desea activarlo. Al seleccionar la opción afirmativa se usa el método .*Enable*() para activarlo y se asigna el estado *BLUETOOTH_ON*.

En el caso que se encuentre en el estado *BLUETOOTH_ON* el cuadro de dialogo pregunta al usuario si desea desactivar la conexión, si se opta por la opción afirmativa se usa el método .*Disable*() y se asigna el estado *BLUETOOTH_OFF*.

Figura 10. Cuadro de dialogo función Activar/Desactivar Bluetooth

Fuente. Fuente propia

1.3 BÚSQUEDA DE DISPOSITIVOS CERCANOS

Para esta función a la cual se llamó *discover* se sigue usando el objeto BA, lo primero que se hace en esta función es un condicional *if*, en el que se verifica si actualmente está en modo búsqueda, esto se hace mediante el método .*isDiscovering*(), de no encontrarse en este estado, se evalúa la variable *STATE_BLUETOOTH*.

Una vez se ha confirmado que la conexión *Bluetooth* está activada, se usa el método .*startDiscovery*() sobre el adaptador *Bluetooth* BA.

Este proceso cuenta con 2 etapas, la primera etapa se da cuando se encuentra un dispositivo, la segunda se da cuando la búsqueda ha terminado, en este punto se usa un objeto de tipo *IntentFilter*, este objeto informa al sistema de posibles estados, o acciones que se puedan llevar acabo.

Figura 11. Estado de búsqueda de Dispositivos

Fuente. Fuente propia

El encargado de recibir estas posibles acciones será un *BroadcastReceiver* el cuál evaluara cada una de las posibles acciones que se declararon anteriormente, se asignó la constante *ACTION_FOUND* para saber que se encontró un dispositivo, una vez encontrado se agrega el nombre y la dirección del dispositivo a un *ListView* llamado "*listDevices*" y un *ArrayAdapter* llamado "*ArrayAdapterDevices*" previamente creados.

La constante *ACTION_DISCOVERY_FINISHED*, indica que la búsqueda ha finalizado, un dato a tener en cuenta es el tiempo que toma realizar el escaneo el cual es de aproximadamente de 12 segundos, una vez se cierra el cuadro de dialogo creado anteriormente.

En caso de que no se encuentre ningún dispositivo cercano, el *ArrayAdapter* estará vacío y se mostrará un mensaje, indicando que no se encontraron dispositivos.

Figura 12. Lista de dispositivos encontrados

Fuente. Fuente propia

Figura 13. Aviso de Búsqueda no exitosa

Fuente. Fuente propia

1.4 MOSTRAR DISPOSITIVOS VINCULADOS

Esta función fue hecha de manera similar a la anterior, se usa un condicional *if* con fin de verificar que la conexión *Bluetooth* esté activada, en caso de que no se mostrara de igual manera un mensaje o "*Toast*" informando al usuario que necesita activar la conexión.

Una vez la conexión esta activada se procede a limpiar el *ListView* y el *ArrayAdapter* que se usaron anteriormente, (este procedimiento se recomienda hacer para el proceso de buscar dispositivos cercanos), en este punto se debe creó un objeto de la clase *BluetoothDevice* con el nombre "*BDdevice*", al usar el método ".*getBondedDevices*()" se obtienen las direcciones, nombres, estados y clases de los dispositivos que se han vinculado con anterioridad, luego se agregan los nombres y direcciones a "*listDevices*" y "*ArrayAdapterDevices*", con el fin de visualizarlos en el *ListView* creado en la parte gráfica de la aplicación.

Figura 14. Lista de dispositivos vinculados

Fuente. Fuente propia

Desde este punto es importante tener en cuenta el concepto de "*Socket*", este "*Socket*" es un objeto que mediante sus métodos permite gestionar los procesos de comunicación entre dispositivos, procesos como envío y lectura de datos, conexión con otros dispositivos, almacenamiento de datos como nombre y dirección del dispositivo al que se encuentra conectado actualmente, entre otros..

1.5 VERIFICAR ESTADO ACTUAL

Esta es la primera función que utiliza un objeto de la clase *BluetoothSocket*, este objeto se creó bajo el nombre *BTSocket*, esta función se asignó al botón VERIFICAR, cuando se llama esta función se usa un condicional if, para verificar la variable *STATE_BLUETOOTH,* si la conexión Bluetooth se encuentra activa se usa otro condicional *if,* para evaluar la variable de estado STATE_CONNECTION, y saber si hay algún dispositivo conectado.

Figura 15. Aviso Verificación de estado actual

Fuente. Fuente propia

1.6 CONECTAR A UN DISPOSITIVO

Para todas aquellas tareas relacionadas con la conexión a otro dispositivo se usaron constantes que determinan en qué estado se encuentra la conexión, los estados se definieron de la siguiente forma:

- *int STATE_CONNECTION:* En esta variable se guarda el estado de la conexión con algún dispositivo

- int DEVICE_CONNECTED = 20: Se asigna esta estado cuando hay una conexión establecida con otro dispositivo.

- int DEVICE_DISCONNECTED = 21: Se asigna esta estado cuando no hay una conexión establecida con otro dispositivo.

- *int DEVICE_DISCONNECTED_MANUAL = 22:* Se asigna este estado cuando el usuario desactiva la conexión de forma voluntaria.

Se usan dos variables de estado para la desconexión con el fin de identificar en qué momento se perdió la conexión por factores ajenos al usuario, y en qué momento por voluntad del usuario, de esta forma la aplicación identifica en que momentos debe intentar recuperar la conexión y en qué momentos no.
El proceso de conexión consta de dos etapas, la primera es la obtención del dispositivo seleccionado, y el segundo es la conexión misma.
La primera etapa inicia desde el *ListView* en el que se visualizan los dispositivos, ya sea encontrados o vinculados.

El sistema debe reconocer cual es el dispositivo al que se desea conectar, para eso se usa el método "*OnItemClickListener*" sobre el *ListView* creado, este método nos permite realizar alguna acción cuando se selecciona algún ítem que se encuentre en dicha lista, también permite obtener datos como la posición del ítem que el usuario seleccionó, este dato se guarda en la variable "pos", con el fin de usarla más adelante para identificar el dispositivo, una vez realizado esto, se crea un cuadro de dialogo, este pregunta al usuario si desea conectarse al dispositivo seleccionado, de ser afirmativa la respuesta se inicia el proceso de conexión.

Figura 16. Consulta al usuario para conexión

Fuente. Fuente propia

La segunda etapa inicia con la creación y el inicio de la clase *ConnectThread* que contiene un *Thread* o también conocido como hilo, el cual se encarga de realizar el proceso de conexión, al llamar esta clase se le debe ingresar la dirección MAC y el estado actual de la conexión.

El Thread inicia con un "*try and catch*", este es un proceso en el que el sistema intenta realizar alguna tarea, la cual se define en el "*try*", en caso que de que se presente algún error realizando dicho proceso/s, se responde con otra tarea que se define en el "*catch*".

En este caso el *try* contiene dos tareas, la primera tarea es asignar el dispositivo que seleccionamos en la lista al *Socket* que creamos previamente (*BTSocket*), es importante asignarle un *UUID* es dicho Socket, este *UUID* es un código estándar de 128 bits, este código es un identificador que se usa para reconocer al dispositivo dentro del sistema, este código se definió como una constante al inicio del programa. La segunda tarea es iniciar una tarea que permitirá a la aplicación estar pendiente de cualquier dato que quiera entrar se hablará de forma más detallada sobre este proceso más adelante.

Como respuesta al fallo de estas tareas, se mostrara un mensaje al usuario informando de este problema.

Después de esto, se usa otro "*try and catch*", el *try* tendrá la tarea de realizar la conexión, esto se hace mediante el método ".*connect*()", si se realiza de forma exitosa la conexión, se mostrara un mensaje con el nombre del dispositivo al que se conectó. De igual manera al "*catch*" anterior, se mostrara un mensaje al usuario informando del problema en caso que exista un error.

Finalmente se crean dos objetos de clase InputStream y OutputStream, los cuales cumplen la función de buffers en los que el Socket va a almacenar los datos que se van a enviar o recibir.

Figura 17. Conexión exitosa

Fuente. Fuente propia

Una vez finalizado este proceso, se asigna a la variable STATE_CONNECTION el valor correspondiente a si se logró o no establecer la conexión con el dispositivo deseado.

Figura 18. Conexión fallida

Fuente. Fuente propia

1.7 DESCONECTAR DE UN DISPOSITIVO

La desconexión de un dispositivo se puede dar de dos formas, una es en la que el usuario desea desconectarse de un dispositivo, la otra en la que se pierde la conexión de forma inesperada.

1.7.1 Desconexión voluntaria

Cuando el usuario desde desconectarse de un dispositivo de forma voluntaria, se usa el método desconectar, el cual inicia con una verificación del estado de la conexión Bluetooth y de la conexión con el dispositivo, para realizar la desconexión los estados deben ser *BLUETOOTH_ON* y *DEVICE_CONNECTED,* una vez realizada la verificación se usa un *try and catch* con la intención de cerrar las variables *mmInStream* y *mmOutStream,* las cuales se encargan de almacenar los datos que se envían y reciben.

Una vez realizado lo anterior mencionado se procede a usar los métodos *cancel()* de los *Thread* que se hayan usado anteriormente para así poder detener los procesos que estaban realizando, y de esta forma evitar futuros conflictos en la aplicación, es aconsejable usar el método esperar a que un *Thread* realice todos sus procesos para iniciar otro nuevo y así evitar errores, esta espera se lleva acabo aplicando sobre el Thread el método join().
Luego de esto se procede a limpiar los *ArrayList y ArrayAdapter* para evitar que queden almacenados en estos datos anteriores, también de debe establecer el estado actual *DEVICE_DISCONNECTED_MANUAL.*

1.7.2 Desconexión inesperada

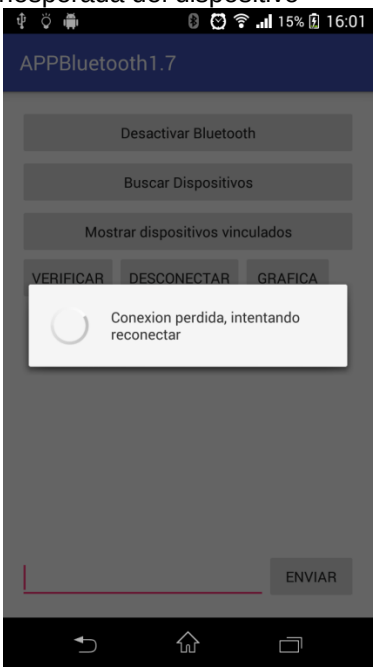
Esta desconexión se detecta cuando el dispositivo no recibe datos aun estando conectado, cuando sucede esto se usa el método *lost()*, lo primero que se hace en este método es establecer el estado *DEVICE_DISCONNECTED*, luego se abre un cuadro de progreso informando al usuario que se está intentando recuperar la conexión. Mientras se muestra el cuadro de progreso se usa un ciclo *While*, en el que se va a intentar recuperar la conexión usando de nuevo el *Thread* de conexión *ConnectThread,* este *while* va a intentar reconectar el dispositivo mientras que el estado de la conexión sea *DEVICE_DISCONNECTED,* o haya pasado aproximadamente 10 segundos, este tiempo de espera se logra mediante el uso de un contador que aumentara cada segundo.

Si la conexión no se puede recuperar se informa al usuario de esto, se establecen los estados correspondientes a la desconexión y se cierran todos los Thread usados anteriormente.

Figura 19. Aviso de desconexión voluntaria del dispositivo

Fuente. Fuente propia

Figura 20. Aviso de desconexión inesperada del dispositivo



Fuente. Fuente propia

2 RECEPCIÓN, LECTURA Y ENVIO DE DATOS

2.1 RECEPCIÓN DE DATOS

Para la recepción de datos se implementó una clase con el nombre de *RxDataB* la cual contiene un Thread que realiza la recepción de los datos, se crea e inicia un objeto de esta clase cada vez que se requiere la recepción de datos.

Este proceso de creación e iniciación se hace inmediatamente después que el proceso de conexión con otro dispositivo se hace de manera exitosa, se debe hacer después de verificar el estado de la variable *STATE_CONNECTION* para confirmar si la conexión con otro dispositivo se llevó a cabo con éxito, de no ser así se informa al usuario que no hay un dispositivo para recibir datos.

Cuando se va a crear un objeto de esta clase, se le deben ingresar dos parámetros, el primero es un objeto de tipo *Socket,* el cual se obtiene del *Thread* llamado *ConnectThread*, el cual se usó para realizar la conexión al dispositivo que suministra los datos, el segundo parámetro es un objeto de la clase *Handler,* este objeto nos permite conocer el estado de la recepción de los datos.

Lo primero que hace este *Thread RxDataB* obtener los valores que se encuentran almacenados en el *Socket* que le fue ingresado, estos valores se almacenan en la variable mmInStream la cual cumple la función de buffer para almacenar los datos

que se reciben. Después de esto sigue el método *run* en el que se inicia la lectura de los datos recibidos.

La lectura de datos se realiza de la siguiente forma: al ser una cadena de números y/o letras también llamados caracteres, se debe recibir y evaluar cada carácter para después almacenarlo. Para que la aplicación reconozca cuando dato se terminó de recibir se debe enviar un carácter de fin del dato, que para este caso se usó el carácter "#".

Para implementar lo anterior mencionado es importante el uso de las variables *cPos* y *buffer*, en *cPos* se almacena la posición del carácter que se está evaluando en la cadena, mientras que *buffer* almacena el carácter propio se usó un *try and catch*, el cual inicia con un condicional *if* en el que se pregunta si *buffer* es diferente a *null*, es decir si contiene algún dato, si esto se cumple se sigue con el siguiente paso, de lo contrario se envía al *handler* una variable de estado llamada *CONNECTION_LOST*, con la intención de posteriormente verificar la conexión.

En el segundo paso se almacena el carácter y la posición en *buffer*, después se usa otro condicional *if,* en el que se pregunta si el carácter almacenado es igual a "#", si se cumple entonces se crea una variable de tipo *String* llamada *data,* en la que se almacena el dato completo, finalmente se envía la variable *data* al handler, mediante el método *obtainMessage* en el que se le dan como parámetros la variable que indica el estado actual, la cantidad de posiciones del datos y la variable *data*, es importante tener en cuenta que se debe eliminar el carácter que indica el fin del dato, para así evitar que el usuario lo vea, a la vez que se evitan errores dentro del programa.

Se debe tener en cuenta que la frecuencia de muestreo la establece el usuario, pero deber ser aproximadamente 5Hz como máximo, es decir un dato cada 200 milisegundo. Esta frecuencia es la máxima que el dispositivo es capaz de procesar y mostrar el dato.

2.2 INTEGRIDAD DE LOS DATOS

Una de las tareas más importantes que se debe tener en cuenta a la hora del diseño la esta aplicación es la caracterización de los datos obtenidos, esto con el fin de evitar que se almacenen datos repetidos o erróneos, se debe tener en cuenta que es posible obtener datos erróneos debido a fallos del sistema que está realizando la captura y envío de datos al dispositivo móvil, en este caso la aplicación los aceptará de igual forma, lo que permitirá evidenciar estos fallos en el sistema, por lo tanto los únicos datos erróneos que se serán tratados dentro de la aplicación serán aquellos se sean producidos por problemas dentro de la misma.

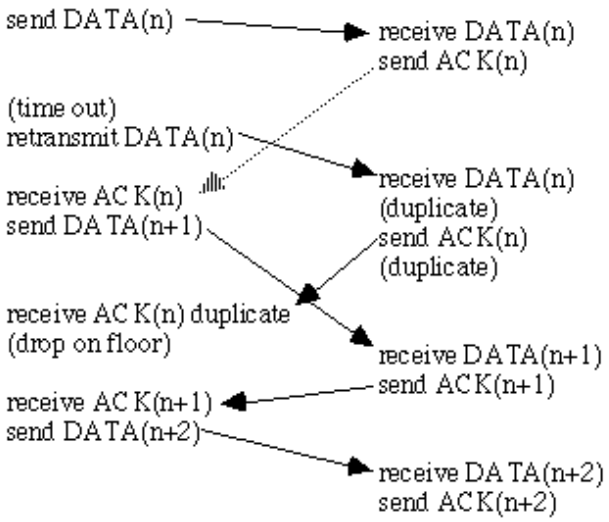
Para garantizar la integridad de los datos se debe realizar un diseño de dato, este proceso se basa en 3 técnicas, validación, normalización e identificación.

2.2.1 Proceso de validación

Para tener un mayor control sobre el flujo de datos entre el sistema emisor de datos y la aplicación se usa un protocolo de ventana deslizante. Este protocolo busca evitar errores en la comunicación, ya sean producidos por conflicto entre las velocidades de transmisión y recepción o incluso pérdida de datos.

El funcionamiento del protocolo en el sistema emisor se basa en el uso de un buffer con determinado tamaño que almacena aquellos datos que están listos para ser enviados, al ser enviado el dato la aplicación debe enviar un mensaje de confirmación ACK, el cual indica que se recibió el dato, cuando el sistema emisor recibe esta confirmación envía el siguiente dato, si el dato no es recibido por la aplicación y por tanto esta tampoco envía un mensaje de confirmación el sistema emisor puede contar con otro buffer que almacena los datos que han sido enviados pero no confirmados por el receptor, para luego reenviarlos, también cuenta con un tiempo de límite de espera para el mensaje ACK, al expirar este tiempo reenvía el dato.

Diagrama 3. Funcionamiento básico del protocolo de ventana deslizante



Fuente. Disponible en http://www.docjava.com/cpe471/lectures/datalink/datalink.htm imagen 28 de Julio 2016

El sistema de recepción cuenta con un buffer que almacena el dato recibido, para luego compararlo con una lista de datos recibidos previamente y verificar que no existan datos repetidos, esto se lleva a cabo en el proceso de normalización.

2.2.2 Proceso de normalización

El proceso de normalización tiene como función eliminar datos repetidos, usando como referencia la hora en que se recibió el dato, cuando se presenta el caso en que existe más de un dato con la misma hora de recibido quiere esta da a entender que el buffer llamado *"data"* encargado de almacenar los datos recibidos contenía datos previos al que se estaría recibiendo actualmente.

Una vez iniciada la recepción de datos se propone como solución el algoritmo mostrado en el *diagrama 3*.

Si aún después de realizar el proceso anterior se almacenan en el buffer datos con la misma hora se realiza una verificación antes de almacenar el dato en una lista, esta verificación se hace mediante un condicional "*if*", en el que se pregunta si algún elemento de la lista ya contiene la hora del dato actual, de ser así no se almacena el dato actual en la lista.

2.2.3 Proceso de identificación

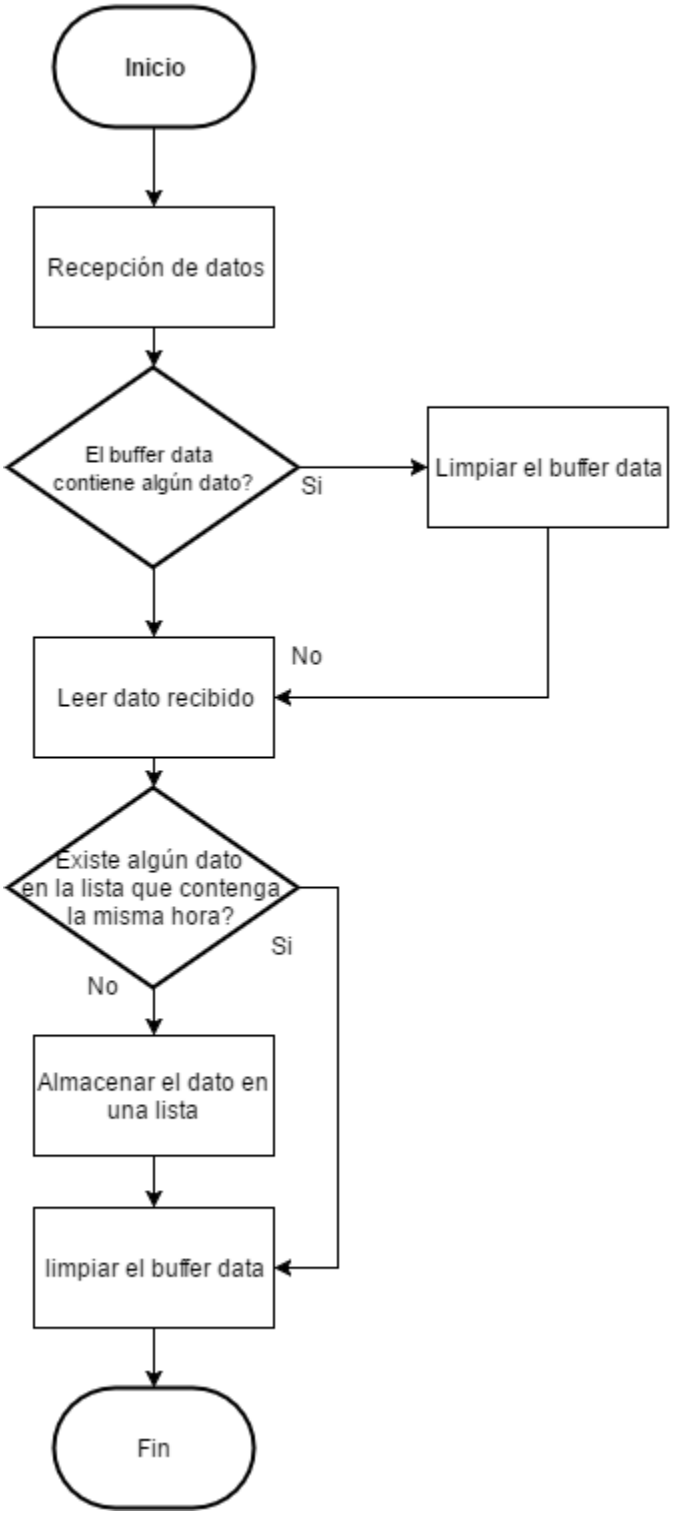
Un vez se reciben los datos de manera correcta se busca generar la mayor cantidad de información posible con el fin de facilitar posteriores procesos de interpretación y documentación que el usuario desee realizar.
Para esto se le darán atributos a cada dato obtenido como se muestra en la Tabla 2.

Tabla 2. Atributos otorgados a cada dato que se recibe

Posición en la lista	Fecha	Hora	Valor obtenido	Unidad de medida
Numero de posición	Año Mes Día	Hora Minuto Segundo	Dato	Unidad de medida

Fuente. Fuente propia

Diagrama 4. Algoritmo de normalización para integridad de datos



Fuente. Fuente propia

2.3 ENVÍO DE DATOS

El usuario tiene la posibilidad de escribir un dato en un textView o campo de texto llamado *dataS* y enviarlo al dispositivo al que se encuentre conectado,

El Envío de datos se realiza en el método *send*, el que fue asignado al botón enviar, como en gran parte de los métodos se han creado anteriormente se debe verificar el estado de la conexión Bluetooth y del dispositivo usando condicionales *if*, para realizar la tarea de envío las variables de estado *STATE_BLUETOOTH y STATE_CONNECTION* deben ser *BLUETOOTH_ON y DEVICE_CONNECTED* respectivamente. De no ser así se le informa al usuario que no hay conexión Bluetooth activa, o no hay dispositivos conectados.

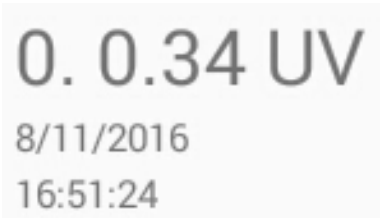
Una vez realizada la verificación de estados se usa sobre *dataS* los métodos *getText().toString().getBytes()*, con el fin de capturar el dato que el usuario ingreso en el campo de texto, para luego usarlo en el método *write().* Después de capturar el dato hay que enviarlo, para esto es necesario usar la variable mmOutStream, la cual representa el *buffer* que usa el *Socket* previamente creado en el proceso de conexión, usando el método *write()* en el que se debe ingresar como parámetro el dato que se capturó anteriormente para que el *Socket* haga el proceso de envío

3 VISUALIZACIÓN DE DATOS

Para el caso de este ejemplo se va a crear una lista de objetos que serán organizados de forma vertical. El objeto constará de su posición en la lista, dato, la fecha, la hora en la que se obtuvo y la unidad de medida, como se muestra en la Figura 20.

El dato se obtiene de un *buffer* creado en el handler que tiene por nombre readMessage usando el método *msg.obj* captura el dato y lo almacena, para luego asignarlo como parámetro en el método *nuevoDato*, el cual se encargará de crear el objeto a mostrar y almacenarlo en un *ArrayList.*

Figura 21. Visualización del objeto



Fuente. Fuente propia

3.1 CREACIÓN DEL OBJETO DATO

Como primer paso para esto se debe usar la herramienta *LinearLayout*, la cual establece una alineación vertical u horizontal de los elementos en la pantalla, a diferencia de otros tipos de *Layout*, este no permite la superposición de elementos. Los parámetros usados fueron:

- *Orientation*: Este parámetro define de qué forma se van a alinear los elementos en la pantalla, se le puede asignar los valores vertical y horizontal.
- *Padding*: Este parámetro define el espacio que hay entre los bordes de la pantalla y el borde del objeto.

Se debe tener en cuenta que se debe crear un archivo *.xml* el cual contiene el diseño del objeto que se va a usar en la lista, en este ejemplo el archivo que contiene el diseño del objeto se llama *item_data.xml*

El *LinearLayout* que se usó se le asignó una orientación vertical, para ordenar los elementos en de esa forma, dentro de este *Layout* se encuentran 3 *TextView*, para mostrar el dato, la fecha y la hora.

3.2 VISUALIZACIÓN EXPLICITA DEL DATO

Lo siguiente es crear una clase que contenga las variables que va a tener el objeto es decir, dato, fecha y hora, esta clase se va a usar para crear los objetos en una *ArrayList*.

Una vez creado el objeto en su respectivo archivo .*xml*, ahora se debe usar un *ListView*, un *ListView* es un sistema que permite mostrar una serie de elementos en forma de lista, en este punto se creó un *ArrayList* llamado *arrayListDato* que contiene cada elemento que se quiere mostrar, como se mencionó anteriormente cada objeto consta de tres *TextView*, se creó un método llamado *ItemAdd*, el cual se crea cada uno de los elementos y se asignan al *ArrayList* llamado *Lista_Items*.

Luego tener un *ArrayList* con los objetos, se debe usar un *ArrayAdapter*, esta instrucción permite vincular el *ArrayList* creado con el *Listview* que se usará para mostrar los objetos, por esto se debe indicar que contexto en el que se va a mostrar la lista, el contexto en el que se encuentra el objeto y el *ArrayList*.
Por último se usa el método *getView* para leer las posiciones del *listView*, cuando se encuentra una posición vacía se asigna el objeto creado en el archivo *item_data*.

Figura 22. Ejemplo de visualización explícita

Fuente. Fuente propia

3.3 VISUALIZACIÓN GRÁFICA DEL DATO

En la visualización gráfica se usó una librería externa creada por Jonas Gehring llamada *android-graphview*.

Para hacer uso de esta herramienta es importante agregarla en *Android Studio*, la forma que usó y que es las más sencilla es agregándola la siguiente línea de código en el archivo *build.gradle(Module: app)* del proyecto: compile 'com.jjoe64:graphview:4.1.0'. De esta manera ya están disponibles todos los componentes de la librería.

La programación correspondiente a la visualización gráfica debe hacerse en una nueva actividad, lo que implica que la conexión se puede perder en el cambió de *layout*, por esto se almacena en una variable la dirección MAC del dispositivo al que se estaba conectado con el fin de restablecer la conexión en la nueva actividad.

Se puede acceder a esta opción de graficado mediante el botón *GRAFICA*, el cual crea un *intent* que permite acceder a la nueva actividad, lo que para el usuario significa que va a acceder a una nueva ventana en la aplicación, dentro de este *intent* se va a almacenar la dirección *MAC* del dispositivo al que se encuentra conectado, de esta forma se podrá recuperar este dato al iniciar la nueva actividad.

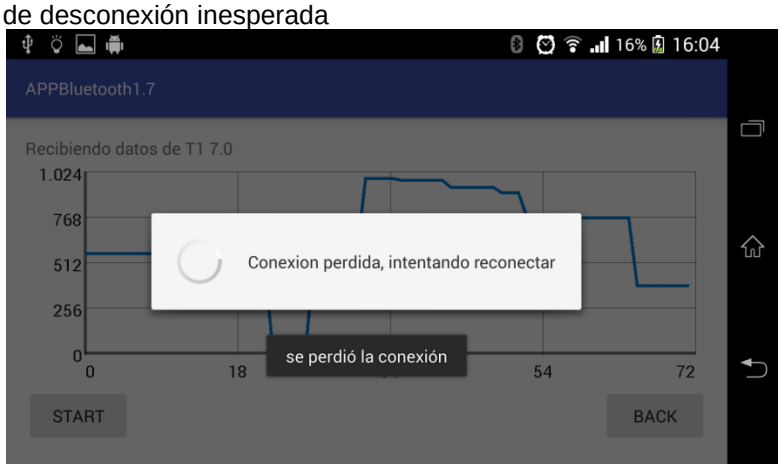
3.3.1 Restablecimiento de la conexión

El almacenamiento de esta dirección se hace en el método llamado *grapA*, en el que se crea el *intent* y se almacena la MAC, se crea un objeto de la clase *Intent* el cual se nombró como *iGraph*, mediante el método*.putExtra()* se almacena la *MAC* en una variable de tipo *String,* con la etiqueta *"MAC",* esta variable se usa posteriormente para identifica el dato que se guardó.

Una vez iniciada la nueva actividad la primera tarea será restablecer la conexión con el dispositivo, para esto se debe recuperar la dirección *MAC*, se crea un objeto de la clase *Intent* el cual fue llamado *i_graph*, a diferencia del paso anterior se aplica el método *.getIntent() y* así obtener lo almacenado anteriormente. Para obtener puntualmente la dirección guardada se crea una variable de tipo *String* y se usan los métodos *.getExtras().getString("MAC")* en *i_graph*, de esta forma ya está almacenada la dirección *MAC* del dispositivo que estaba conectado anteriormente.

Habiendo realizado este proceso se asigna como parámetro la dirección almacenada en los Threads usados anteriormente para los procesos de conexión y recepción de datos, la ventaja en este punto es que el usuario no tiene intervenir en todos los pasos para realizar la conexión.

Figura 23. Aviso de desconexión inesperada



Fuente. Fuente propia

3.3.2 Configuración de la gráfica.

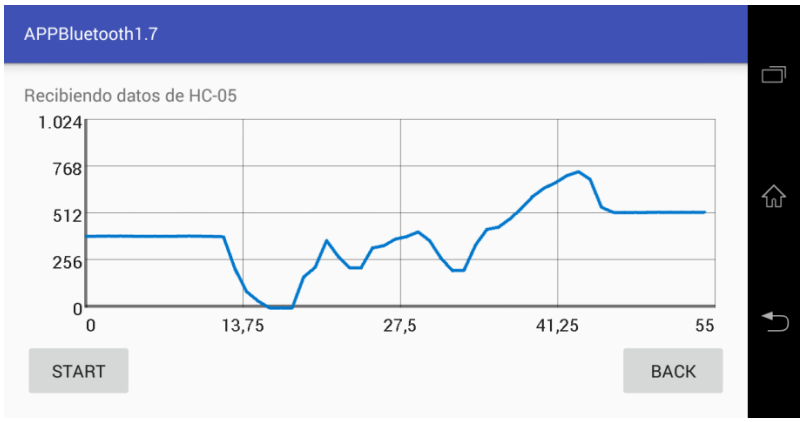
En esta etapa se definen los aspectos que definen la interfaz gráfica que se mostrará al usuario, primero se creó un objeto de la clase *GraphView* llamado *mGraphView*, esta será la gráfica misma, luego un objeto de la clase *LineGraphSeries* llamado *series*, este objeto es la línea que se va mostrando en la gráfica, luego ser creados estos objetos se debe asignar el objeto *series* a la gráfica, esto mediante el método *.addSeries().*

El método *getViewport()* es el que permite tener acceso a otros métodos para editar la interfaz de la gráfica, habiendo mencionado esto se definieron los valores máximo y mínimo que se van a registrar en el eje Y de la gráfica, mediante el método *getViewport().setYAxisBoundsManual(true)* se declara que se van a asignar valores de forma manual, con*.getViewport().setMinY(); y .getViewport().setMaxY()* se definen los valores.

Los últimos parámetros a definir son *Scrollable y Scallable*, estos parámetros permiten al usuario desplazarse de manera horizontal en la gráfica, de la misma manera hacer acercamientos y alejamientos.

Y finalmente para la captura de datos en la gráfica se hace un proceso similar a la visualización explícita, ya que se usa un *handler* para la recepción de datos la diferencia con la visualización explícita es que no se debe crear un objeto para visualizar sino que se usa el método *appendData()* en *series* para agregar y mostrar cada dato que llega.

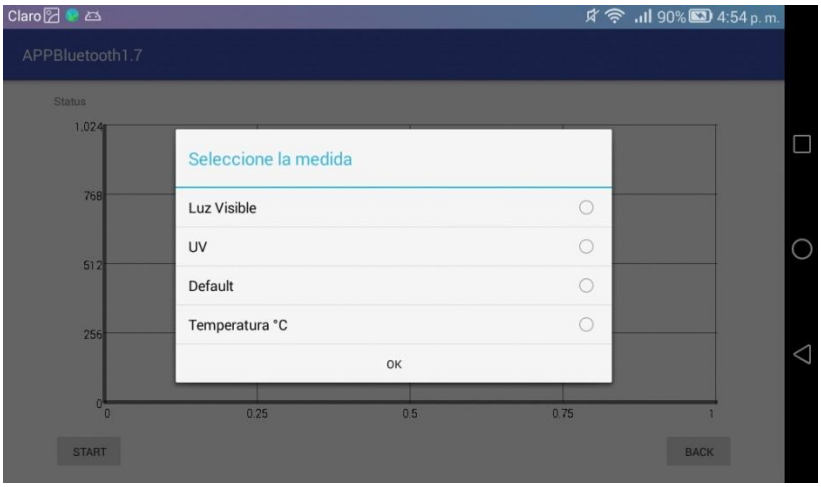
Figura 24. Ejemplo de visualización gráfica



Fuente. Fuente propia

Con el fin de mantener la integridad de los datos se da la opción al usuario de seleccionar el sistema de medida que quiere visualizar, esto mediante un cuadro de dialogo con las respectivas opciones como se puede ver en la Figura 24.

Figura 25. Lista de posibles sistemas de medida



Fuente. Fuente propia

Tabla 3. Caracterización de los posibles tipos datos a registrar

Medida	Unidad	Resolución ADC	Resolución por bit
Temperatura	Celsius (°C)	8 bits	$19.5mV/_{bit}$
Radicación UV	Índice UV	10 bits	$488mV/_{bit}$
Luz visible	Lux (E_v)	10 bits	$488mV/_{bit}$

Fuente. Fuente propia

4 ALMACENAMIENTO DE DATOS

4.1 UBICANDO LA RUTA DE LA MEMORIA SD

En Android el almacenamiento externo no siempre aplica a dispositivos extraíble como los son las memorias SD, algunos dispositivos asignan una parte de su propia memoria interna para funciones de memoria externa, como lo puede ser la creación de archivos por parte de aplicaciones diferentes a las propias del dispositivo, o archivos de tipo multimedia que el usuario desee almacenar, esto se hace con el fin de evitar tener un acceso sencillo a los archivos que son propios del funcionamiento del sistema operativo.

Android Studio ofrece la posibilidad de obtener dicha ruta a la memoria externa mediante el método *Environment.getExternalStorageState()*, pero por lo anterior mencionado no se recomienda hacerlo de esta forma, ya que puede estar apuntando a la dirección de la memoria interna más no a la memoria extraíble.

Por esto se creó el método *verificaRutaSd(),* este método se basa en un serie de condicionales if, en cada uno que se crea un objeto de la clase *file* con una posible ubicación de la memoria SD, se evalúan diferentes rutas ya que no todos los dispositivos cuentan con la misma, una vez creado el objeto se pregunta si ya existe, en caso de ser así el método *verificaRutaSd()* retorna la dirección encontrada.

4.2 CREACIÓN Y ESCRITURA DEL ARCHIVO DE TEXTO EN LA MEMORIA SD

Para la creación del archivo de texto se usa el método *guardarDatos()*, al usar este método debemos asignarle como parámetros la fecha de día en que se están tomando los datos, y el *ArrayList* donde estamos almacenándolos.

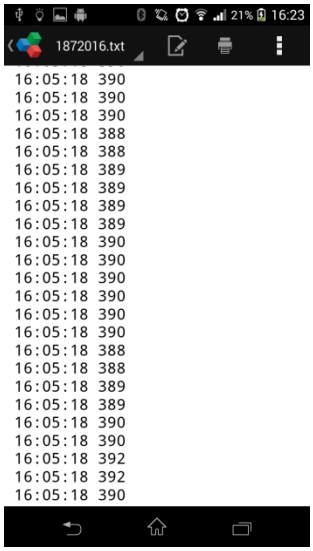
La fecha se usará para asignarle el nombre al archivo, para que el usuario pueda reconocerlo con más facilidad, se crea una variable de tipo String en la que se definirá el nombre de la siguiente manera: Datos (fecha de creación).txt. Luego de esto se debe crear el archivo, se crea mediante un objeto de la clase *File,* al crear este objeto se le debe asignar la ruta donde se va a crear la cual obtuvimos al inicio, y el nombre.

Una vez realizado esto se debe crear un objeto de la clase *FileOutputStream*, de esta manera se especifica se este archivo es para escribir en la ruta especificada, luego se crea un objeto de la clase *OutputStreamWriter*, para realizar la escritura dentro del archivo que se creó.

En la escritura de datos dentro del archivo se usa un ciclo *for,* en el que para cada dato leído del ArrayList que contiene los datos se escribe un nuevo dato mediante el método *.Write()*, a este método se le asigna como parámetro el dato que se quiere escribir el cual consta de la hora y el dato mismo, al finalizar el proceso de escritura se debe cerrar el proceso de escritura con el método *.close().*

Si el proceso de creación del archivo de texto o su escritura falla, se informa al usuario que no se pudo realizar la creación del archivo.

Figura 26. Ejemplo de archivo creado



Fuente. Fuente propia

4.3 LECTURA DE ARCHIVOS DE TEXTO CREADOS CON ANTERIORIDAD

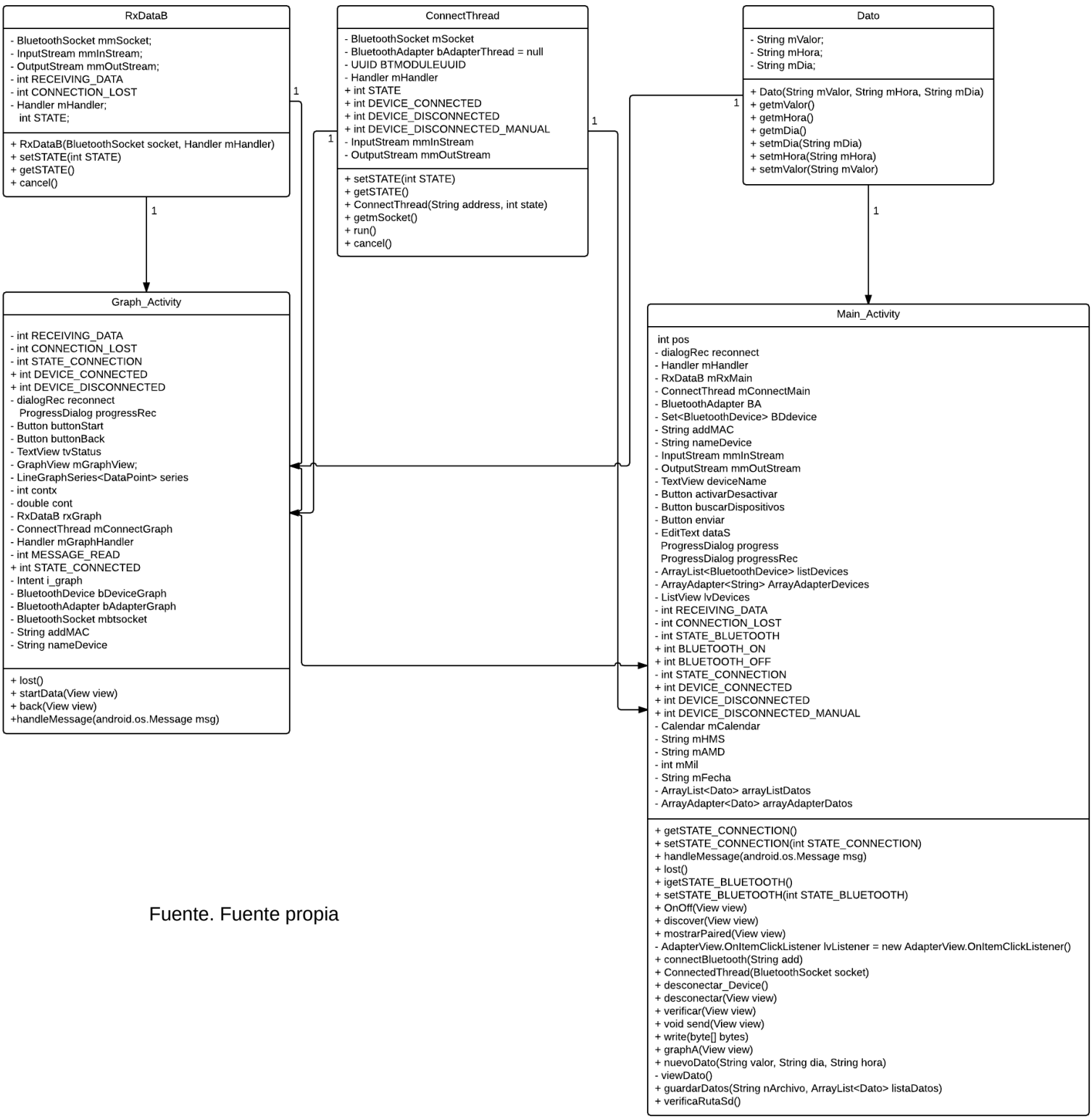
Una de las funciones que tiene esta aplicación es la lectura de archivos creados anteriormente, se puede visualizar tanto de forma gráfica como de forma explícita los datos almacenados en un archivo.

El desarrollo de esta función se da en el método *leerArchivo* e inicia con la creación de un objeto de tipo *ArrayList* en el cual se va a almacenar los datos obtenidos del archivo abierto, una vez creado este objeto se crea un nuevo objeto de tipo file, este se usará para realizar la búsqueda del archivo en el directorio externo del dispositivo, el usuario deberá ingresar la fecha en un cuadro de dialogo creado anteriormente, el cual aparecerá en el momento que desee abrir un archivo, ya que la aplicación le asigna por nombre al archivo la fecha en la que se hizo la lectura de los datos.
El objeto de tipo *file* que fue creado anteriormente captura la fecha que ingresó el usuario con el fin de realizar la búsqueda, se usa un condicional *if* para verificar si el archivo existe, de no ser así se le informa al usuario que no existe, de lo contrario, se procede a realizar la lectura de cada línea del archivo, para llevar acabo la lectura de este archivo se usa un objeto de tipo *BufferedReader*, en el que se almacena cada una de las líneas.
Esta lectura se lleva acabo usando un ciclo *while*, el cual permitirá este proceso de lectura mientras el buffer que almacena el contenido de las líneas sea diferente a *null,* es decir mientras no se encuentre vacío, esta líneas se irán almacenando en el *ArrayList* creado anteriormente, para luego mostrarlo en pantalla mediante un *listView*.

El proceso mencionado anteriormente aplica en la visualización explícita, para la visualización gráfica se debe realizar una conversión en los tipos de variables, ya que al leer los datos del archivos estos se capturan en variables de tipo *String,* esto no resulta útil para realizar el graficado, ya que esta librería necesita que los datos a mostrar se ingresen en tipo *Float* o *Double,* esta conversión se realiza usando el método *formatLabel.*
El último aspecto que se debe tener en cuenta es la longitud de datos leído, ya que se leen de derecha a izquierda dígitos del dato, esto debido a que se leen con el siguiente formato: hora/minuto/segundo dato (hh/mm/ss_xxxx), al variar la longitud del dato que se quiere visualizar se pueden presentar errores de lectura, para evitar este problema se hace una lectura mediante un condicional *if*, en este se pregunta por la longitud de la línea de texto leída, si la línea de texto contiene 10 caracteres significa que el dato contiene uno digito y se lee el último carácter de la línea, en caso de que la línea tenga 11 caracteres significa que contiene dos dígitos, por ende se lee los dos últimos de la línea, de esta forma se hace la verificación del número de dígitos de cada dato leído.

Finalmente se realiza el mismo proceso mencionado en la sección “VISUALIZACIÓN GRÁFICA” para la adición de datos a la gráfica.

Diagrama 5.Diagrama de clase de la aplicación



Fuente. Fuente propia

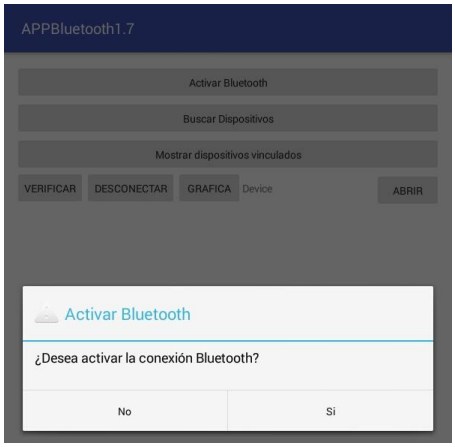
5 PRUEBAS REALIZADAS Y RESULTADOS OBTENIDOS

5.1 INTERFAZ USADA

En esta sección se mostrará los pasos para configurar la aplicación de tal forma que permita visualizar los datos en tiempo real.

1. Activar la conexión Bluetooth en el dispositivo móvil, mediante el botón "*Activar Bluetooth".*

Figura 27. Cuadro de diálogo para Activar conexión Bluetooth



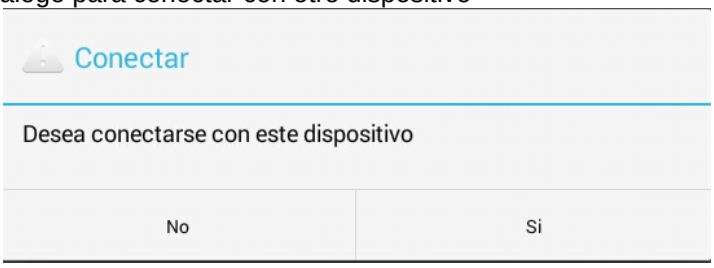
Fuente. Fuente propia

2. Encontrar el dispositivo al que se desea conectar. Para esto se debe recurrir a los botones "*Buscar Dispositivos*" o "Mostrar dispositivos vinculados" en caso de ya haber realizado una conexión previa. Una vez hecho esto aparecerá una lista de dispositivos disponibles.

Figura 28. Lista de dispositivos disponibles

Fuente. Fuente propia

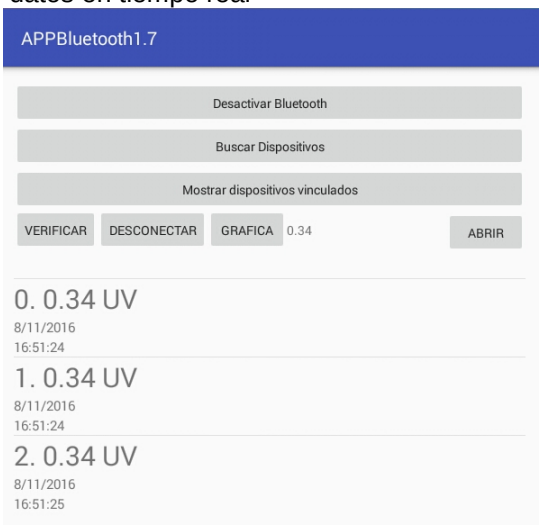
Figura 29. Cuadro de diálogo para conectar con otro dispositivo



Fuente. Fuente propia

3. Una vez realizados los pasos anteriores, la aplicación mostrara los datos que está recibiendo en tiempo real con el formato que se propuso en la tabla 2 (Posición en la lista, Fecha, Hora, Valor obtenido, Unidad de medida).
4.

Figura 30. Visualización de datos en tiempo real



Fuente. Fuente propia

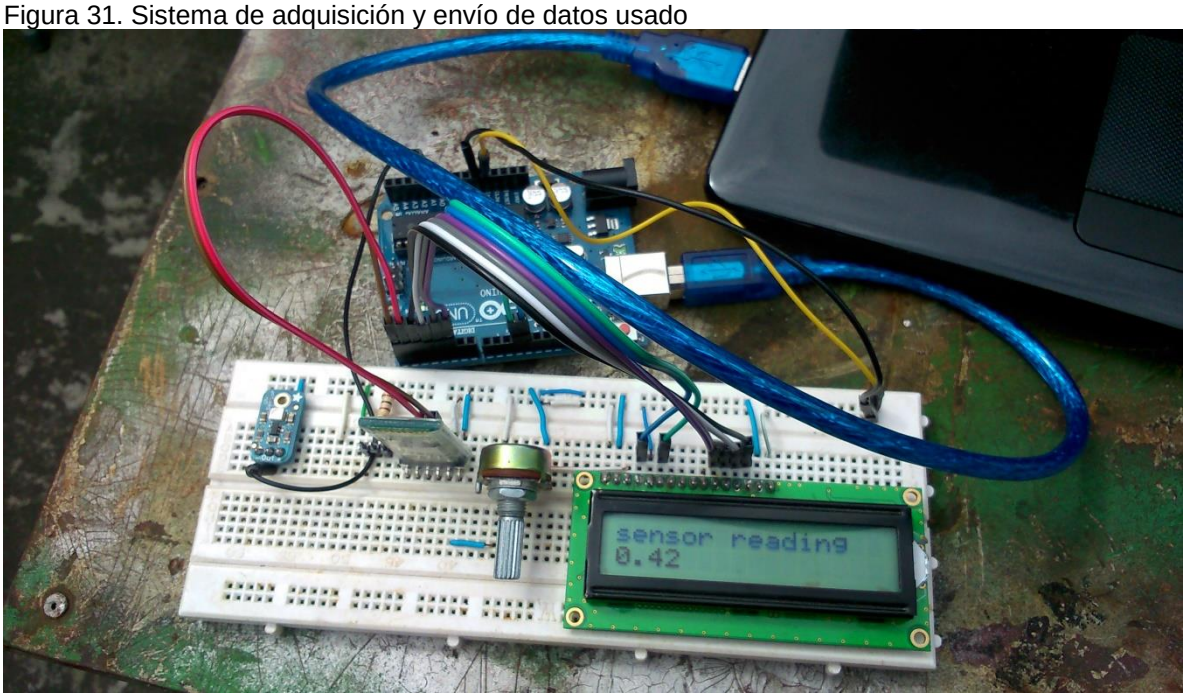
5. Para acceder a la función de graficado en tiempo real se debe recurrir al botón "*GRAFICA"* y seleccionar la unidad de medida que se requiere para ajusta la escala.

5.2 PRUEBAS REALIZADAS

Lectura de datos de radiación UV

Para poner a prueba el funcionamiento de la aplicación, se usó un sistema de captura de datos de radiación UV, el cual enviaba datos a la aplicación cada dos segundos.

Figura 31. Sistema de adquisición y envío de datos usado

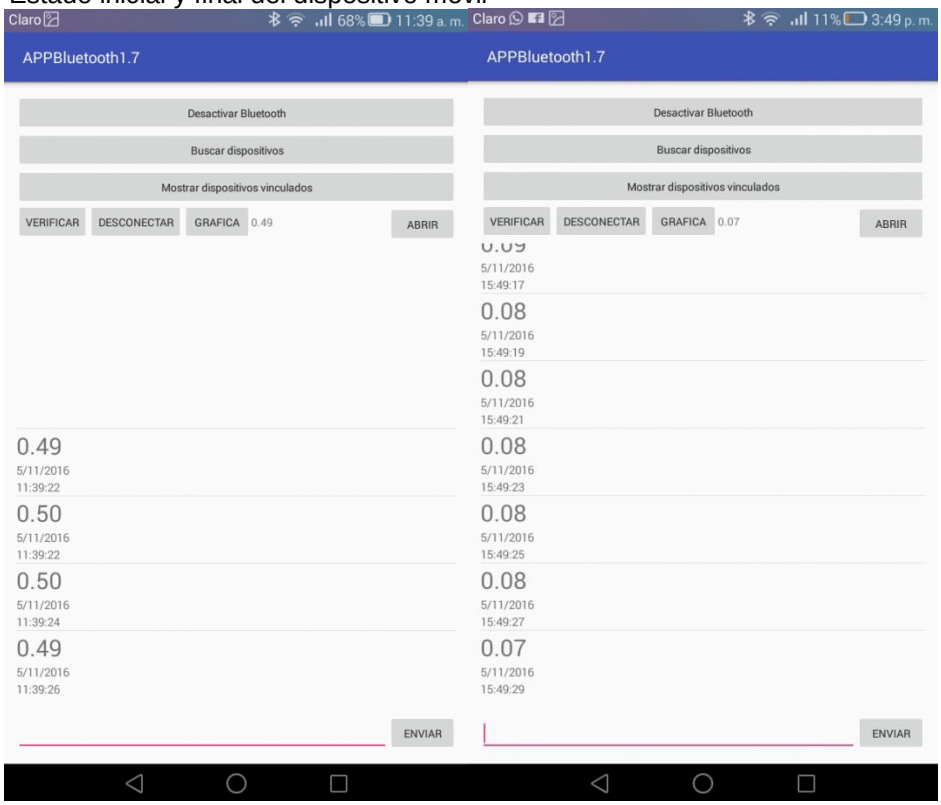


Fuente. Fuente propia

La captura de datos inició a las 11:39:22 a.m. y finalizó a las 3:49:35 p.m., es decir que fueron 4 horas y 10 minutos destinadas a la captura de datos, el sistema contaba con una pantalla lcd 16x2, que permitía visualizar el dato que el sensor GUVA S12SD registraba, esto se hizo con el fin de poder realizar una comparación rápida entre el dato capturado y el recibido por la aplicación. Se usó una tarjeta Arduino para la programación de dicho sistema.

Se tomaron capturas de pantalla para así poder analizar el consumo de batería por parte de la aplicación al encontrarse en constante funcionamiento

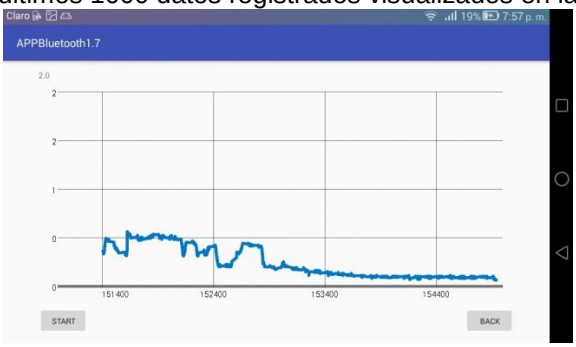
Figura 32. Estado inicial y final del dispositivo móvil



Fuente. Fuente propia

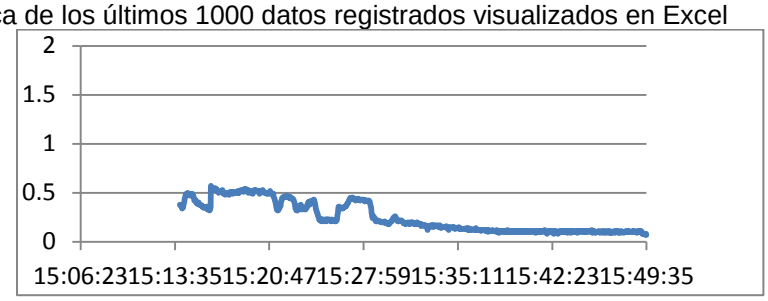
Como se puede observar en la figura 31, el dispositivo móvil perdió un 57% de carga en la batería en 4 horas y 10 minutos.
Durante esta toma de datos se registraron 7344 datos, los cuales fueron almacenados en un archivo de texto, la aplicación permita graficar máximo 1000 datos, por lo que se configuro para que en caso de obtener un numeró mayor de datos, se grafiquen los últimos 1000 datos obtenidos. A continuación se muestra la gráfica obtenida en la aplicación, y una generada en Excel, al igual que la gráfica obtenida de los todos los datos registrados.

Figura 33. Gráfica de los últimos 1000 datos registrados visualizados en la aplicación



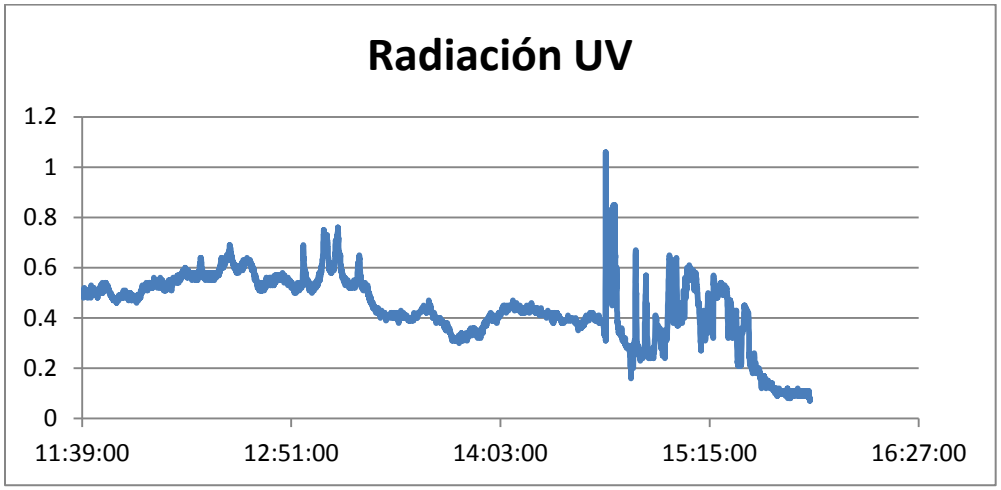
Fuente. Fuente propia

Figura 34. Gráfica de los últimos 1000 datos registrados visualizados en Excel



Fuente. Fuente propia

Figura 35. Gráfica de todos los datos obtenidos

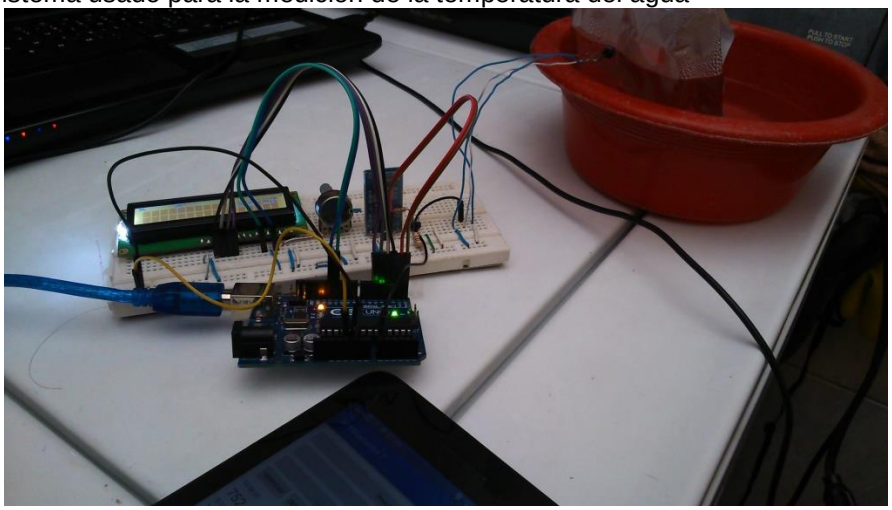


Fuente. Fuente propia

Lectura de datos de temperatura

Se realizó una segunda toma de datos, con el fin de mostrar que la aplicación es capaz de recibir más de un tipo de dato, en esta prueba se usó un sensor de temperatura lm35, el cual media la temperatura del agua contenida en una botella, como se ve en la figura 35. Se utilizó el mismo sistema de adquisición de datos, mencionado en la prueba de radiación UV.

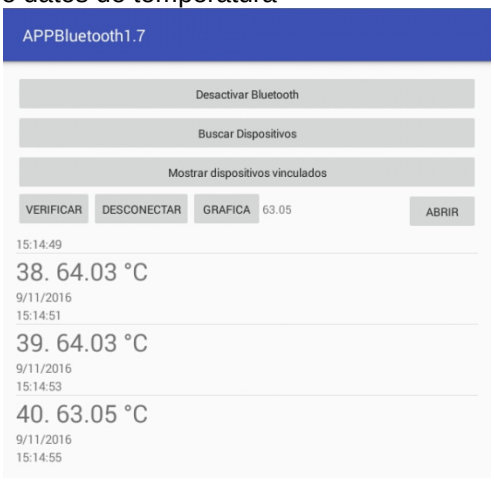
Figura 36. Sistema usado para la medición de la temperatura del agua



Fuente. Fuente propia

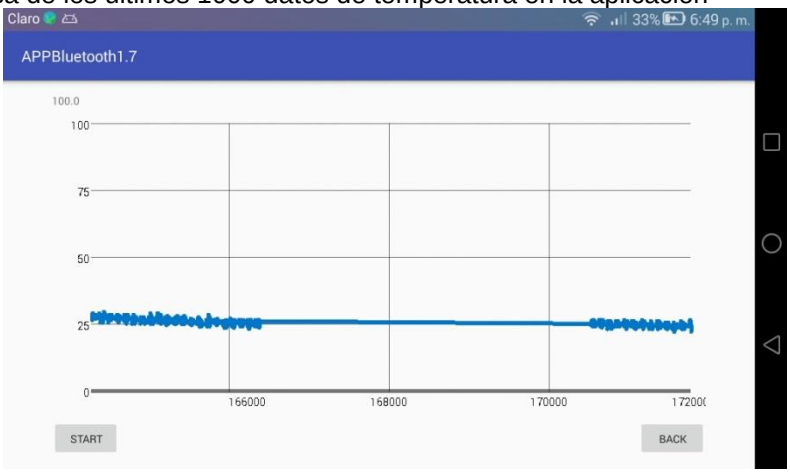
La toma de datos se realizó por 2 horas, se inició a las 3:13:50 p. m. y finalizó a las 5:12:29 p. m., se recibía un dato cada dos segundos, durante este tiempo se obtuvieron 3565 datos.

Figura 37. Inicio de la toma de datos de temperatura



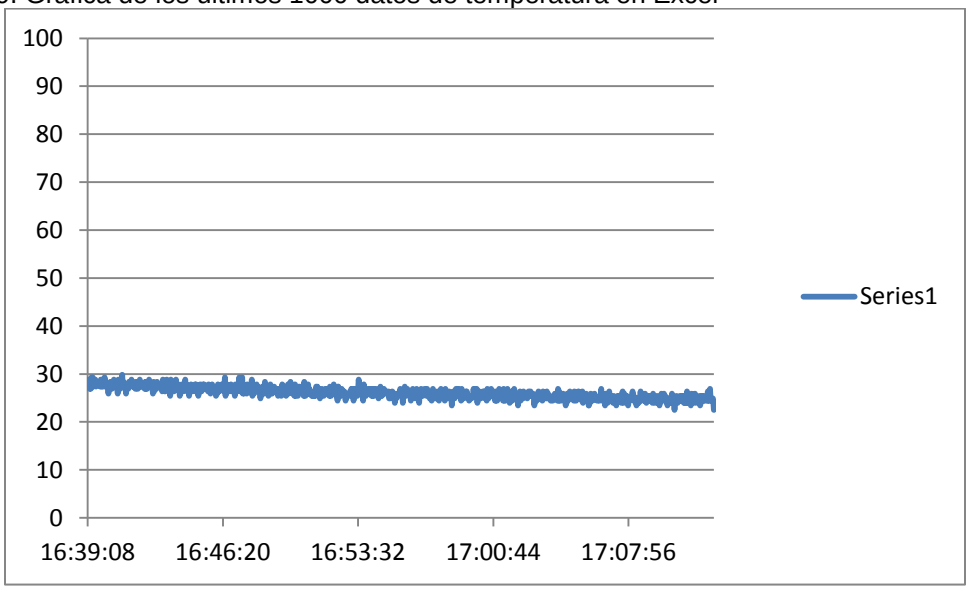
Fuente. Fuente propia

Figura 38. Gráfica de los últimos 1000 datos de temperatura en la aplicación



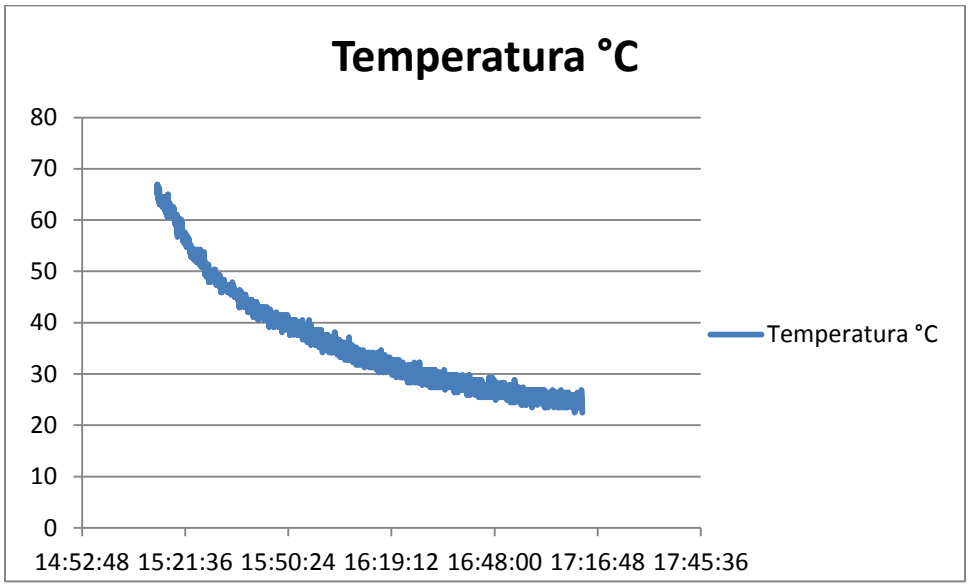
Fuente. Fuente propia

Figura 39. Gráfica de los últimos 1000 datos de temperatura en Excel



Fuente. Fuente propia

Figura 40. Gráfica de todos los datos obtenidos



Fuente. Fuente propia

CONCLUSIONES

- La visualización de datos de forma explícita exige menos recursos por parte del dispositivo para su correcto funcionamiento.

- La visualización de datos de forma gráfica es la forma más exige recursos por parte del dispositivo, para su constante uso se debe tener en cuenta la duración de la batería, y capacidad de procesamiento y visualización de los datos.

- La versión del sistema operativo con la que cuente el dispositivo en el que se use la aplicación es un factor importante para que el almacenamiento de los datos se haga de forma correcta.

- El uso de este tipo de data logger permite ahorrar costos en el proceso de adquisición y procesamiento de datos, ya que se puede contar con unidades de conexión, envío y recepción de datos, procesamiento de datos, visualización y almacenamiento en un solo dispositivo.

- El uso del registrador de datos desarrollado permite obtener los datos, visualizarlos y almacenarlos, por tanto el usuario es capaz de realizar un análisis mientras se reciben los datos, e incluso un análisis posterior con los que se almacenan, cubriendo así gran parte de la necesidades que se plantearon al inicio.

- Al almacenar los datos en un dispositivo extraíble como una memoria SD, el sistema no depende de tener conexión a internet para su correcto funcionamiento. Lo que hace al sistema útil en zonas sin cobertura de red.

TRABAJOS FUTUROS

A continuación se mencionan algunos posibles trabajos futuros que se pueden realizar a partir del desarrollo de este proyecto para ofrecer más ventajas para el usuario en determinadas áreas de investigación o también de forma general.

- Implementar una opción que permita al usuario almacenar los datos obtenidos en algún sistema de almacenamiento en la nube.

- Permitir el almacenamiento de la gráfica generada en un archivo de imagen

- Ofrecer métodos de conexión alternos a la Bluetooth.

- Implementar un sistema de conexión que permita vincularse a más de un dispositivo.

- Desarrollar un sistema que permita a la aplicación ser usada para diferentes sistemas de medida, es decir que ajuste unidades, valores máximos y mínimos en la visualización.

BIBLIOGRAFÍA

NationalInstruments. (s.f.). *National Instruments*. Recuperado el 8 de Noviembre de 2016, de National Instruments: http://www.ni.com/data_logger/esa/whatis.htm (akajjoe64), J. G. (Diciembre de 2014). *GRAPH VIEW*. Recuperado el Junio de Julio de 2016, de http://www.android-graphview.org/

Bluetooth. (s.f.). *Bluetooth.com*. Recuperado el 8 de noviembre de 2016, de https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth

Dominguez-Vega, Z.-T., Martinez-Mendez, R., & Lorias-Espinoza, D. (25-27 de Feb de 2015). *IEEE Xplore Digital Library.* Recuperado el 15 de Julio de 2016, de http://biblioteca.libertadores.edu.co:2087/xpl/articleDetails.jsp?arnumber=7086947&newsearch=true&queryText=datalogger

HunaE., M. F., Azrulhisham, A., Hamzani, K., Sulong, W. M., Abdullah, S., Basri, M. J., y otros. (27-29 de Aug de 2014). *IEEE Xplore Digital Library*. Recuperado el 15 de 07 de 2016, de http://biblioteca.libertadores.edu.co:2087/xpl/articleDetails.jsp?arnumber=7006256&newsearch=true&queryText=datalogger

J.P. Oria, L. A. (3 de Marzo de 2011). *IEEE Xplore Digital Library.* Recuperado el 19 de Julio de 2016, de http://biblioteca.libertadores.edu.co:2087/stamp/stamp.jsp?tp=&arnumber=5724135

MicrosofDeveloperNetwork. (s.f.). *Microsoft Developer Network.* Recuperado el Julio de 2016, de Microsoft Developer Network: https://msdn.microsoft.com/es-es/library/aa290752(v=vs.71).aspx

NationalInstruments. (s.f.). *National Instruments*. Recuperado el 8 de Noviembre de 2016, de National Instruments: http://www.ni.com/data_logger/esa/whatis.htm

Pietta, L. P., Xavier, P., & Michels, L. (29 de Nov de 2015). *IEEE Xplore Library.* Recuperado el 15 de Julio de 2016, de http://biblioteca.libertadores.edu.co:2087/xpl/articleDetails.jsp?arnumber=7420136&newsearch=true&queryText=datalogger

ANEXOS

ANEXO A. LINEAS DE CODIGO USADAS PARA EL DISEÑO DE LA APLICACIÓN

MainActivity.java

- Handler para recepción de datos

```
    mHandler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            if (msg.what == RECEIVING_DATA) {

                String readMessage = null;
                try {
                    readMessage = new String((byte[]) msg.obj, "UTF-8");
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
                deviceName.setText(readMessage.replace("\n", ""));
                mAMD =
java.text.DateFormat.getDateInstance().format(Calendar.getInstance().getTime());
                mHMS =
java.text.DateFormat.getTimeInstance().format(Calendar.getInstance().getTime());
                mFecha = (String.valueOf(mCalendar.get(Calendar.DAY_OF_MONTH))) +
(String.valueOf(mCalendar.get(Calendar.MONTH) + 1) +
(String.valueOf(mCalendar.get(Calendar.YEAR))));

                nuevoDato(readMessage.replace("\n", ""), mHMS, mAMD);

            } else if (msg.what == CONNECTION_LOST) {
                if (STATE_CONNECTION == DEVICE_CONNECTED) {
                    lost();
                }
            }
        }
    };
}
```

- Método para el manejo de desconexión inesperada

```
public void lost() {

      Log.e("AA", "se desconectó el dispositivo");
      Toast.makeText(getApplicationContext(), "se perdió la conexión",
Toast.LENGTH_SHORT).show();
      setSTATE_CONNECTION(DEVICE_DISCONNECTED);
      reconnect = new dialogRec();
      reconnect.execute();
   }
```

- Activar y desactivar conexión bluetooth

```
   public void OnOff(View view) {
      if (STATE_BLUETOOTH == BLUETOOTH_OFF) {
         new AlertDialog.Builder(this)
               .setTitle("Activar Bluetooth")
               .setMessage("¿Desea activar la conexión Bluetooth?")
               .setPositiveButton("Si", new DialogInterface.OnClickListener() {
                  @Override
                  public void onClick(DialogInterface dialog, int which) {
                     BA.enable();
                     setSTATE_BLUETOOTH(BLUETOOTH_ON);
                     activarDesactivar.setText("Desactivar Bluetooth");
                     dialog.dismiss();
                  }
               })
               .setNegativeButton("No", new DialogInterface.OnClickListener() {
                  @Override
                  public void onClick(DialogInterface dialog, int which) {
                     dialog.dismiss();
                  }
               })
               .setIcon(android.R.drawable.ic_dialog_alert)
               .show();
      } else if (STATE_BLUETOOTH == BLUETOOTH_ON) {
         new AlertDialog.Builder(this)
               .setTitle("Desactivar Bluetooth")
               .setMessage("¿Desea desactivar la conexión Bluetooth?")
               .setPositiveButton("Si", new DialogInterface.OnClickListener() {
                  @Override
                  public void onClick(DialogInterface dialog, int which) {
                     BA.disable();
                     setSTATE_BLUETOOTH(BLUETOOTH_OFF);
                     activarDesactivar.setText("Activar Bluetooth");
                     dialog.dismiss();
                  }
```

```
                })
                .setNegativeButton("No", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                    }
                })
                .setIcon(android.R.drawable.ic_dialog_alert)
                .show();
        }
    }
```

- Búsqueda de dispositivos Cercanos y vinculados

```
    public void discover(View view) {

        if (BA.isDiscovering()) {
            BA.cancelDiscovery();
            buscarDispositivos.setText("Buscar dispositivos");
            Toast.makeText(getApplicationContext(), "Se ha detenido la busqueda",
Toast.LENGTH_SHORT).show();
        } else if (STATE_BLUETOOTH == BLUETOOTH_ON) {
            listDevices.clear();
            if (arrayAdapterDatos != null) arrayAdapterDatos.clear();
            ArrayAdapterDevices.clear();
            BA.startDiscovery();
            progress.show();
            buscarDispositivos.setText("Cancelar búsqueda");

            IntentFilter filter = new IntentFilter();
            filter.addAction(BluetoothDevice.ACTION_FOUND);

            filter.addAction(BA.ACTION_DISCOVERY_FINISHED);
            registerReceiver(blReceiver, filter);

        } else
            Toast.makeText(getApplicationContext(), "La conexión Bluetooth no está activada",
Toast.LENGTH_SHORT).show();
    }

    final BroadcastReceiver blReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            String action = intent.getAction();
            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                listDevices.add(device);
                ArrayAdapterDevices.add(device.getName() + "\n" + device.getAddress());
                ArrayAdapterDevices.notifyDataSetChanged();
            } else if (BA.ACTION_DISCOVERY_FINISHED.equals(action)) {
                progress.dismiss();
                buscarDispositivos.setText("Buscar dispositivos");
                if (ArrayAdapterDevices.isEmpty())
```

```
                Toast.makeText(getApplicationContext(), "No se encontraron dispositivos",
Toast.LENGTH_SHORT).show();
            }
        }
    };

    public void mostrarPaired(View view) {

        BDdevice = BA.getBondedDevices();
        if (STATE_BLUETOOTH == BLUETOOTH_ON) {
            ArrayAdapterDevices.clear();
            listDevices.clear();
            if (arrayAdapterDatos != null) arrayAdapterDatos.clear();
            for (BluetoothDevice device : BDdevice) {
                listDevices.add(device);
                ArrayAdapterDevices.add(device.getName() + "\n" + device.getAddress());
            }
            Toast.makeText(getApplicationContext(), "Mostrando dispositivos",
Toast.LENGTH_SHORT).show();
        } else
            Toast.makeText(getApplicationContext(), " La conexión Bluetooth no está activada",
Toast.LENGTH_SHORT).show();
    }

    private AdapterView.OnItemClickListener lvListener = new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            if (STATE_CONNECTION == DEVICE_DISCONNECTED || STATE_CONNECTION ==
DEVICE_DISCONNECTED_MANUAL) {
                pos = position;
                AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
                builder.setTitle("Conectar")
                        .setMessage("Desea conectarse con este dispositivo")
                        .setPositiveButton("Si", new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
                                dialog.dismiss();
                                addMAC = listDevices.get(pos).getAddress();
                                nameDevice = listDevices.get(pos).getName();
                                connectBluetooth(addMAC);
                            }
                        })
                        .setNegativeButton("No", new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
                                dialog.dismiss();
                            }
                        })
                        .setIcon(android.R.drawable.ic_dialog_alert);
                AlertDialog dialog = builder.create();
                dialog.show();
            } else
                Toast.makeText(getApplicationContext(), "esta conectado a" + " " + nameDevice,
Toast.LENGTH_SHORT).show();
        }
    };
```

- Protocolo ventana deslizante

```
public void slideproto(String bff1[]) throws Exception {
        Socket socketsld = new Socket(mmSocket.getRemoteDevice());
        DataInputStream tmpInsld = mmSocket.getInputStream();
        int pos = 0;
        int rdata = -1;
        int nfsld;
        int rwsdata = 8;
        String databufsld[] = new String[8];
        String datas;
        Log.e("dato recibido", "dato recibido");
        nfsld = Integer.parseInt(tmpInsld.readLine());
        if (nfsld <= rwsdata - 1) {
           for (pos = 1; pos <= nfsld; pos++) {
              rdata = ++rdata % 8;
              databufsld[rdata] = tmpInsld.readLine();
              Log.e("dato recibido", "se recibió " + rdata + "databufsld[rdata]");
           }
           rwsdata -= nfsld;
           Log.e("ACK enviado", "ACK enviado");

        }
           datas = tmpInsld.readLine();

        while(datas.equals("yes"));

  }
```

- Desconexión de un dispositivo

```
  public void desconectar_Device() {

     if (STATE_BLUETOOTH == BLUETOOTH_ON) {
        if (STATE_CONNECTION == DEVICE_DISCONNECTED)
           Toast.makeText(getApplicationContext(), "no esta conectado a ningún dispositivo",
Toast.LENGTH_SHORT).show();
        else if (STATE_CONNECTION == DEVICE_CONNECTED) {
           try {

              Log.e("closeBff", "close() of connect " + mRxMain + " Thread closed");

              if (mmInStream != null) {
                 mmInStream.close();
                 mmInStream = null;
                 Log.e("closeBff", "close() of connect " + mmInStream + " bffIn closed");
              }
              if (mmOutStream != null) {
                 mmOutStream.close();
                 mmOutStream = null;
                 Log.e("closeBff", "close() of connect " + mmOutStream + " bffOut closed");
              }
              Toast.makeText(getApplicationContext(), "Se deconectó de" + " " + nameDevice,
Toast.LENGTH_SHORT).show();
```

```
                mRxMain.cancel();
                mConnectMain.cancel();
                Log.e("closeBff", "scktclosed");

                try {
                    mConnectMain.join();
                } catch (InterruptedException e) {
                    Log.e("AA", "no ha terminado el thread");
                    e.printStackTrace();
                }
                mRxMain.interrupt();
                mRxMain = null;
                mConnectMain.interrupt();
                mConnectMain = null;

                deviceName.setText("Device");
                setSTATE_CONNECTION(DEVICE_DISCONNECTED_MANUAL);
                ArrayAdapterDevices.clear();
                guardarDatos(mFecha, arrayListDatos);
                if (arrayAdapterDatos != null) arrayAdapterDatos.clear();
                lvDevices.setAdapter(ArrayAdapterDevices);
            } catch (IOException e) {

            }
        } else
            Toast.makeText(getApplicationContext(), "no esta conectado a ningún dispositivo",
Toast.LENGTH_SHORT).show();
    } else
        Toast.makeText(getApplicationContext(), "La conexión Bluetooth no está activada",
Toast.LENGTH_SHORT).show();
  }
```

- Acceder a la función de graficado

```
  public void graphA(View view) {

    if (STATE_BLUETOOTH == BLUETOOTH_ON) {
        desconectar_Device();
        Intent iGraph = new Intent(this, Graph_Activity.class);
        iGraph.putExtra("MAC", addMAC);
        startActivity(iGraph);

    } else if (STATE_CONNECTION == DEVICE_DISCONNECTED)
        Toast.makeText(getApplicationContext(), "Debe conectarse a un dispositivo primero",
Toast.LENGTH_SHORT).show();
  }
  public void nuevoDato(String valor, String dia, String hora) {
    Dato nuevoDato = new Dato(valor, dia, hora);
    arrayAdapterDatos.add(nuevoDato);
    arrayAdapterDatos.notifyDataSetChanged();

  }
```

- Almacenamiento de datos obtenidos

```
   public void guardarDatos(String nArchivo, ArrayList<Dato> listaDatos) {
      String name = "Datos" + nArchivo + ".txt";

      try {
         File root = Environment.getExternalStorageDirectory();
         File myFile = new File(root, String.valueOf(name));

         FileOutputStream fOut = new FileOutputStream(myFile);
         OutputStreamWriter myOutWriter =
               new OutputStreamWriter(fOut);
         Log.e("ERROR", "file" + myOutWriter.getClass().getName());
         for (Dato dt : listaDatos) {
            myOutWriter.write(((dt.getmHora() + " " + dt.getmValor()) + "\n"));
            Log.e("ERROR", "root" + dt.getmValor());
         }
         myOutWriter.close();
         fOut.close();
      } catch (Exception e) {
         Log.e("ERROR", "fa53ed");
         Toast.makeText(getApplicationContext(), "f creacion del archivo",
Toast.LENGTH_SHORT).show();
         e.printStackTrace();
      }
      arrayListDatos.clear();
   }
```

- Abrir archivo guardado

```
   public void abrirArchivo(View view) {
      desconectar_Device();
      LayoutInflater layoutInflaterLeer = LayoutInflater.from(MainActivity.this);
      View dataView = layoutInflaterLeer.inflate(R.layout.txtdialog, null);
      AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(MainActivity.this);
      alertDialogBuilder.setView(dataView);

      final EditText editTextA = (EditText) dataView.findViewById(R.id.edittext_año);
      final EditText editTextM = (EditText) dataView.findViewById(R.id.edittext_mes);
      final EditText editTextD = (EditText) dataView.findViewById(R.id.edittext_dia);

      alertDialogBuilder.setCancelable(false)
            .setNeutralButton("Ver Grafica", new DialogInterface.OnClickListener() {
               @Override
               public void onClick(DialogInterface dialog, int which) {
                  nameFile = editTextD.getText().toString() + editTextM.getText().toString() +
editTextA.getText().toString();
                  leerArchivo(verificaRutaSd(), nameFile, false);
                  dialog.dismiss();
               }
            })
            .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
               @Override
               public void onClick(DialogInterface dialog, int which) {
                  nameFile = editTextD.getText().toString() + editTextM.getText().toString() +
editTextA.getText().toString();
                  Toast.makeText(MainActivity.this, nameFile, Toast.LENGTH_SHORT).show();
```

```
                leerArchivo(verificaRutaSd(), nameFile, true);
                dialog.dismiss();
            }
        })
        .setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.cancel();
            }
        });
    AlertDialog txtAlert = alertDialogBuilder.create();
    txtAlert.show();
}

public void leerArchivo(String ruta, String nombre, boolean mostrar) {

    ArrayList<String> listaDatosArchivo = new ArrayList<String>();
    File root = Environment.getExternalStorageDirectory();
    File file = new File(root, String.valueOf("Datos" + nombre + ".txt"));
    boolean existe = false;

    if (file.exists()) {
        existe = true;
        Toast.makeText(MainActivity.this, "se encontró el archivo" + root + "/" + nombre,
Toast.LENGTH_SHORT).show();
    } else Toast.makeText(MainActivity.this, "no existe el archivo",
Toast.LENGTH_SHORT).show();
    if (mostrar == true) {
        StringBuilder text = new StringBuilder();
        try {
            BufferedReader br = new BufferedReader(new FileReader(file));
            String line;
            while ((line = br.readLine()) != null) {
                text.append(line);
                text.append('\n');
            }
            br.close();
            ArrayAdapterDevices.add(text.toString());
        } catch (IOException e) {
            Toast.makeText(MainActivity.this, "no se pudo abrir el archivo",
Toast.LENGTH_SHORT).show();
        }
    } else if (mostrar == false && existe == true) {
        Intent iGraph = new Intent(this, Graph_Activity.class);
        iGraph.putExtra("nombreArchivo", nombre);
        startActivity(iGraph);
    }
}
```