UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Aplicación móvil en Android para registro de telemetría vía bluetooth de un watthorímetro digital

TESIS

Que para obtener el título de

Ingeniera en Computación

P R E S E N T A

María Alejandra Castillo Martínez

DIRECTOR DE TESIS

M. I. Juan Ricardo Damián Zamacona

Ciudad Universitaria, Cd. Mx., 2024

A MIS PADRES,
QUE ME APOYARON CON AMOR Y
CUIDADO A TRAVÉS DE TODA MI
TRAYECTORIA ACADÉMICA.

AGRADECIMIENTOS

Quiero agradecer a los miembros del laboratorio de electrónica del ICAT, por toda la guía que me han dado. A todos los profesores con los que he tenido el placer de tomar clases. A mis amigos por su apoyo incondicional, especialmente a Mar, por escuchar mis divagaciones de programadora. A mi familia, que me vio crecer. A mi hermanito por cuidarme y preocuparse por mí. Y, sobre todo, a mis padres, por haberme criado con amor para ser la persona que soy ahora.

CONTENIDO

1 INTRODUCCIÓN

En el Grupo de Electrónica del Instituto de Ciencias Aplicadas y Tecnología, se tiene interés en la electrónica de potencia. En este ámbito, se puede mencionar algunos proyectos como: el diseño y construcción del control electrónico y etapa de potencia para un vehículo eléctrico experimental (Castillo, et al., 2020), la participación en un proyecto para el ahorro de energía en edificios (Castillo, et al., 2013) y el desarrollo de la electrónica para el control de motores de inducción (Castillo, et al., 2007), entre otros, ver Figuras 1, 2 y 3. A partir de esto surgió la necesidad de medir variables como la potencia promedio y la energía. Por esta razón, se diseñó e implementó un watthorímetro que se probó en 2019 en un vehículo eléctrico (Castillo, et al., 2019). En este último, en lo particular, el diseño propuesto almacenaba la información en memoria para después descargarla y procesarla. Es importante resaltar que, aunque contaba con las terminales para conectar un módulo de comunicación inalámbrica, este no se había usado y esto ofreció una oportunidad para la creación de una aplicación móvil.

Figura 1. Prototipo para el control electrónico de un motor BLDC usado en un vehículo eléctrico.

Figura 2. Medición inalámbrica del consumo eléctrico para el ahorro de energía en edificios.

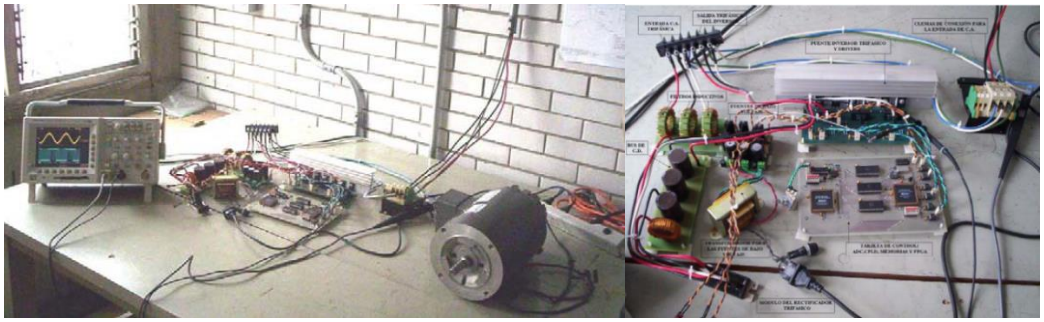
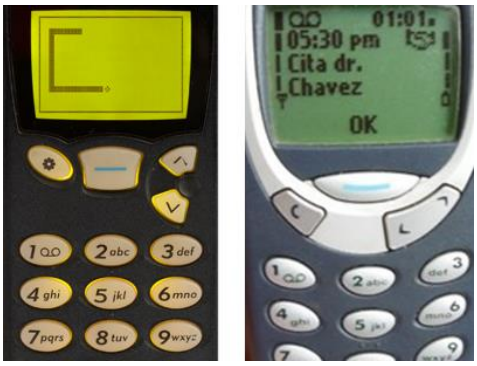


Figura 3. Implementación de un algoritmo SV-PWM en un FPGA para control de motores de inducción.

Una aplicación móvil es un programa de software que es posible instalar en dispositivos que cuenten con un sistema operativo propio. Un sistema operativo es el software base de cualquier dispositivo, que actúa de intermediario entre el usuario y el hardware (Silberschatz, et al., 2004). Dado que el sistema operativo es el único programa que se comunica directamente con el hardware, sirve a su vez como un intermediario entre este y el resto de los programas. El sistema operativo frece una serie de abstracciones que permite a los programas enfocarse en la resolución de sus problemas específicos y no en la complejidad de interactuar directamente con el hardware. Un ejemplo de esto son los sistemas de archivos, los cuales permiten encontrar la información solicitada en poco tiempo sin la necesidad de sumergirse en el intrincado mundo de la memoria secundaria. Con este ejemplo, se encuentra de frente con otra tarea de vital importancia del

sistema operativo: la administración de recursos. Un dispositivo tiene acceso a cierta cantidad de recursos (entiéndase por esto, la memoria RAM, el espacio de almacenamiento, el procesamiento, etc.). Por ende, el sistema operativo tiene la responsabilidad de gestionar estos recursos ante los diversos procesos que se efectúan en el dispositivo y asignarlos de forma eficiente acorde a las necesidades del sistema. En caso de tratarse de un sistema multiusuario y/o multitarea, el sistema operativo también debe encargarse del aislamiento de los respectivos usuarios y procesos, para que estos no tengan que preocuparse de las otras entidades conviviendo en el mismo entorno, y así ofrecer la misma experiencia que se tendría si el sistema estuviera dedicado únicamente a ese proceso o usuario en específico (Wolf , et al., 2015).

Los sistemas operativos se diseñan según las necesidades del equipo en el que serán instalados. En el caso de los dispositivos móviles, sus sistemas operativos deben proporcionar un entorno en el que el usuario pueda interactuar fácilmente con el dispositivo y ejecutar programas de la forma más eficiente posible. En sus inicios, en los años 90, las aplicaciones eran programas sumamente sencillos, tales como la agenda de contactos en el celular y las alarmas. También existían algunos videojuegos simples, como Snake, ver Figura 4. Sin embargo, con la llegada de sistemas operativos como iOS y Android, el mercado de las aplicaciones móviles creció a pasos agigantados.



a) b)

Figura 4. a) El juego para celular Snake y b) la aplicación de alarma.

Actualmente existen dos sistemas operativos que dominan el mercado de teléfonos celulares a nivel mundial. Estos son: Android, de Google, y iOS, de Apple. iOS surgió por primera vez en 2007, entonces llamado “iPhone OS”. Su mayor característica fue la tecnología multitouch, la cual es, actualmente, una función casi indispensable de los teléfonos móviles. Android, por otra parte, surgió en 2003 como un sistema operativo para cámaras fotográficas y era desarrollado por una empresa independiente del mismo nombre. Fue en 2004 cuando cambiaron su enfoque a teléfonos móviles y en 2005 cuando se integró a Google. Hay múltiples razones por las que fueron justo estos dos sistemas operativos los que sobrevivieron hasta la fecha, pero la más importante es su gran variedad de aplicaciones.

Existen otros sistemas operativos móviles aún en el mercado, Tizen, por ejemplo, es un sistema operativo de código abierto el cual es un proyecto de la Fundación Linux. Hasta el momento, Tizen puede ser empleado en sistemas de entretenimiento de los automóviles, en Smart TV, celulares y Smart Watch (Linux Foundation, 2012).

Otro ejemplo es KaiOS, el cual está pensado para móviles de tapa y con teclas. Este sistema permite vivir la experiencia de un celular moderno en dispositivos que carezcan de una pantalla táctil, es empleado para llevar la experiencia del internet móvil a gente que no puede acceder a teléfonos celulares de una mayor gama (Kai OS Technologies, 2021).

Respecto a sistemas operativos en desuso, existen ejemplos como Windows Phone, el cual a pesar de ser un sistema operativo rápido y optimizado, fracasó al momento de ofrecer la variedad de aplicaciones clave para triunfar en este nicho. Otro es BlackBerry OS, el cual fue uno de los pioneros en el mundo de los Smartphone y era muy querido por la comunidad; lamentablemente este sistema operativo llegó a su fin en 2013 al no dar el siguiente paso a la tecnología multitouch. Symbian es otro ejemplo, en su tiempo fue un sistema popular que

dominaba el mercado. Sin embargo, crear aplicaciones para esta plataforma era sumamente complicado, esto lo llevó a ser descontinuado en 2012.

Los dispositivos inteligentes son aparatos electrónicos con un sistema operativo integrado, tales como teléfonos celulares, televisores y relojes, ver Figura 5. En la actualidad, hay una gran variedad de estos dispositivos y es posible crear aplicaciones que se adapten a cada uno de ellos; pero para fines de este trabajo se centrará únicamente en aplicaciones para teléfonos celulares.

Figura 5. Dispositivos inteligentes: Smart TV, smartphone, tableta y smartwatch.

La evolución de las aplicaciones móviles ha sido impresionante, tan solo en el aspecto visual el cambio ya es bastante notorio, ver Figura 6, pero el cambio va más allá de eso. A partir de la incorporación del internet a los teléfonos inteligentes, estos tomaron un papel importante en la vida cotidiana y, por ende, las aplicaciones en ellos también. La gente suele pasar más tiempo usando sus teléfonos que cualquier otro dispositivo. Cada red social tiene su propia aplicación, cada página web, su propia versión móvil. Las aplicaciones actuales ofrecen imágenes nítidas y coloridas, así como un periodo de respuesta notoriamente bajo. En ellas se pueden ver videos e imágenes en alta definición, escuchar música de excelente calidad, tomar fotos, grabar videos, audios y muchas otras cosas.

a) b)

Figura 6. Comparación entre la aplicación de mensajería a) en un teléfono antiguo y b) en uno moderno.

Los teléfonos celulares actuales ya vienen con una serie de aplicaciones integradas, las cuales son, en su mayoría, herramientas básicas como la cámara fotográfica, el servicio de mensajes de texto, la aplicación para llamadas telefónicas, las alarmas, el calendario, etc. Por otro lado, en la actualidad el mundo de las aplicaciones es tan amplio que un solo dispositivo no puede contenerlas todas. Las aplicaciones de la actualidad son sumamente variadas y cada usuario debe buscar las que mejor se adapten a sus necesidades. Para esto, existen las tiendas de aplicaciones desde las cuales se tiene acceso a un amplio catálogo de aplicaciones que pueden ser de paga o gratuitas. Las dos tiendas de aplicaciones más grandes y conocidas que existen son: Google Play, de Google, desde la que se pueden descargar aplicaciones para el sistema operativo Android, ver Figura 7; y App Store, de Apple, desde donde se pueden descargar aplicaciones para el sistema operativo iOS.

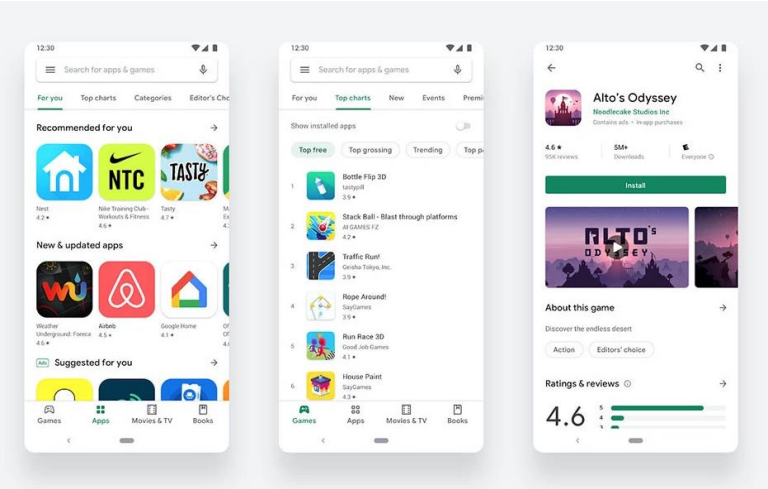


Figura 7. Capturas de pantalla de Google Play.

En estas tiendas de aplicaciones se pueden encontrar aplicaciones que cubren una gran variedad de necesidades. Las más populares son las redes sociales, los videojuegos y los servicios de streaming, transporte y comida a domicilio. Por otro lado, muchos tipos de empresa tienen sus propias aplicaciones aun cuando existan locales físicos a los cuales acudir, los mejores ejemplos de esto son los bancos, algunos restaurantes y tiendas de todo tipo, en especial si tienen servicio de ventas en línea. Sin embargo, el mundo de las aplicaciones móviles no se detiene solo en esto, las aplicaciones existen para cubrir necesidades, por lo que el público al que están dirigidas puede ser tan general como específico. Un artista puede encontrar aplicaciones de diseño para ayudarlo con su trabajo. Un deportista puede encontrar una serie de aplicaciones que lo ayuden a mejorar su desempeño, como cronómetros especializados y guías de alimentación. Si se quiere aprender un nuevo idioma o estudiar para algún examen también hay aplicaciones que ayudan con eso. Incluso es común encontrar sistemas de monitoreo como aplicaciones enlazadas a cámaras de vigilancia, o al sistema de calefacción de una casa.

Tomando en cuenta esto y ante la clara necesidad de contar con un medio para poder visualizar los datos obtenidos del watthorímetro en tiempo real, se propuso el diseño de una aplicación móvil que permite enlazarse al watthorímetro vía

bluetooth, para adquirir y almacenar datos en tiempo real. Con este propósito, es necesario ahondar más en los tipos de aplicaciones, los componentes que las conforman, las características de diseño que debe presentar una buena aplicación móvil y la forma en que se relacionan con el sistema operativo.

El rasgo distintivo de las aplicaciones de hoy en día es la facilidad con la que se pueden usar. No suele ser necesario emplear un manual, al menos no para usar los elementos más básicos en ellas. El usuario tiene que poder acceder a los espacios que busca de forma intuitiva; no espera pensar demasiado cuando emplea una aplicación, quiere que todo sea sencillo de comprender y lo guie amablemente a cada paso. Como se puede apreciar en la Figura 8, los elementos con los que se puede interactuar indican de una forma sencilla cuál es su objetivo, siendo, por ejemplo, el botón azul el que lleve a la actividad azul y el botón rojo el que introduzca a la actividad roja.

Figura 8. Ejemplo grafico de navegación intuitiva en una aplicación.

Existen tres tipos de aplicaciones móviles. El tipo más básico son las aplicaciones web. Estas son versiones para celular de páginas web normales y están desarrolladas con los mismos lenguajes de programación que se emplean para desarrollar las versiones de escritorio. El siguiente tipo son las aplicaciones nativas. Estas son aplicaciones que viven en el dispositivo y son diseñadas para

un único sistema operativo; se desarrollan en Java o Kotlin, en el caso de Android, y en Swift en el caso de iOS. Las últimas, son las aplicaciones híbridas, las cuales también viven en el dispositivo, pero se desarrollan independientemente del sistema operativo en el que se planea usarlas. Estas últimas se desarrollan empleando herramientas como Flutter y React Native. Por motivos que se explicarán más adelante, para este trabajo, se decidió desarrollar una aplicación nativa en Android.

El sistema operativo Android es un sistema Linux multiusuario, en donde cada aplicación es un usuario. El sistema otorga a cada aplicación (usuario) un identificador (ID) al que asigna permisos específicos, de tal forma que sea la única capaz de acceder a sus propios recursos. Así mismo, cada aplicación tiene su propia máquina virtual para que se ejecute de forma independiente a las otras. Cada que se ejecuta alguno de sus componentes, se inicia un proceso independiente, el cual se cierra tan pronto como deja de usarse (Google, 2020). Además, todos los procesos que manejen acciones o información sensible para el usuario requieren de permisos puntuales. Algunos de estos permisos se otorgan al momento en que se instala la aplicación, pero los más delicados deben de otorgarse de forma expresa por el usuario en el momento en que sean empleados. Estos se eliminan luego de algunos meses, por lo que es necesario renovarlos cada tanto (Google, 2023). Estas características, hacen de Android un sistema sumamente seguro.

Los dispositivos Android están pensados para una gran variedad de público, esto incluye desde la clase trabajadora, hasta las clases más privilegiadas. Este enfoque ha desembocado en que exista lo que se conoce como “gamas” de celulares. Los dispositivos de gama baja están pensados para ofrecer las comodidades de un smartphone a un precio reducido, por lo tanto, son construidos con materiales más económicos y, en consecuencia, su potencia y almacenamiento se ven mermados. Por el contrario, las gamas altas tienen como objetivo ofrecer la más alta calidad en todos los ámbitos, se construyen con los

mejores materiales y tiene procesadores que pueden rivalizar con el de muchas computadoras, todo esto a cambio de un precio mucho más elevado.

Las aplicaciones en Android pueden contener 4 tipos de componentes: actividades, servicios, receptores de emisiones y proveedores de contenido (Google, 2020). En esta aplicación se emplearon en 3 de ellos: las actividades, los receptores de emisiones y los servicios.

Una actividad representa una pantalla individual con una interfaz de usuario desde la cual se puede interactuar con la aplicación. Las actividades tienen un ciclo de vida delicado y voluble, pues no depende enteramente de las decisiones del usuario o de la aplicación misma, sino que está relacionado con los recursos del teléfono celular en que están instaladas. Un teléfono celular de gama baja cuenta con una memoria RAM y una capacidad de procesamiento considerablemente menor a los de una computadora. Por esto mismo, el sistema operativo otorga diversos grados de prioridad a cada tarea en curso. En caso de necesitar más memoria o procesamiento para los procesos de mayor jerarquía, suspende tareas de menor relevancia y destruye sus respectivas actividades (Google, 2023). Al hacer esto, es posible que el estado de la actividad se pierda. Por ejemplo, si la actividad tiene alguna entrada de texto, el usuario podría escribir algo, salir de la actividad para atender otras tareas o recibir una llamada, y cuando regrese, lo que había escrito ya no estará allí. Por esta razón, es necesario crear la aplicación de forma que se mantenga la integridad de los datos empleados en cada parte del proceso, independientemente del tipo de aplicación. Configurar la aplicación de forma que se respeten estos parámetros, creará un ambiente más cómodo para el usuario, pero a su vez, supone un mayor reto para el programador.

Las actividades tienen un ciclo de vida y comprenderlo ayudará a diseñar las aplicaciones de la mejor manera. En la Figura 9, se puede apreciar un diagrama que ilustra el funcionamiento del ciclo de vida de una actividad. Esta cuenta con 6 estados a los que se tiene acceso a través de 6 devoluciones de llamadas de la clase Activity, los cuales se describen a continuación:

- onCreate() corresponde al estado *Created*. Este estado es aquel con el que se crea la aplicación. Se emplea su devolución de llamada para inicializar variables y procesos que estarán presentes a lo largo de toda la vida de la actividad.
- onStart() corresponde al estado S*tarted*. En este estado la actividad es visible para el usuario, pero aún no está lista para interactuar con ella. Se accede a este estado al momento de iniciar la actividad y cuando se regresa a ella después de que no haya sido visible. Por esto mismo, es en su devolución de llamada cuando se inicializan procesos que se requiere que sean visibles desde el modo multiventana, ya que este modo es aquel a través del cual se suele acceder a la aplicación después de haber perdido su visibilidad.
- onResume() corresponde al estado *Resumed*. Este es el estado más duradero del ciclo de vida, pues es aquel en el que el usuario finalmente puede interactuar con la actividad. Este estado estará presente mientras la actividad mantenga el foco y en su devolución de llamada se inicializan los procesos que serán empleados solamente mientras la actividad mantiene el foco de atención.
- onPause corresponde al estado *Paused*. Este estado es aquel en el que la actividad ha perdido el foco, pero en donde aún es parcialmente visible. Algunos ejemplos de los momentos en que se está en este estado son cuando se recibe una llamada, suena una alarma o cuando se entra en el modo multiventana. En su devolución de llamada se liberan aquellos procesos y elementos que no serán empleados ahora que se ha perdido la capacidad de interactuar con la actividad, esos mismos procesos se volverán a inicializar en el método onResume() cuando se recupere el foco.
- onStop() corresponde al estado *Stopped*. Este estado es aquel que ocurre cuando la actividad sigue existiendo en memoria, pero se pierde por completo su visibilidad en la pantalla, por ejemplo, cuando se regresa a la pantalla de inicio, se abre otra actividad o se apaga la pantalla del celular.

En este estado se liberan todos los procesos que requieran la visibilidad total o parcial de la actividad, mismos procesos que se vuelven a inicializar en onStart() cuando se recupere la visibilidad de la actividad.

- onDestroy() corresponde al estado *Destroyed*. Este es el último estado en el ciclo de vida de la actividad y puede ser llamado cuando se va a descartar por completo la actividad o cuando se va a finalizar temporalmente para implementar alguna configuración (como cuando se rota la pantalla o se entra al modo multiventana). En esta devolución de llamada se destruyen los procesos que no serán necesarios una vez que la actividad deje de existir en memoria. Además, los datos que deben ser retenidos, se almacenan en este momento (Google, 2020).

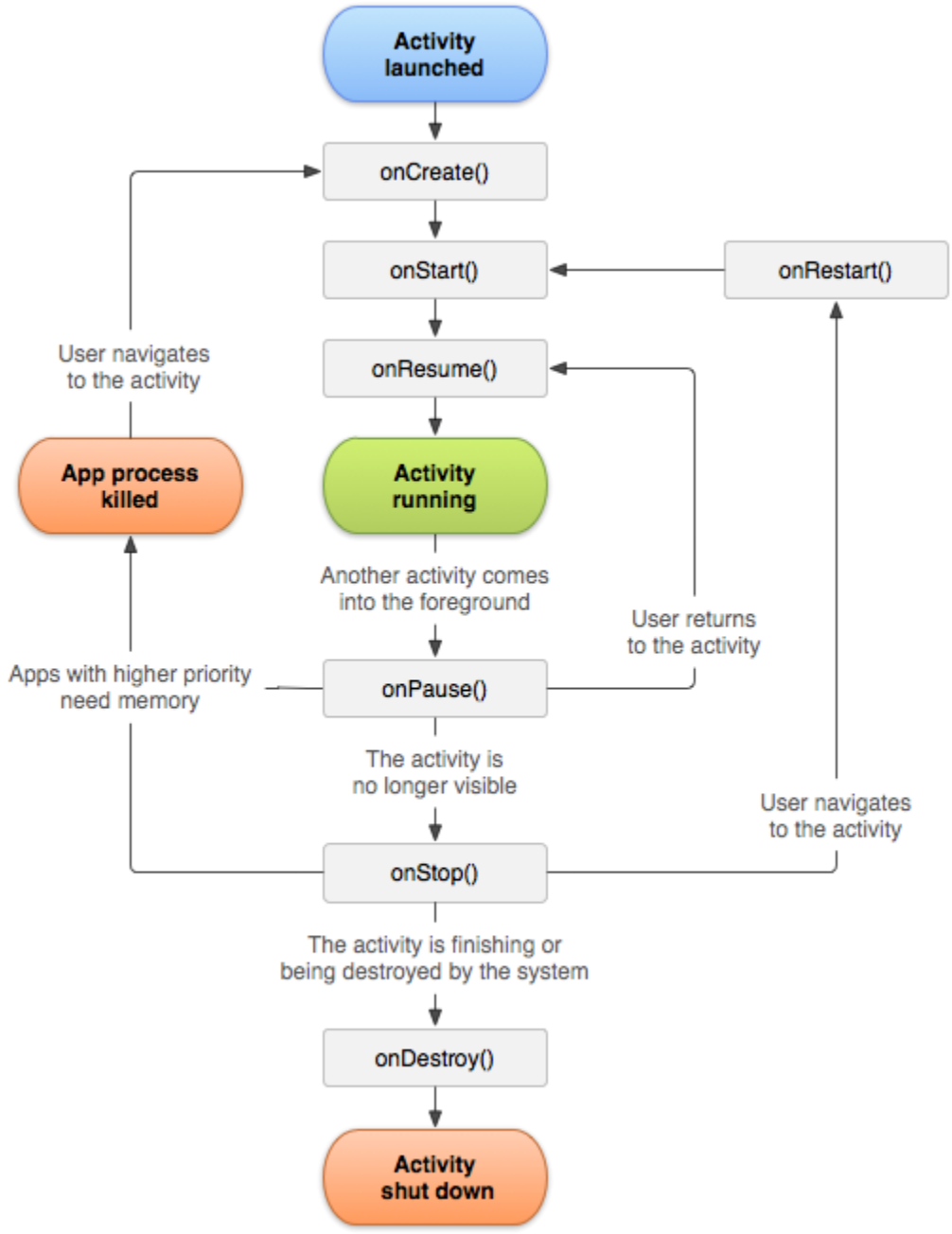


Figura 9. Diagrama de flujo del ciclo de vida de una actividad

Los receptores de emisiones son componentes que permiten detectar emisiones y eventos fuera del flujo habitual de usuarios. Gracias a estos componentes se puede recibir información de diversas emisiones del sistema de forma que se puede responder oportunamente a ellas. Los receptores de emisión pueden ser un

punto de entrada a la aplicación, esto permite que el sistema entregue emisiones cuando la aplicación no está activa e inicie así los procesos deseados. Las emisiones pueden provenir tanto del sistema (por ejemplo, cuando se apaga la pantalla, la batería está baja o el bluetooth cambia su estado), tanto como de otras aplicaciones (como cuando en una aplicación se descargan datos que podrían ser de utilidad para otra.

Los servicios son tareas diseñadas para desempeñarse en segundo plano y que pueden prevalecer si se descarta la actividad en donde se iniciaron o incluso si se cierra su aplicación de origen. Existen 3 tipos de servicio, cada uno dedicado a diversas situaciones. Por un lado, se tienen los servicios en segundo plano, es decir, aquellas tareas que ocurren sin que el usuario sea consciente de ello. Estos servicios ttienen una baja prioridad de ejecución, por lo que pueden ser interrumpidos en cualquier momento. Por ejemplo, en caso de que sea necesario que la aplicación comprima su almacenamiento cada tanto, se emplea un servicio en segundo plano para esta tarea. El usuario no sabrá cuando este servició esté en curso y el sistema puede suspender la tarea fácilmente si requiere liberar memoria o procesamiento. Los servicios de primer plano o servicios foreground son aquellos de los que el usuario es consciente. Siguen funcionando incluso si se deja de interactuar con la aplicación de origen y el sistema les da prioridad sobre otros procesos. Además, estos servicios deben mostrar una notificación que indique que se están ejecutando, ver Figura 10. Un ejemplo de estos, son los servicios de sincronización en aplicaciones como Whatsapp, en donde se informa por medio de una notificación que la operación se está llevando a cabo, pero no se puede interactuar con el proceso a menos que sea para detenerlo. Otra característica de estos servicios es que no permiten la interacción con ellos, esto no incluye solamente el usuario, sino a la actividad misma, pues no es posible manipular la IU (Interfaz de Usuario) desde un servicio en primer plano. Finalmente están los servicios de enlace, estos ofrecen una interfaz cliente-servidor que permite que los componentes interactúen con el servicio. Un ejemplo

de estos, son las aplicaciones de música, ya que la música sigue sonando incluso cuando se sale de la aplicación, pero se puede controlar la música, ya sea desde su notificación o desde alguna actividad de la aplicación, sin necesidad de detener el servicio (Google, 2021).

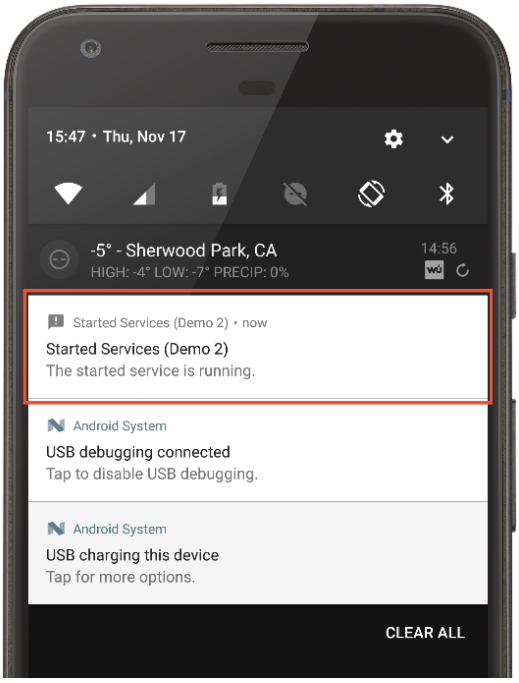


Figura 10. Notificación de un servicio en primer plano.

1.1 Definición del problema

Se requiere diseñar y desarrollar una aplicación apropiada con base en un teléfono celular para registrar las variables eléctricas de voltaje, corriente, potencia y energía que son leías y emitidas por un watthorímetro eléctrico. Esta línea de desarrollo tiene como antecedente el trabajo (de Gortari, 2019), en donde se desarrolla un watthorímetro que incluye una aplicación con una interfaz gráfica sencilla que se despliega en una computadora y en un teléfono celular.

En la propuesta de este trabajo, se tiene interés en el desarrollo de una aplicación que ofrezca un despliegue gráfico atractivo e intuitivo y un manejo del almacenamiento de información apoyado en la transferencia vía bluetooth.

1.2 Entorno actual

En el Laboratorio de Electrónica del ICAT se ha trabajado en el desarrollo de un watthorímetro eléctrico. Este watthorímetro se emplea para medir los parámetros de voltaje, corriente, potencia y energía que demande cualquier dispositivo al que se conecte; esto con el objetivo de analizar y evaluar las necesidades y limitaciones de energía de dicho sistema.

Después de pruebas y análisis, se consideró necesario el diseño de una aplicación móvil que mostrara la información recabada en una interfaz gráfica atractiva y amable para el usuario y que, a su vez, la almacenara en una base de datos para disponer de ellos cuando sea necesario.

1.3 Relevancia y limitaciones

Este trabajo, a pesar de estar basado en el uso de un watthorímetro digital, no incluye el diseño electrónico del mismo. La aplicación, incluso, está proyectada para poder usarse como base de un software más general que reciba, refleje y almacene información aun cuando no esté relacionado con los watthorímetros o dispositivos afines.

1.4 Objetivos y resultados esperados

1.4.1 Objetivos generales

Desarrollar una aplicación móvil robusta que permita la conexión con un watthorímetro eléctrico vía bluetooth para recibir, reflejar y almacenar información relacionada con el consumo energético de diversos dispositivos.

1.4.1 Objetivos específicos

- Desarrollar un software con el que se pueda recibir y visualizar datos de forma comprensible para el usuario.
- Desarrollar una base de datos para almacenar la información y tenerla disponible para el usuario.

2 REQUERIMIENTOS

La aplicación requiere conectarse al watthorímetro a través de un dispositivo inalámbrico. Esto implica que debe de ser capaz de acceder a la lista de dispositivos sincronizados, así como realizar una nueva búsqueda de dispositivos. La aplicación también debe ser capaz de iniciar una conexión con el watthorímetro, el cual enviará la información en una cadena de caracteres codificada. Esta codificación es necesaria para así poder verificar la integridad de los datos. Posteriormente, se decodificará la cadena y la información obtenida será reflejada en pantalla de forma que sea de fácil interpretación para el usuario.

La información obtenida deberá ser almacenada en una base de datos. De esta manera, si el usuario así lo requiere, podrá descargar un archivo con los datos desde la misma. Dado que la base guardará la información de cada prueba hecha, es necesario pensar en un medio por el cual sea posible discernir los datos de una prueba a los de otra. También deberá ser diseñada de forma que autogenere el ID de cada nuevo elemento.

Para la comodidad del usuario, será necesario crear un ambiente amigable, con una navegación intuitiva y una representación clara de los datos mostrados. Por otra parte, tendrá que mostrar los datos al usuario en tiempo real de una forma en que sean fácilmente interpretables. En (de Gortari, 2019), estos datos se mostraban por medio de una gráfica que se construía en tiempo real, sin embargo, esta resultó ser una forma poco atractiva y amable para el usuario. Las gráficas podrán obtenerse posteriormente mediante el archivo descargable.

3 DISEÑO

Se realizó un diagrama de casos de uso basado en los requerimientos mencionados, ver Figura 11. Hay dos tareas principales que el usuario podría realizar: la descarga de archivos y la recepción de datos. Estas tareas se analizaron para poder incluir en el diagrama acciones necesarias u opcionales para ambas. Primero se tiene la descarga de archivos. En este caso hay dos acciones previas que son imperativas para completar la tarea: seleccionar un archivo para descargar, y confirmar la descarga cuando el sistema así lo solicite. Por otra parte, se tiene la recepción de datos. Para iniciar la recepción de datos, es necesario que se haya conectado previamente a un dispositivo, es decir, el watthorímetro. Para esto, el bluetooth deberá estar encendido y el dispositivo al que se quiera conectarse necesitará estar disponible en pantalla. Por lo tanto, en caso de que alguna de estas condiciones no se cumpla, es necesario que el usuario pueda corregir el inconveniente. En consecuencia, se incluyen las acciones de encender el bluetooth e iniciar una nueva búsqueda de dispositivos.

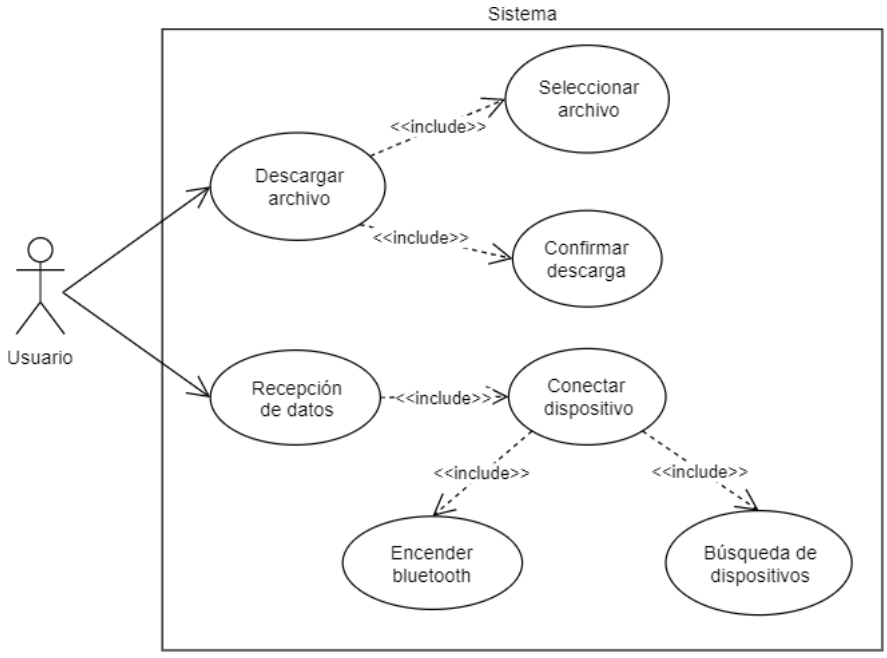


Figura 11. Diagrama de casos de uso de la aplicación.

A partir de lo anterior, se crea el siguiente sistema de bloques, ver Figura 12. En este se modela cómo una señal codificada ingresa al sistema, es recibida por el módulo de recepción de datos, y se decodifica en el mismo. Los datos obtenidos son almacenados en una base de datos y a la vez se reflejan en pantalla.

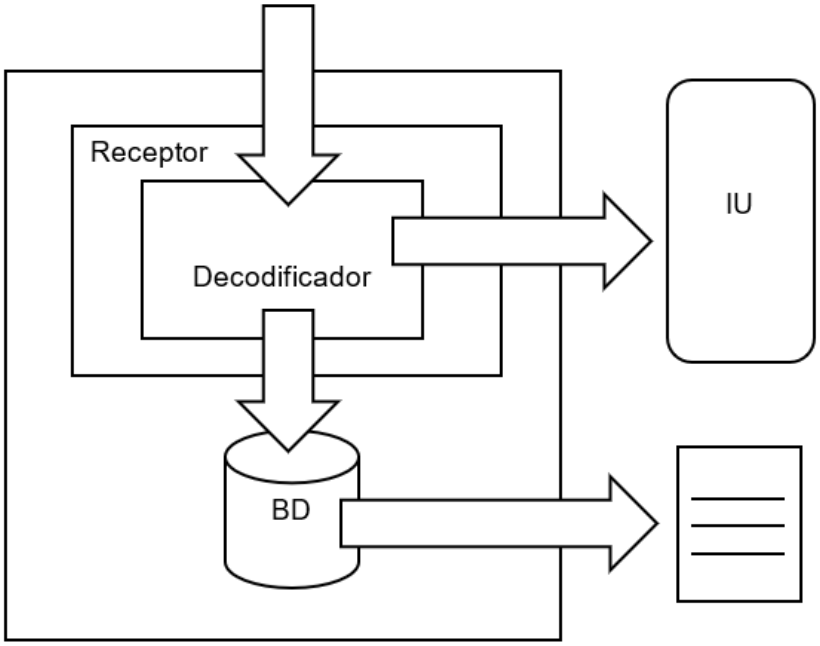


Figura 12. Diagrama de bloques de la aplicación.

Basándose en el diagrama de casos de uso y en el diagrama de bloques, se elaboró un prototipo de la aplicación. Dado que habrá solo 2 acciones principales que se puedan realizar con la aplicación (conectarse a un dispositivo para la recepción de datos y descargar archivos desde la base de datos), se optó por emplear un menú de bienvenida en donde se presenten estas dos opciones, ver Figura 13-a. Con excepción de la actividad principal, todas las actividades de la aplicación presentarán un botón de retorno en el appbar. El appbar es una barra en la parte superior de la pantalla en donde se muestra información y se disponen acciones relacionadas con la aplicación o la actividad en curso.

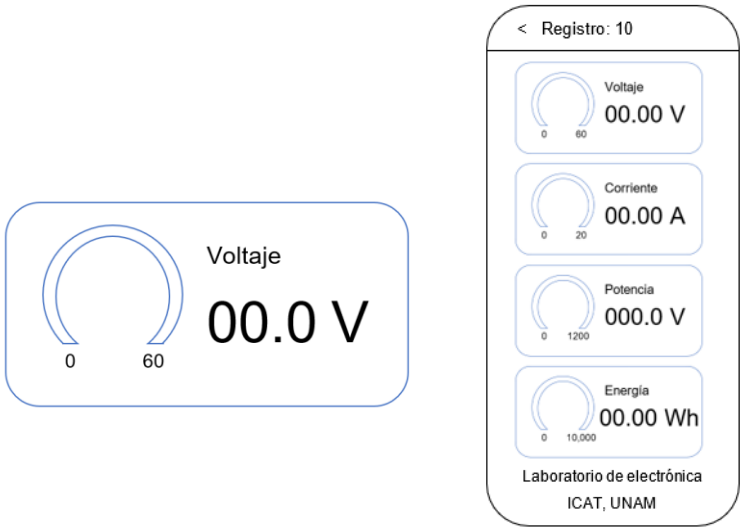
Si se elige conectarse a un dispositivo. La aplicación deberá redirigir al usuario a la segunda actividad, ver Figura 13-b. En esta actividad es desde donde el usuario tendrá acceso a los controles del bluetooth (encenderlo, apagarlo, buscar un dispositivo y conectarse a un dispositivo). Esta actividad deberá contar con un switch para encender y apagar el bluetooth, una lista de dispositivos bluetooth que ya hayan sido sincronizados con el teléfono del usuario, y un espacio para la lista de los dispositivos que se encuentren al iniciar una búsqueda, así como el botón para iniciar y detener la búsqueda. La leyenda de este botón cambiará de “Iniciar Búsqueda” a “Cancelar Búsqueda” según sea el caso.

a) b)

Figura 13. Imágenes prototipo del a) menú de bienvenida y b) la pantalla de conexión.

Una vez que la aplicación se haya conectado a algún dispositivo, la aplicación llevará a la actividad donde se realizará la recepción de datos. A nivel interno, se recibirá una cadena de caracteres, se decodificará y la información obtenida se almacenará en la base de datos. A nivel externo, esta información se visualizará en pantalla.

La forma en que se mostrarán los datos recibidos en pantalla también será importante. Es necesario que el usuario pueda comprender la situación general con un vistazo, pero también deberá poder tener acceso a los datos específicos. Con este fin, se propone crear para cada variable una tarjeta que contengan una serie de elementos útiles, ver Figura 14-a. El primer elemento es un medidor circular que podrá proporcionar una referencia visual intuitiva dentro de un rango de valores. Junto a este medidor, aparecerá el nombre de la variable referida y el número exacto del valor percibido, incluyendo sus respectivas unidades. Estas tarjetas serán puestas una tras otra en la pantalla vertical del dispositivo, abarcando las variables de voltaje, potencia, corriente y energía. En la parte inferior de la pantalla, aparecerá la leyenda “Laboratorio de electrónica, ICAT, UNAM” para referir al departamento e instituto en donde fue desarrollada la aplicación. Además, en el appbar se mostrará el registro en curso para conocer su ID correspondiente y poder buscarlo en la actividad de Descargas. Ver Figura 14-b.



a) b)

Figura 14. Imágenes prototipo de a) el medidor de voltaje y b) la pantalla de recepción de datos.

Se trabajará con una base de daros relacional y local. Esta contará con dos tablas. La primera tabla será “DATOS”. En esta tabla se registrarán el ID autogenerado y los datos obtenidos del watthorímetro (es decir: tiempo, voltaje, corriente, potencia y energía), así como el ID de su registro correspondiente. La segunda tabla será la tabla REGISTRO. Esta tabla se empleará para discernir un registro de datos de otro. Es decir, cuando se inicie una nueva prueba con el watthorímetro, se creará un nuevo registro para distinguir los datos de esa prueba de los de otras. En esta tabla se guardará el ID autogenerado del registro y la fecha en que se ha realizado. El diseño de la base de datos se muestra en la Figura 15.

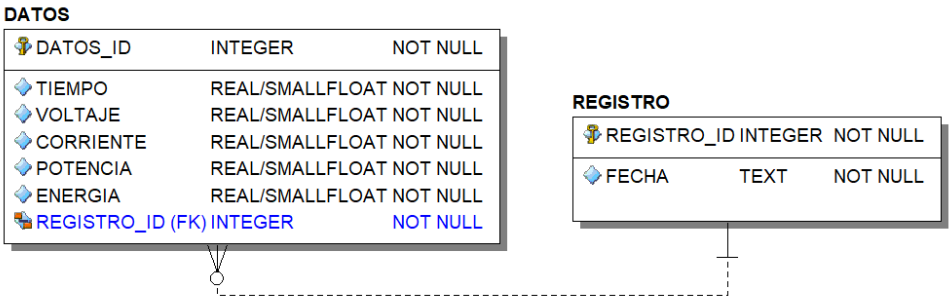


Figura 15. Modelo relacional de la base de datos

La cuarta actividad, a la cual se tendrá acceso desde el menú de bienvenida, permitirá descargar la información. La aplicación desplegará una lista de elementos identificados por el ID de cada registro y su fecha correspondiente. En esta etapa el usuario puede descargar los archivos con los datos correspondientes a dicho registro. Ver Figura 16.

Figura 16. Imagen prototipo de la pantalla de descargas.

En este trabajo se tiene la intención de crear, en primera instancia, una aplicación que funcione empleando actividades y receptores de emisiones. Las actividades son imprescindibles en cualquier aplicación que interactúe con el usuario, y los receptores de emisiones son útiles en tareas como la detección de dispositivos bluetooth y la recepción de datos que se llevará a cabo una vez que se haya enlazado al watthorímetro. Esta primera versión de la aplicación, aunque funcional, correrá el riesgo de detener la recepción de datos cuando se reciba una llamada o suene una alarma, lo que puede provocar que se pierdan datos. Para solucionar este problema, posteriormente se integrará un servicio en primer plano que será enlazado con la actividad de recepción de datos, el cual permitirá seguir recibiendo datos a pesar de que se deje de interactuar con la aplicación.

4 RECURSOS

Al momento de decidir el tipo de aplicación a desarrollar, se tomó en consideración la experiencia previa del desarrollador, así como la disponibilidad de los dispositivos adecuados para realizar las pruebas.

La aplicación ha sido diseñada para Android, el cual es el sistema operativo al que se tiene mayor acceso. Android es un sistema operativo de código abierto basado en el Kernel de Linux (Open Handset Alliance, s.f.). Una ventaja de programar para este SO, es que es el más usado en el mundo, superando incluso a Windows (Statcounter, 2023), lo que aumenta la probabilidad de que los usuarios de la aplicación tengan acceso a un dispositivo Android.

La aplicación se desarrolló en Java, el cual es un lenguaje de programación con el que se tiene experiencia previa. Java es un lenguaje de programación de propósito general orientado a objetos, creado por James Gosling y Bil Joy y comercializado por primera vez en 1995 por Sun Microsystems (Belmonte, 2005) (Oracle Corporation, 2022). Según el índice TIOBE, Java es el tercer lenguaje de programación más popular en el mundo, siendo antecedido tan solo por Python y C (TIOBE Software BV, 2023). Que Java sea un lenguaje de programación tan popular garantiza que será sencillo encontrar referencias de su uso. Así mismo, en la página oficial de Android para desarrolladores están disponibles diversos ejemplos y tutoriales en Java.

XML (Extensible Markup Language) es un lenguaje de marcado de propósito general (Mozilla Corporation, 2022). Este es el lenguaje típicamente usado para implementar la interfaz gráfica en las aplicaciones móviles desarrolladas con Java y por consiguiente es el empleado en el desarrollo de esta aplicación.

La base de datos será diseñada para funcionar con SQLite. SQLite es un motor de base de datos implementado como una biblioteca en lenguaje C y se encuentra integrado en todos los teléfonos móviles (SQLite, 2023). Al ser una herramienta

disponible y eficiente, se ha elegido trabajar con ella. Así mismo, Android ofrece bibliotecas especiales para trabajar con este motor de base de datos.

Además de esto, para su análisis, los datos se pueden descargar como archivos CSV (Comma Separated Values). Los archivos CSV son archivos de texto en donde se almacenan datos que bien podrían representarse en una tabla. En estos archivos, las filas de las supuestas tablas se distinguen con saltos de línea y los valores en cada una de estas filas se separan por medio de comas, emulando así las columnas. A partir de estos archivos se pueden crear gráficas y tablas para el debido análisis de sus datos. Por su sencillez, estos archivos son fáciles de compartir y almacenar, pues requieren poco espacio y pueden abrirse con cualquier programa de texto sin que la información resulte confusa, aunque abrirlos desde programas especializados facilita su interpretación.

Gradle es una herramienta de código abierto para automatizar la compilación de casi cualquier tipo de software (Gradle Inc., 2022). Es la herramienta de compilación que comúnmente se usa en Android, ya que es la que viene integrada con Android Studio y es la que se usará por esta misma razón.

Android Studio es el IDE (Entorno de Desarrollo Integrado) oficial de Android, cuenta con un potente editor de código y es un entorno pensado para el desarrollo móvil (Google, 2023). Todos los elementos mencionados convergen en Android Studio permitiendo así una experiencia de desarrollo óptima. Por este motivo, se ha elegido Android Studio como el IDE que se usará para desarrollar la aplicación.

Finalmente cabe mencionar, que la aplicación usa unos medidores circulares para mostrar los datos entrantes en tiempo real. Estos medidores fueron extraídos del repositorio (Kleczkowski, et al., 2019). La biblioteca fue ligeramente modificada para poder adaptarse a las necesidades de la aplicación.

5 DESARROLLO

5.1 MainActivity

Esta actividad es un menú de bienvenida. Este es el punto central de la aplicación, en donde se puede elegir entre iniciar una nueva sesión de pruebas o descargar archivos CSV con los datos registrados en pruebas anteriores. Esta actividad presenta un mensaje de bienvenida y dos botones, cada uno de los cuales redirige la aplicación a una actividad distinta, ver Figura 17.

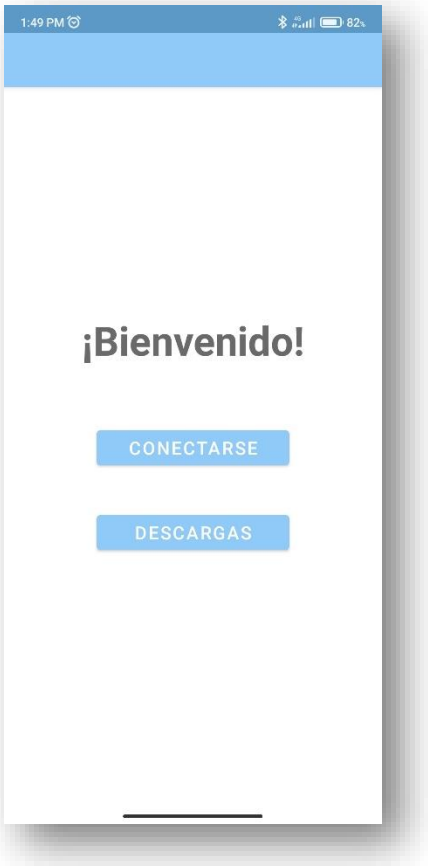


Figura 17. Captura de pantalla de Main Activity

En esta clase se corrobora la presencia del adaptador de bluetooth en el dispositivo. En caso de no encontrar el adaptador, se invisibilizan los botones y el

mensaje de bienvenida se cambia por una leyenda que notifica este inconveniente, ver Figura 18. Por otro lado, si el adaptador de bluetooth está presente, se procede con normalidad.

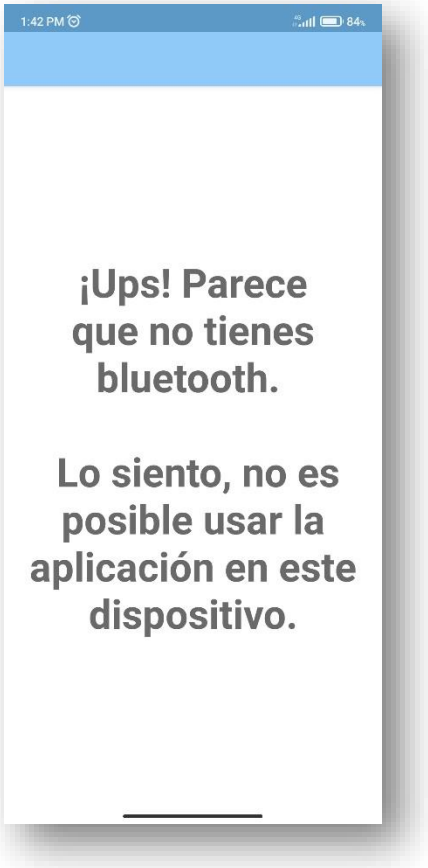


Figura 18. Captura de pantalla de MainActivity cuando no se encuentra el adaptador bluetooth.

5.2 ConnectActivity

Esta actividad es necesaria para poder iniciar conexión vía bluetooth con el watthorímetro. La piedra angular de esta actividad es un objeto de tipo *BluetoothUtils*, la cual es una clase que se creó con el propósito de manejar las operaciones bluetooth de la aplicación.

En esta actividad es posible encender y apagar el bluetooth mediante un SwitchCompat. Dado que habilitar y deshabilitar el bluetooth puede tomar algún tiempo al sistema, se vio conveniente que, cada que se cambia el estado del SwitchCompat, este es remplazado por un ProgressBar para indicar que se está realizando el cambio. Una vez que la operación ha sido concluida, el SwitchCompat vuelve a tomar su lugar indicando el estado del bluetooth (encendido o apagado), ver Figura 19.

Figura 19. El SwitchCompat que controla el estado del bluetooth cuando está a) encendido, b) apagado y c) el Progressbar que lo remplaza mientras se realiza el cambio.

El funcionamiento del SwitchCompat se ilustra en el diagrama de flujo de la Figura 20. Cabe aclarar que en el diagrama solo se muestran los procedimientos relacionados con el manejo del switch y se omiten algunas operaciones que están más relacionadas con el encendido y apagado del bluetooth.

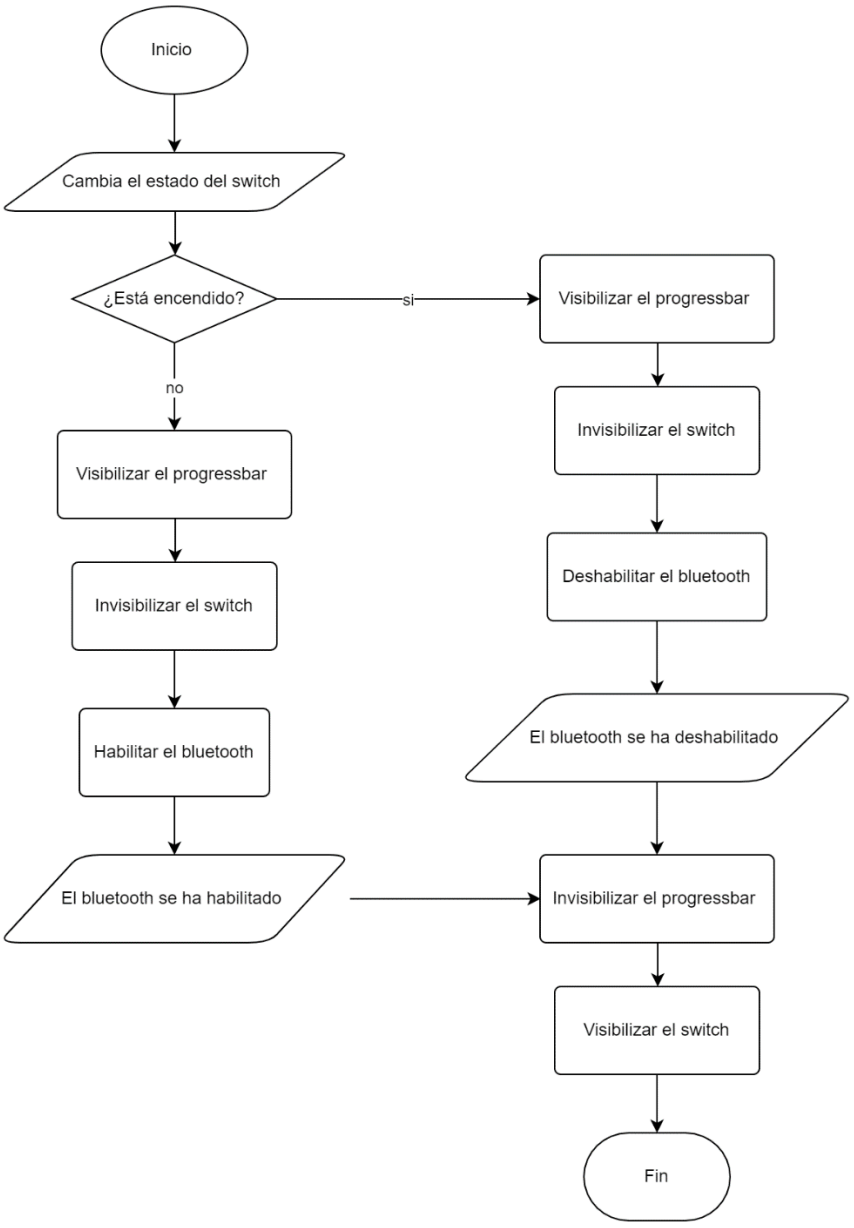
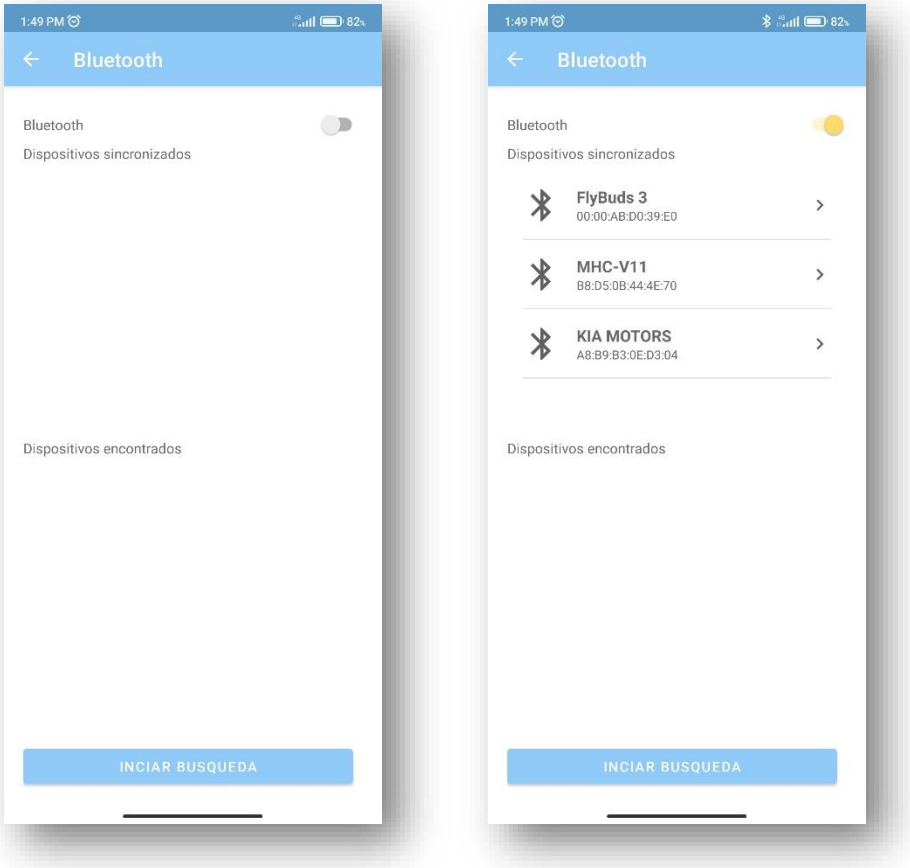


Figura 20. Diagrama de flujo del funcionamiento del SwitchCompat de encendido/apagado del bluetooth.

Si el bluetooth está encendido, se mostrará en pantalla una lista de dispositivos sincronizados. Esta lista se obtiene del adaptador de bluetooth, al que se tiene acceso desde el objeto de tipo *BluetoothUtils*. La lista se presenta mediante un

RecycleView. Esta lista será visible en pantalla mientras el bluetooth esté encendido. En la Figura 21 se puede ver como la lista no es visible cuando el bluetooth esta apagado y como se muestra una vez que el bluetooth se enciende.



a) b)

Figura 21. ConnectActivity cuando a) el bluetooth está apagado y b) el bluetooth está encendido.

Mientras el bluetooth este habilitado, es posible realizar una búsqueda de dispositivos. Esto se hace en caso de que el dispositivo requerido no se encuentre en la lista de dispositivos sincronizados. Para solicitar la búsqueda, basta con presionar sobre el botón INICIAR BUSQUEDA. Al momento de hacer esto, lo primero que se hace es verificar si se cuenta con el permiso

ACCESS_FINE_LOCATION, el cual es un permiso en tiempo de ejecución necesario para realizar la búsqueda de dispositivos. Los permisos en tiempo de ejecución son permisos que manejan acciones y/o información sensible y, por lo tanto, es necesario pedirlos expresamente al usuario en caso de no contar con ellos, ver Figura 22. En este caso, si el usuario se niega a otorgar el permiso, toda la operación se detiene. Por otra parte, si el usuario accede, se inicia la búsqueda de dispositivos.

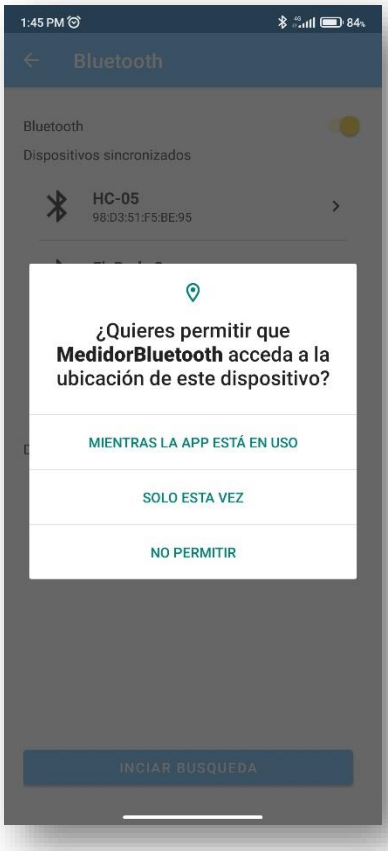


Figura 22. Petición del permiso ACCESS_FINE_LOCATION

El resto de los permisos que maneja la aplicación, son permisos en tiempo de instalación, lo que significa que son otorgados al momento en que el usuario decide instalar la aplicación en su dispositivo. Los permisos en tiempo de instalación con los que cuenta la aplicación son:

- BLUETOOTH, que es necesario para conectarse a otros dispositivos.
- BLUETOOTH_ADMIN, que permite controlar la configuración de bluetooth y es necesario para que las aplicaciones descubran y emparejen dispositivos bluetooth.
- BLUETOOTH _SCAN, es necesario para buscar y emparejar dispositivos bluetooth cercanos.
- BLUETOOTH_CONNECT, que es necesario para conectarse a dispositivos bluetooth emparejados.

Para realizar la búsqueda de dispositivos se emplea un método de la clase *BluetoothUtils* llamado *discovery(boolean)*, que a su vez llama a un método del adaptador del bluetooth llamado *startDiscovery().* Sin embargo, este método no es suficiente para poder mostrar los resultados al usuario, ya que este método solo busca los dispositivos, mas no los almacena en ninguna parte. Para poder retener su información, es necesario que el sistema informe a la aplicación cada que se encuentre un nuevo dispositivo. Esto se consigue por medio de un *BroadcastReceiver*, el cual es un receptor de emisiones encargado de recibir los mensajes que emite el sistema respecto a ciertos eventos de interés. Al *BroadcastReceiver* se le indican los eventos de interés, de modo que sepa qué mensajes entregar. En el caso de esta aplicación, el evento de principal interés es cuando se encuentra un dispositivo nuevo. Este evento se registra con la constante ACTION_FOUND que ofrece la clase *BluetoothDevice*. Cada que el Broadcast es notificado de este evento, se recupera un objeto de tipo *BluetoothDevice* del mensaje recibido, se corrobora que ese dispositivo no haya sido registrado en cualquiera de las dos listas con anterioridad y se agrega a la lista de dispositivos encontrados, ver Figura 23.

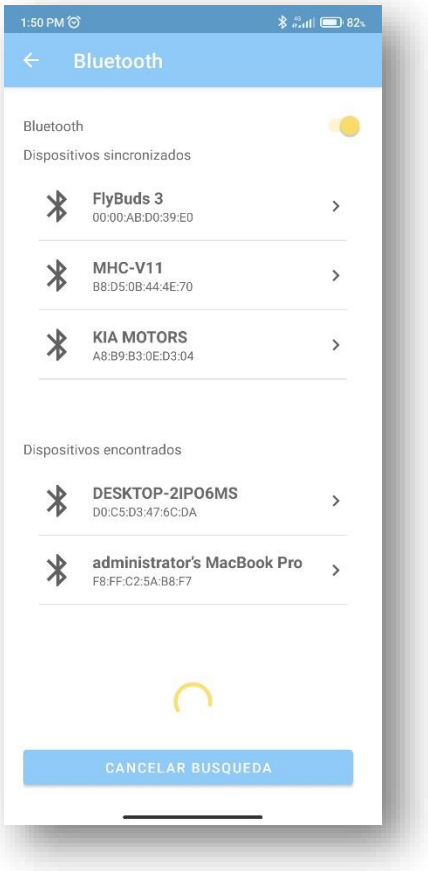


Figura 23. ConnectActivity durante la búsqueda de dispositivos.

Aunque ACTION_FOUND fue el evento registrado más importante, no fue el único. Se registraron otros tres eventos, cuyas constantes las ofrece el adaptador de bluetooth. Estos eventos son:

- ACTION_STATE_CHANGED: notifica cuando el estado de bluetooth cambia de habilitado a deshabilitado o a la inversa, de modo que se pueda manejar el estado del SwitchCompat relacionado al bluetooth de encendido/apagado y la visibilidad del ProgressBar correspondiente.
- ACTION_DISCOVERY_STARTED: indica cuando se ha iniciado una nueva búsqueda de dispositivos. Una vez iniciada la búsqueda, se muestra un ProgressBar en la parte baja de la pantalla. Además, el botón que inicia la

búsqueda ahora es el encargado de detenerla, por lo que el mensaje que se muestra en él es “CANCELAR BUSQUEDA”. En esta actividad se maneja una bandera llamada *discoveryFlag* por medio de la cual se puede indicar al método *discovery(boolean)* de la clase *BluetoothUtils* si hay una búsqueda en curso o no. Esta bandera se enciende cuando la búsqueda inicia y se apaga cuando concluye.

- ACTION_DISCOVERY_FINISHED: indica cuando la búsqueda de dispositivos ha finalizado. La bandera *discoveryFlag se apaga,* el ProgressBar desaparece y se restablece la función de búsqueda del botón, por lo que el mensaje que muestra vuelve a ser “INICIAR BÚSQUEDA”.

La lista obtenida por la búsqueda de dispositivos se muestra en pantalla, refrescándose cada que un nuevo dispositivo es agregado. Al seleccionar un elemento, sin importar la lista de donde provenga, el sistema intentará conectarse al dispositivo seleccionado. Si el dispositivo no ha sido previamente sincronizado, se desplegará una solicitud para sincronizar los dispositivos. Si el dispositivo ya ha sido sincronizado, o una vez que haya sido sincronizado, se procederá a iniciar la conexión. Si la conexión es exitosa, se redirigirá a la actividad *MessageReceiverActivity* donde se llevará a cabo la recepción de datos. En el diagrama de la Figura 24 se ilustra de forma simplificada lo que ocurre cada que el bluetooth se habilita o deshabilita y los procesos relacionados con la búsqueda de dispositivos. En el diagrama solamente se muestran los procesos internos, omitiendo cualquier cambio que haya en la IU durante estos procesos.

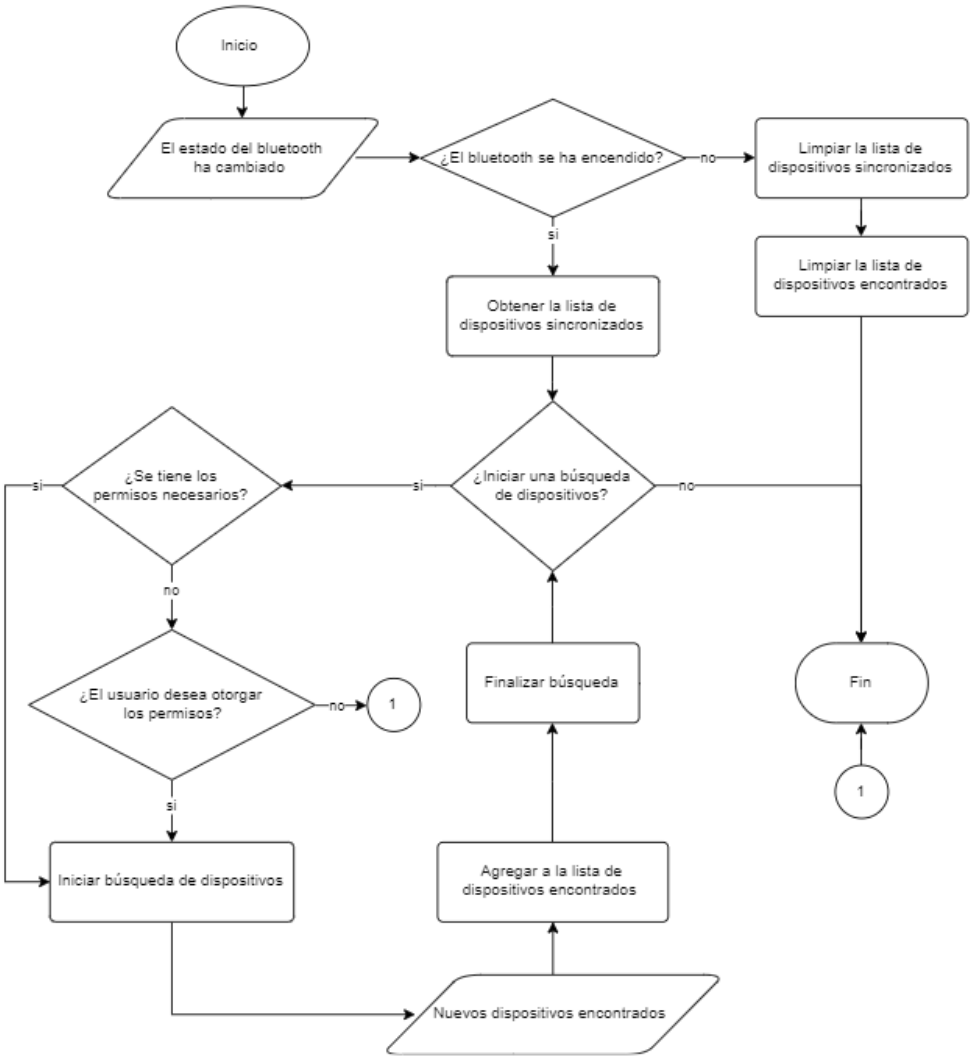


Figura 24. Diagrama de flujo de las funciones bluetooth en la actividad ConnectActivity.

La conexión a otro dispositivo se realiza mediante una clase llamada ConnectAsyncTask. Esta clase sirve para crear un hilo o subproceso encargado de realizar la conexión vía bluetooth a otro dispositivo. Para crear esta clase, se tomaron dos consideraciones. La primera es que, al momento de conectarse al watthorímetro para comenzar la recepción de datos, habrá un cambio de actividad.

Es necesario que este cambio se ejecute si y solo si se consigue una conexión exitosa. Esto también implica que se realice el cambio solo cuando la conexión ya se ha logrado y no antes, pues la conexión a otro dispositivo es un proceso que dura algunos pocos segundos y si el cambio de actividad ocurre dentro de este tiempo, podría interrumpir el proceso de conexión evitando que se logre.

La segunda consideración está relacionada con el hecho de que esta operación toma más tiempo del que se puede permitir en las tareas que se desempeñan sobre el hilo principal, pues podría bloquearlo si se ejecuta sobre este, por lo que es necesario realizarlo en otro hilo. Cuando se inicia una aplicación en Android, se crea un proceso sobre el cual se correrá la misma. Este proceso crea su propio subproceso o hilo principal. Este es aquel que interactúa con los componentes del kit de la IU en Android y el que se encarga de los eventos de dibujo, por este motivo, también es llamado hilo de la IU. Android no crea por sí mismo subprocesos diferentes para las diversas tareas, sino que todas se ejecutan en el hilo principal colocándose una detrás de otra en una cola de espera. Sin embargo, esto puede crear problemas cuando alguna de estas tareas tarda más que algunos milisegundos en ejecutarse, ya que esto impedirá que el usuario interactúe con la aplicación o que se realice cualquier otra tarea hasta que la tarea en curso haya concluido. Para evitar esto, es necesario crear nuevos hilos que se ejecuten en segundo plano mientras que el subproceso principal sigue corriendo sin problemas. Sin embargo, realizar modificaciones a la IU desde estos hilos secundarios podría generar problemas en la aplicación, por lo que no se debe acceder al paquete de herramientas de la IU de Android desde otro que no sea el subproceso principal.

Android ofrece una clase para manejar subprocesos que permite indicar acciones a seguir una vez que ya se ha realizado la tarea y estas acciones se ejecutaran sobre el subproceso principal. Esta clase se llama AsynTask. AsyncTask funciona bien solo con operaciones de corta duración, por lo que no es recomendable emplearlo para tareas que duran más que unos pocos segundos. En este caso, la

conexión a otro dispositivo es una tarea relativamente corta, por lo que resulta ideal.

AsyncTask cuenta con 3 métodos principales:

- doInBackground(Void...): es en donde ocurren las tareas que se quiere realizar en segundo plano. En este caso, es aquí donde se emplea el socket anteriormente creado para solicitar la conexión con el otro dispositivo y esperar su respuesta.
- onProgressUpdate(Void...): es donde se expresa el progreso de la tarea y es necesario colocarlo siempre, a pesar de que no se use expresamente, como es el caso de esta aplicación.
- onPostExecute(Boolean): es el método que se realiza al terminar la tarea asignada en doInBackground(Void...). Este método también tiene la característica de que se realiza sobre el hilo principal. En este caso, aquí es donde se realiza el cambio de actividad para comenzar con la recepción de datos una vez que se ha realizado la conexión. Aquí también se entrega el socket creado a la clase ShareSocket, con la cual se compartirá el socket con la siguiente actividad. ShareSocket es una clase diseñada para compartir el socket de la conexión bluetooth entre actividades. Todos sus métodos son de tipo *synchronized*, lo que les permite poder trabajar en sincronicidad con otros hilos.

Finalmente, al momento en que se destruye la actividad, se elimina el registro del Broadcast pues ya no es necesario estar atentos a las notificaciones que ofrece.

5.3 MessageReceiverActivity

Esta es la actividad donde se recibe la señal enviada desde el watthorímetro. Esta señal entrega una serie de cadenas de caracteres codificadas. La codificación no es más que un formato sencillo que se le ha dado a cada cadena con la intención de asegurar la integridad de los datos. Dado que estas cadenas tienen una serie

específica de caracteres al inicio y otra al final, es posible identificar con claridad si una cadena se presenta completa o si ha sido comprometida durante la transmisión. La aplicación solo aceptará aquellas cadenas que garanticen que sus datos se encuentran íntegros.

La recepción de datos será una tarea que dure tanto como la actividad se encuentre en uso, por lo que, si se quisiera ejecutar esta tarea en el subproceso principal, se perdería toda interacción con la aplicación y después de un tiempo, el sistema la daría de baja. Para poder controlar la constante recepción de datos al mismo tiempo que se permite interactuar con la pantalla, es necesario usar un subproceso y buscar el modo de comunicar al hilo principal cada que se termine una lectura para actualizar la interfaz gráfica. Además, se debe considerar la posibilidad de que se tenga que atender otras tareas en el dispositivo que obliguen a la actividad a perder el foco. En estos casos, los procesos de la actividad se frenarían en el momento en que se deje de ver la actividad en pantalla, por lo que se debe buscar un medio por el cual se mantengan estos procesos en funcionamiento y garantizar que el sistema no los descartará fácilmente.

Para resolver estos problemas, se emplea un servicio en primer plano que, a su vez, estará enlazado con la actividad. El servicio en primer plano permitirá seguir recibiendo datos incluso si la aplicación ha perdido el foco. Al iniciar el servicio, se creará un nuevo registro en la tabla REGISTRO de la base de datos. Este número de registro será aquel al que serán asociados los datos recibidos durante la sesión en curso. Los servicios en primer plano deben mostrar una notificación mientras estén activos para que el usuario sea consciente de que hay un proceso en ejecución. En este caso, en la notificación aparecerán el número del registro y una línea con los datos recibidos en tiempo real, de forma que se tenga acceso a ellos aun mientras se atienden otras actividades, ver Figura 25. Por otro lado, es necesario enlazar el servicio con la actividad, pues, aunque el servicio en primer plano permite recibir y almacenar los datos, es imposible reflejar esta información en pantalla sin antes realizar este procedimiento.

Figura 25. Notificación del servicio en primer plano.

Desde el servicio, se recupera el socket creado en la actividad anterior mediante ShareSocket, para poder recibir los datos. Posteriormente, se emplea un hilo para leer línea por línea los mensajes recibidos, se eliminan los posibles espacios que se puedan encontrar y se compara la cadena con una expresión regular para corroborar la integridad de los datos. Este es un mensaje cifrado, por lo que se envía a un Handler para su decodificación. De igual modo, si la conexión se pierde, se notifica al *Handler* de esto para que cierre el socket. Todo este proceso se ilustra de forma simplificada en el diagrama de flujo de la Figura 26.

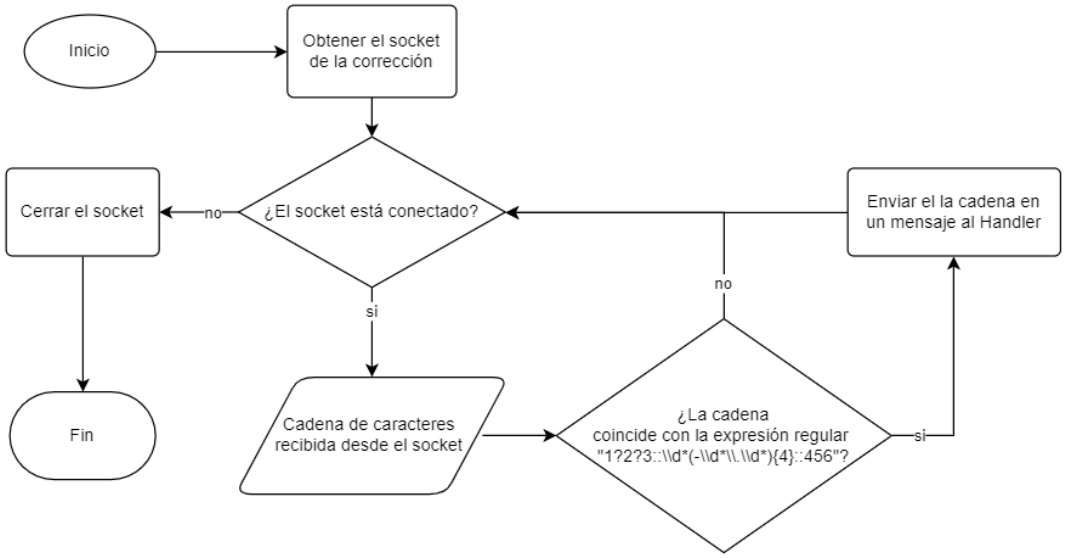


Figura 26. Diagrama de flujo del hilo ConnectedThread

Un *Handler* es un objeto que sirve para enviar, recibir y procesar mensajes, es decir, objetos de tipo *Message* así como modificar la IU. Si se necesita modificar la IU desde un subproceso que implique una tarea de larga duración, se puede usar un *Handler*. Cada *Handler* está asociado a un hilo y su respectiva cola de mensajes, también llamada looper. En este caso, se usa un *Handler* como decodificador. El hilo anteriormente mencionado recibe los datos desde el watthorímetro y lo manda al *Handler* a través de un objeto de tipo *Message*. En el *Handler* se recupera la cadena de caracteres codificada y se realiza el proceso de decodificación.

El formato de la cadena es el siguiente:

- El principio de la cadena debe coincidir con la secuencia de caracteres “123::”. Aunque, dado que el comienzo de la cadena suele ser la información que más fácilmente se pierde, también se aceptan las secuencias “23::” y “3::” si y solo si, el resto del formato de la cadena se mantiene.

- La siguiente parte de la cadena son los datos, estos se presentan uno detrás de otro, separados cada uno por un guion corto. Todos son números decimales positivos salvo por el tiempo, el cual es un numero entero positivo. El orden de aparición es: tiempo, voltaje, corriente, potencia y energía.
- Enseguida de esto, aparece la secuencia de cierre “::456” la cual se espera que siempre se encuentre completa.

El hilo comprueba el formato de esta cadena mediante la expresión regular "1?2?3::\\d*(-\\d*\\.\\d*){4}::456". Si la cadena entrante coincide con este formato, entonces se envía al *Handler* como un mensaje de tipo MESSAGE_READ, los cuales se identifican con una constante de tipo entero creada para este propósito. Además de MESSAGE_READ, el *Handler* recibe también mensajes de tipo CONNECTION_LOST.

Cuando el *Handler* recibe un mensaje de tipo MESSAGE_READ, es porque se ha recibido una cadena codificada aceptable y hay que decodificarla y almacenarla. Para decodificar la cadena, esta se va seccionando. Primero se eliminan los caracteres de apertura y de cierre, y luego se registra el primer dato, se almacena en un objeto de tipo TablaDatos, que es una representación de una fila de la tabla Datos de la base de datos, y se elimina esa parte de la cadena. Este proceso se repite hasta que se han registrado todos los datos. Posterior a esto, los datos se cargan a la base de datos y se reflejan en pantalla, ver Figura 27.

Figura 27. MessageReceiverActivity durante la recepción de datos.

Por otro lado, cuando el *Handler* recibe un mensaje de tipo CONNECTION_LOST es porque la conexión se ha perdido. En este caso, se remplazan los valores de las diversas variables mostradas en pantalla por la secuencia de caracteres “--.--“ y se muestra un mensaje de “Conexión perdida” mediante un Toast, ver Figura 28. Posterior a esto, el servicio se detiene a sí mismo para impedir que siga consumiendo recursos.

El servicio también puede detenerse simplemente saliendo de la actividad, pues la instrucción de desenlazar el servicio y detenerlo se encuentra en la devolución de llamada onDestroy() de la actividad. El socket de la conexión se cerrará en el momento en que el servicio se detenga sin importar si es porque se ha perdido la conexión o porque se ha destruido la actividad.

Figura 28. MessageReceiverActivity cuando la conexión se ha perdido.

Por último, cabe mencionar que la actividad puede pasar por el estado Destroyed si se gira la pantalla y el servició también podría detenerse por este evento. Esto podría resultar problemático para el manejo de la aplicación, por lo que se ha tomado la decisión de bloquear la pantalla en su orientación vertical. Por esto mismo, tampoco fue necesario crear un diseño de pantalla horizontal para esta actividad.

5.4 DownloadActivity

Esta es la actividad desde la que se accede a la base de datos. En esta se muestra simplemente un RecycleView que enlistan los diversos registros que hay almacenados en la base de datos desde el más reciente hasta el más antiguo, ver

Figura 29. Al presionar sobre un elemento de la lista, se inicia la solicitud de descarga de archivo. Se pregunta al usuario si desea guardar el archivo a descargar y, si la respuesta es afirmativa, se crea el archivo.

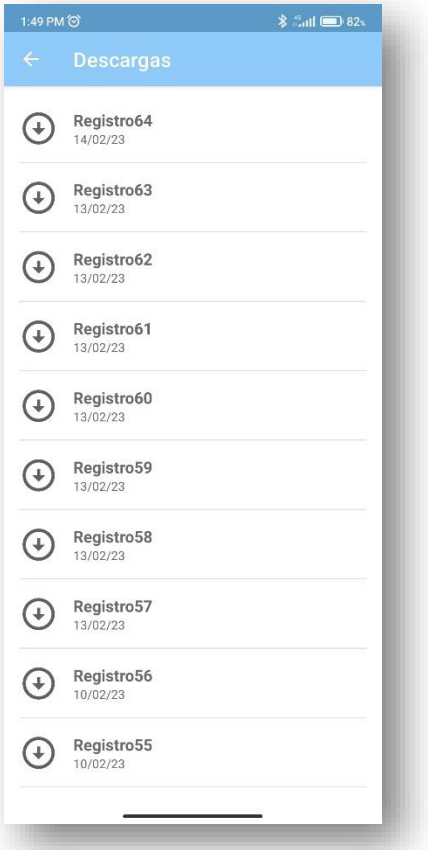


Figura 29. DownloadActivity.

6 PRUEBAS

Se realizaron dos tipos de pruebas. Las primeras fueron pequeñas pruebas de soporte durante la etapa de desarrollo. Se realizaron con un circuito en una protoboard, diseñado por el M. I. Juan Ricardo Damián Zamacona. Este dispositivo, en lugar de suministrar datos reales de voltaje, corriente, potencia y energía, envía una lista de datos predeterminados. Se empleó como una guía durante la etapa de desarrollo, de forma que fuera comprobable el adecuado funcionamiento de la aplicación y que permitiera hacer las modificaciones pertinentes en caso de haber problemas.

La aplicación, junto con un watthorímetro eléctrico, ver Figura 30, fueron presentados en el Congreso de Instrumentación SOMI XXXVI antes de ser concluida (Castillo, et al., 2022). En ese momento, aun o se integraba el servicio a la aplicación, el cual impide que esta deje de funcionar si se deja de interactuar con la actividad de recepción de datos. Por esto mismo, las pruebas tuvieron que ser vigiladas para impedir que la pantalla del celular se apagara.

Posteriormente, se integró el servicio a la aplicación y se realizaron pruebas empleando el mismo watthorímetro eléctrico antes mencionado, Este watthorímetro se conectó a una fuente de voltaje, de forma que el voltaje recibido se pudiera controlar manualmente. Así mismo, el watthorímetro tiene una pantalla en la cual se pueden observar los valores registrados para poder compararlos con los recibidos en la aplicación.

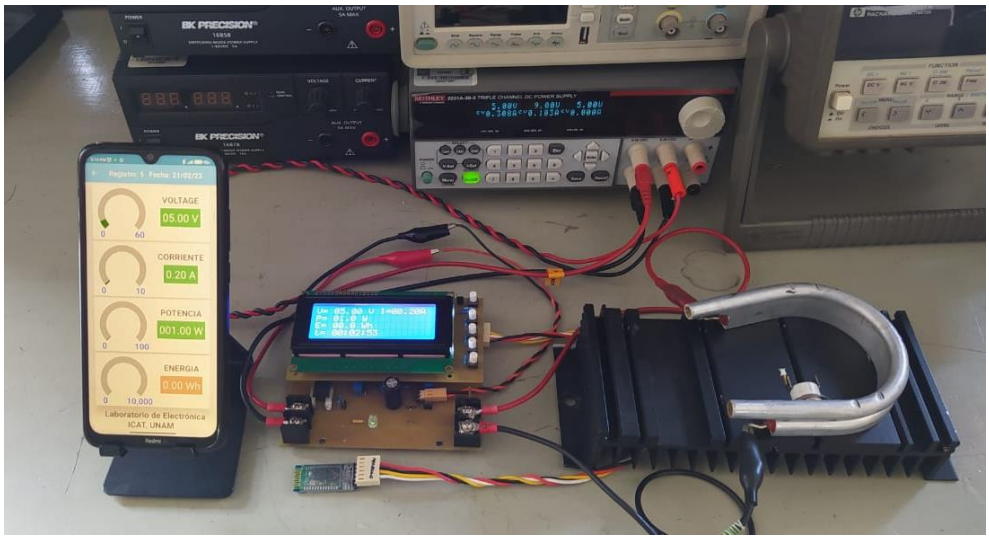


Figura 30. Aplicación en funcionamiento con un watthorímetro eléctrico.

Para esta etapa, se crearon 3 perfiles diferentes con el objetivo de generar una señal acorde de forma manual empleando la fuente de voltaje. Estos perfiles se basan en 3 funciones distintas las cuales fueron graficadas. Los datos se administraron de forma manual, por lo que no fue posible obtener la forma exacta de las gráficas esperadas. Sin embargo, para conseguir una aproximación adecuada, se decidió tomar los datos cada tantos minutos y redondearlos de considerarlo conveniente.

El primer perfil corresponde a la Función (1).

$$f(x) = \begin{cases} \frac{2x}{5} + 6, & x < 30 \\ 18, & 30 \leq x \leq 40 \\ -\frac{2x}{5} + 34, & x > 40 \end{cases} \quad (1)$$

Esta función está conformada por dos rampas y una constante. En la Figura 31 se puede observar su gráfica correspondiente.

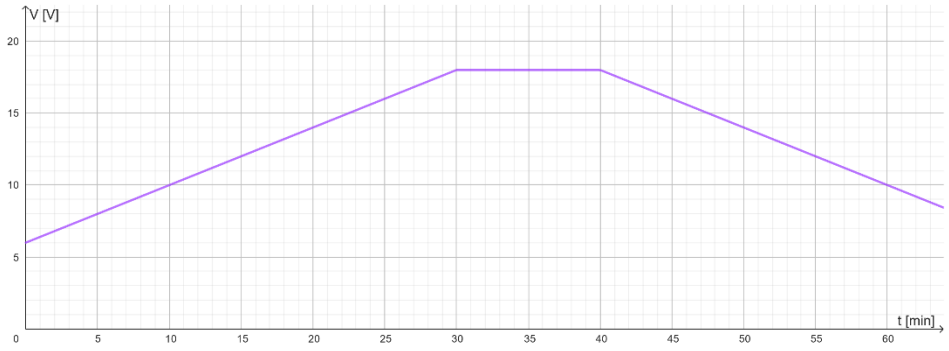


Figura 31. Gráfica correspondiente al perfil de la primera prueba.

Los datos recopilados de esta función se tomaron en intervalos de 5 minutos, dando como resultado la siguiente tabla. Cabe mencionar que hubo un pequeño error al transcribir los datos, por lo que la prueba se hizo con un voltaje de 5 [V] en el minuto 0 en lugar de su correspondiente original de 6 [V].

Tabla 1. Perfil de la primera prueba

TIEMPO [MIN]	VOLTAJE [V]
0	5
5	8
10	10
15	12
20	14
25	16
30	18
35	18
40	18
45	16
50	14
55	12
60	10

Tras realizar la prueba correspondiente y descargar los datos de la base de datos, estos fueron graficados con Excel. En las Figuras 32 a la 35 se puede observar las

gráficas de voltaje, corriente, potencia y energía resultantes. Como se puede observar, la gráfica de voltaje coincide con lo esperado.

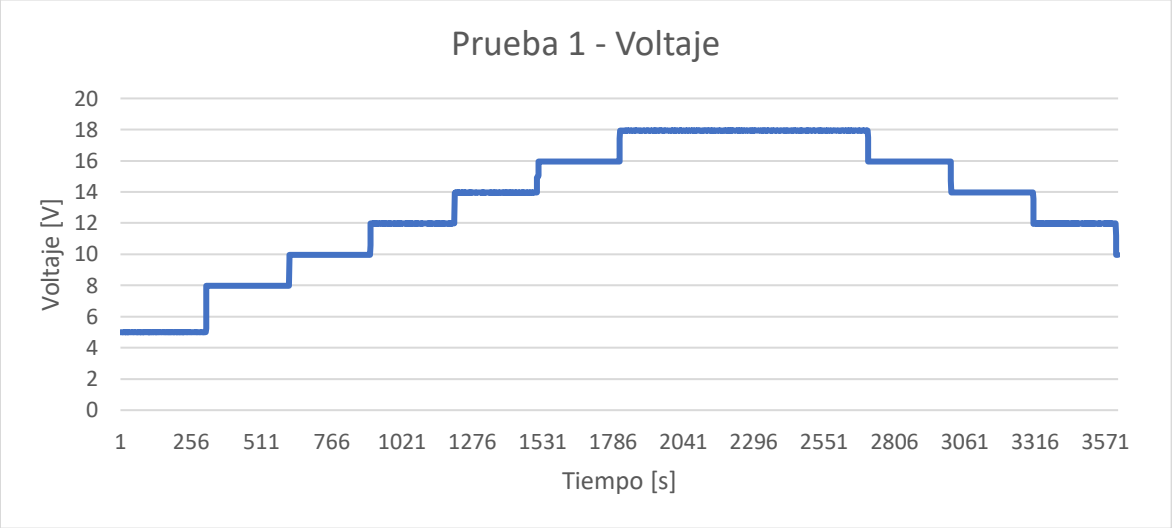


Figura 32. Gráfica del voltaje registrado en la primera prueba.

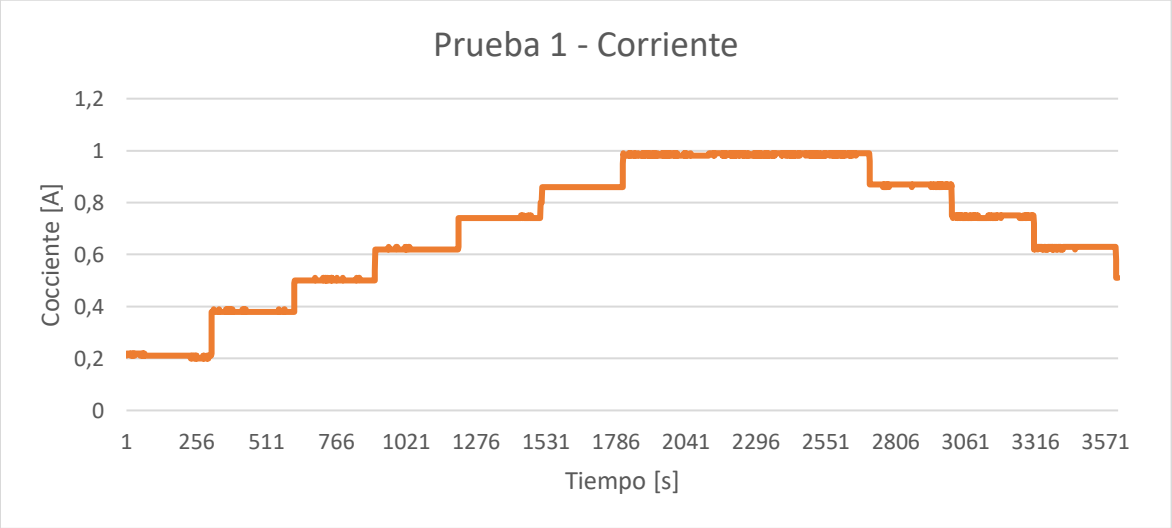


Figura 33. Gráfica de la corriente registrada en la primera prueba.

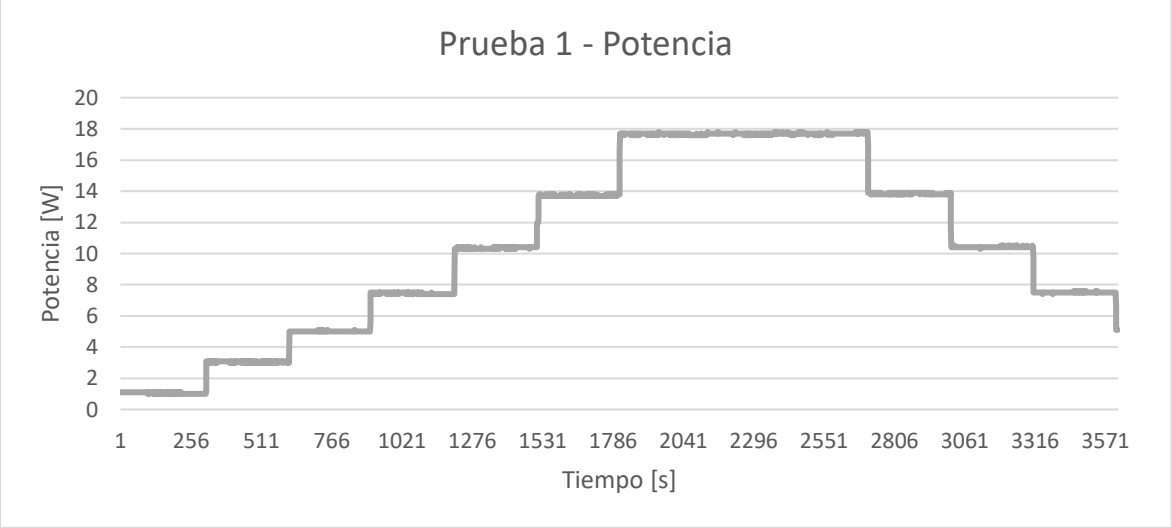


Figura 34. Gráfica de la potencia registrada en la primera prueba.

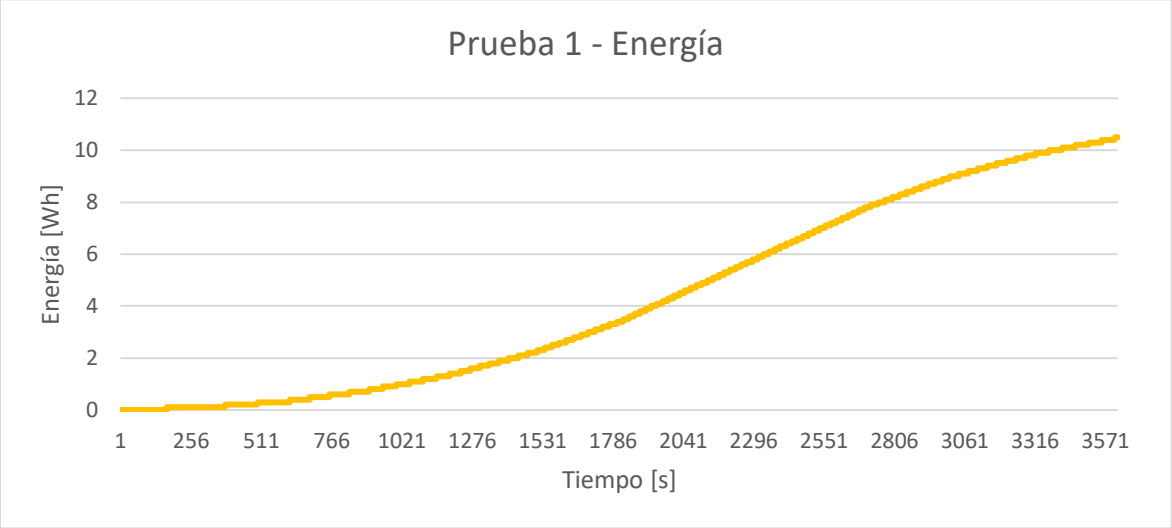


Figura 35. Gráfica de la energía registrada en la primera prueba.

El segundo perfil se trata de una función cuadrática. Se buscó que el valle de esta función coincidiera con el minuto 30, de forma que se pudiera apreciar la forma simétrica de esta función. Así mismo, se buscó que el voltaje en el minuto 0 fuera de 25 [V]. El resultado de ello fue la Función (2) y su gráfica correspondiente se muestra en la Figura 36.

$$f(x) = \frac{(x-30)^2}{45} + 5 \qquad (2)$$

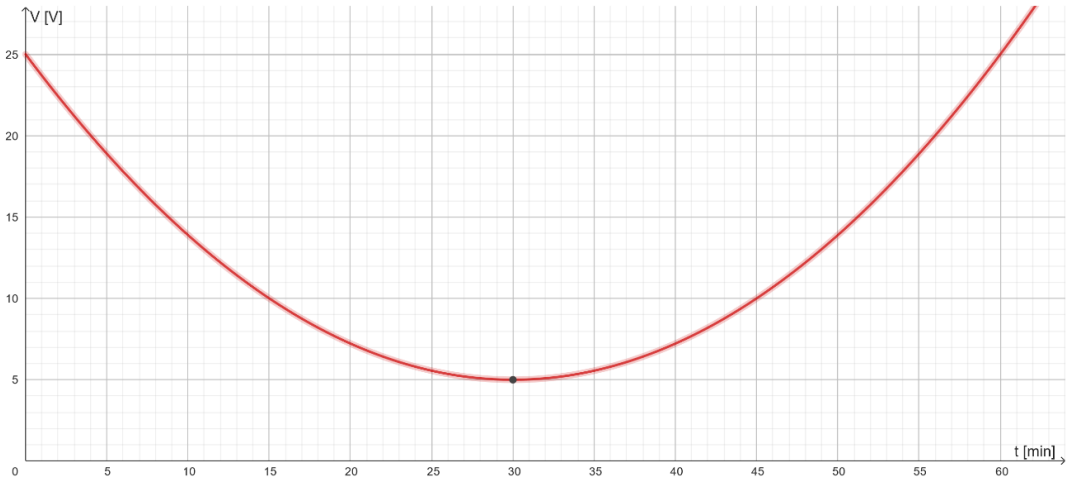


Figura 36. Gráfica correspondiente al perfil de la segunda prueba.

De esta Gráfica se tomaron los datos en intervalos de 2 minutos. Los datos fueron redondeados con el objetivo de simplificar la prueba, por lo que no coinciden exactamente con la función dada. Los datos obtenidos se muestran en la siguiente tabla.

Tabla 2. Perfil de la segunda prueba

TIEMPO [MIN]	VOLTAJE [V]
0	25
2	22.4
4	20
6	17.8
8	15.8
10	13.9
12	12.2
14	10.7
16	9.4
18	8.2
20	7.2
22	6.4
24	5.8
26	5.4
28	5.1
30	5

TIEMPO [MIN]	VOLTAJE [V]
32	5.1
34	5.4
36	5.8
38	6.4
40	7.2
42	8.2
44	9.4
46	10.7
48	12.2
50	13.9
52	15.8
54	17.8
56	20
58	22.4
60	25

Las gráficas obtenidas tras realizar la prueba se muestran en las Figuras 37 a la 40. En ellas se puede observar un cambio abrupto al final, en donde el voltaje, la corriente y la potencia descienden drásticamente; esto es debido a un error humano al momento de cambiar el voltaje.

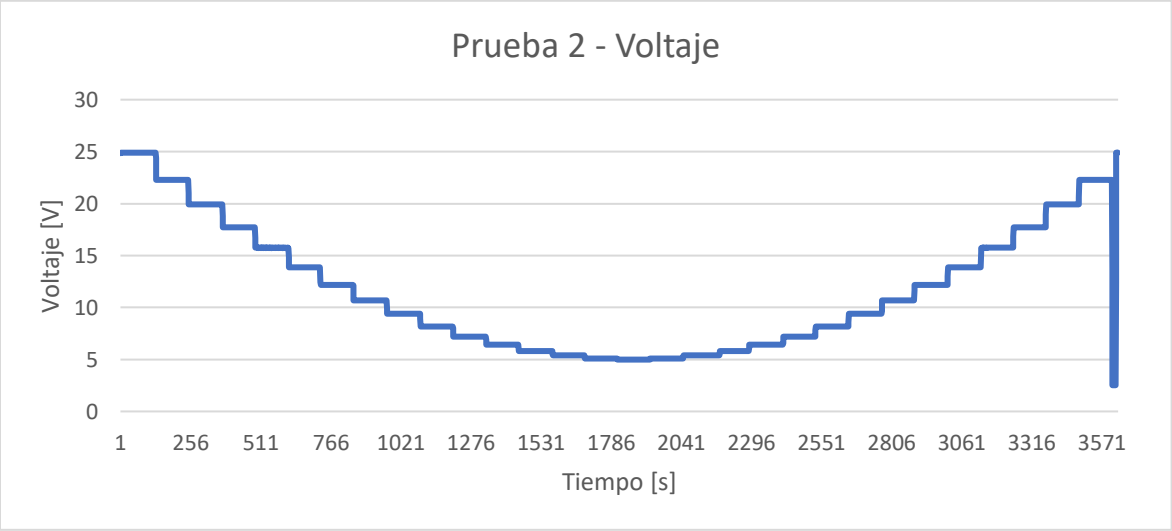


Figura 37. Gráfica del voltaje registrado en la segunda prueba.

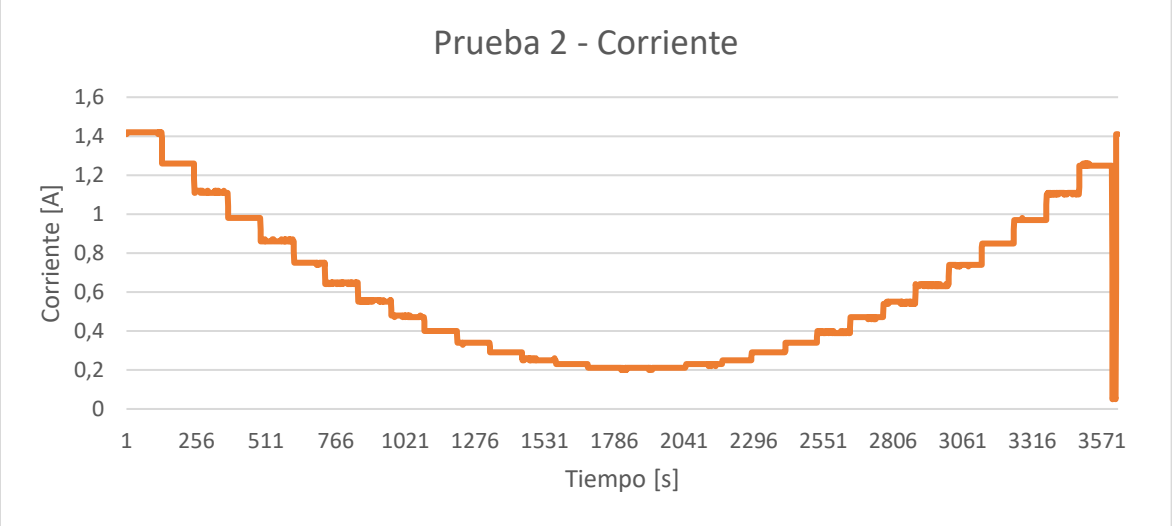


Figura 38. Gráfica de la corriente registrada en la segunda prueba.

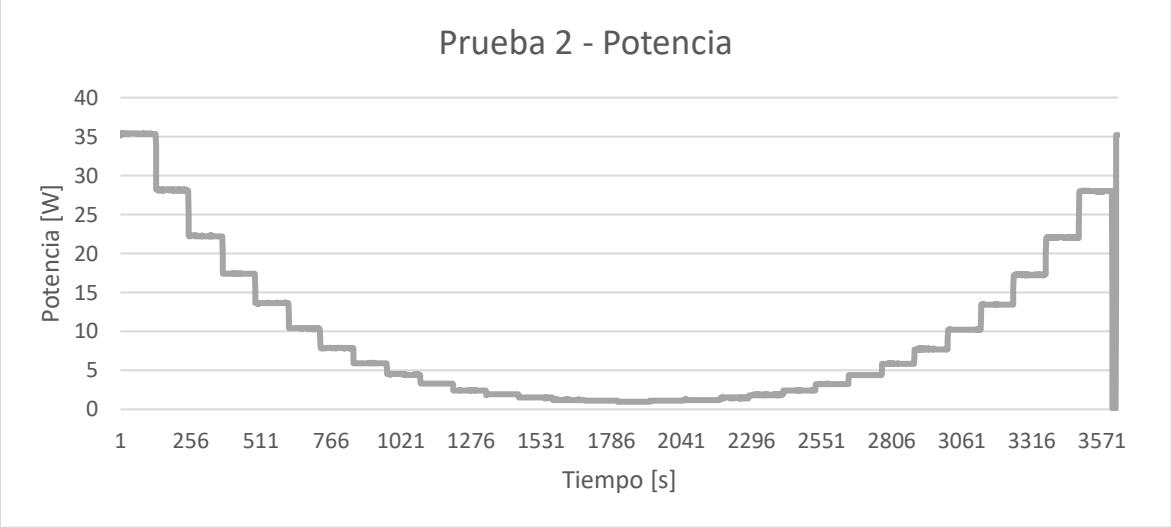


Figura 39. Gráfica de la potencia registrada en la segunda prueba.

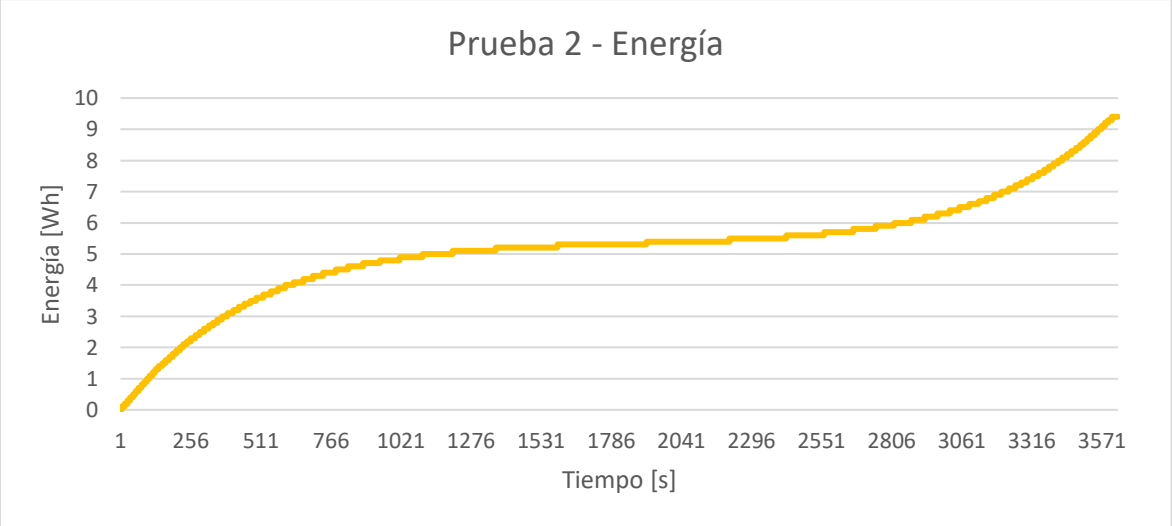


Figura 40. Gráfica de la energía registrada en la segunda prueba.

El tercer perfil es una función senoidal. Para esta función se buscó que las crestas y los valles coincidieran con minutos cerrados, aunque no se vio la necesidad de elegir alguno en especial. Lo que sí se consideró necesario, fue establecer las crestas en 15 [V] y los valles en 5 [V]. La función (3) es el resultado de estas especificaciones y se puede apreciar su gráfica en la Figura 41.

$$f(x) = 5sen\left(\frac{x}{4.456}\right) + 10 \quad (3)$$

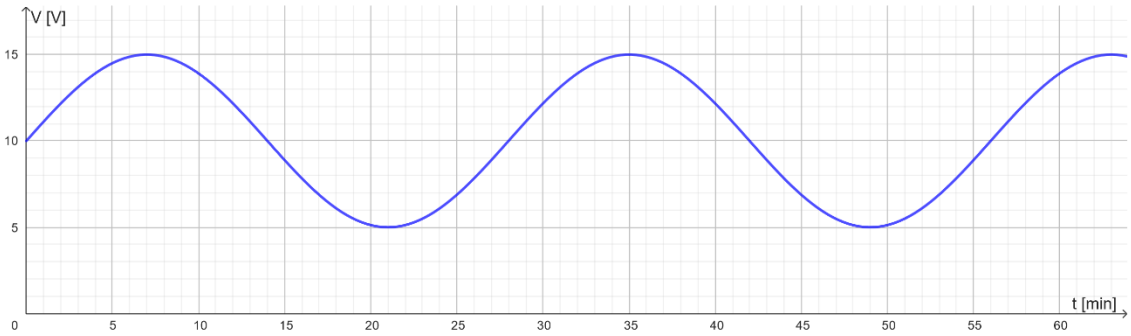


Figura 41. Gráfica correspondiente al perfil de la tercera prueba

De esta gráfica, se tomaron los datos en intervalos de 1 minuto. El redondeo de los datos fue mínimo, pues era necesario mantener cierta precisión para poder conservar la forma de la gráfica. Los datos resultantes se muestran en la siguiente tabla.

Tabla 3. Perfil de la tercera prueba.

Tiempo [min]	Voltaje [V]
0	10
1	11
2	12
3	13
4	14
5	14.5
6	14.9
7	15
8	14.9
9	14.5
10	14
11	13
12	12
13	11
14	10
15	9
16	8
17	7
18	6
19	5.5
20	5.1
21	5
22	5.1
23	5.5
24	6
25	7
26	8
27	9
28	10
29	11
30	12
31	13
32	14
33	14.5
34	14.9
35	15
36	14.9
37	14.5
38	14
39	13
40	12
41	11
42	10
43	9
44	8
45	7
46	6
47	5.5
48	5.1
49	5
50	5.1
51	5.5
52	6
53	7
54	8
55	9
56	10
57	11
58	12
59	13
60	14

De la prueba, resultan las siguientes gráficas mostradas en las Figuras 42 a la 45. Es posible observar una distorsión ligeramente mayor a lo usual en la primera

cresta; esta distorsión nuevamente se debe a un error humano al momento de cambiar el voltaje durante la prueba.

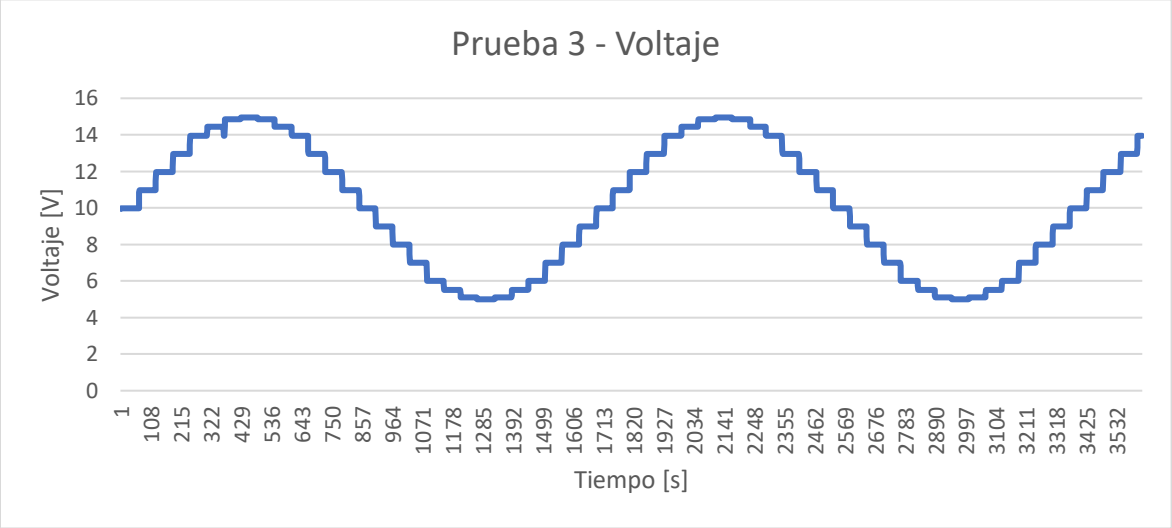


Figura 42. Gráfica del voltaje registrado en la tercera prueba.

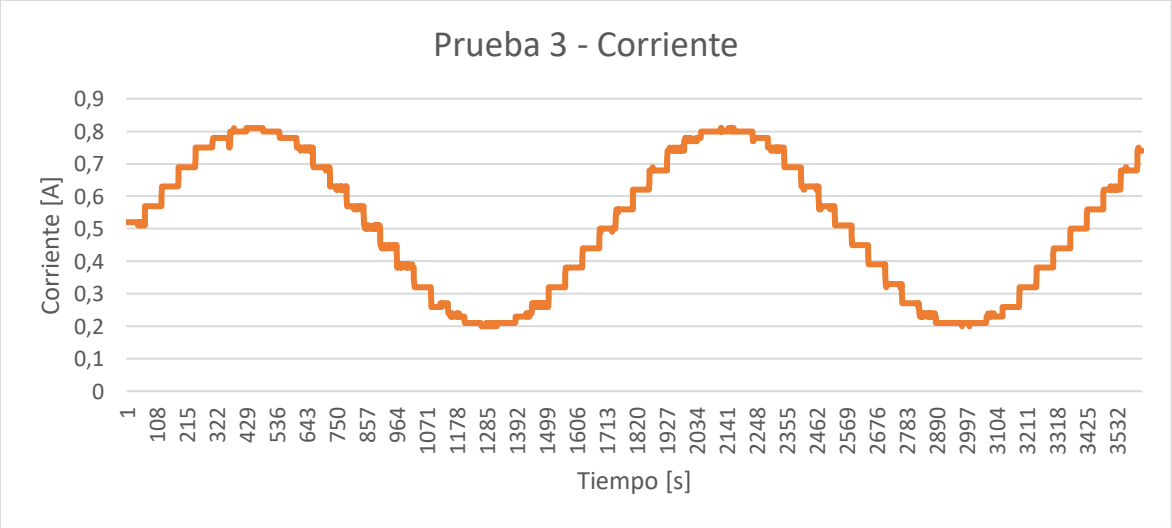


Figura 43. Gráfica de la corriente registrada en la tercera prueba.

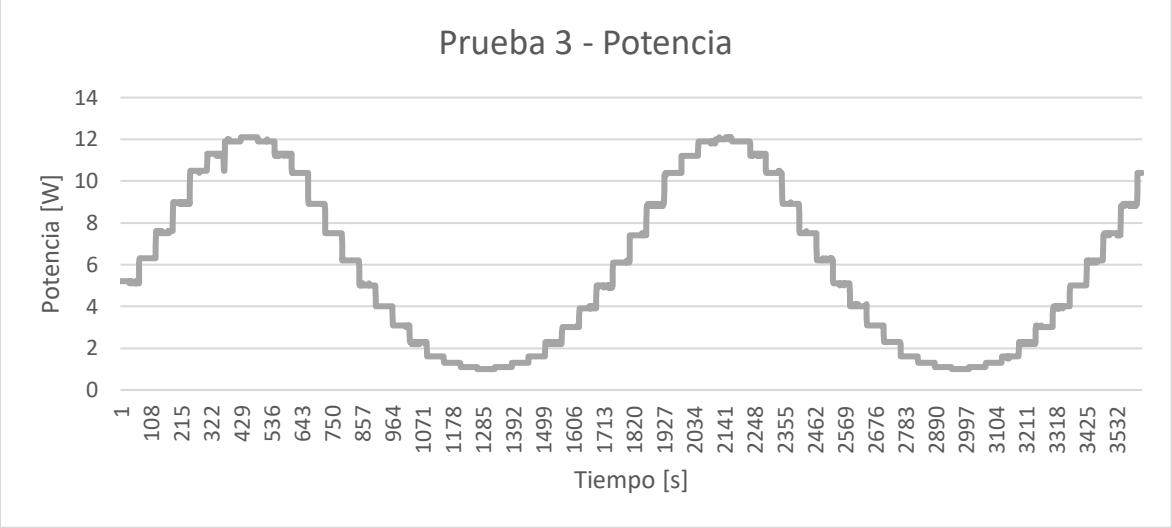


Figura 44. Gráfica de la potencia registrada en la tercera prueba.

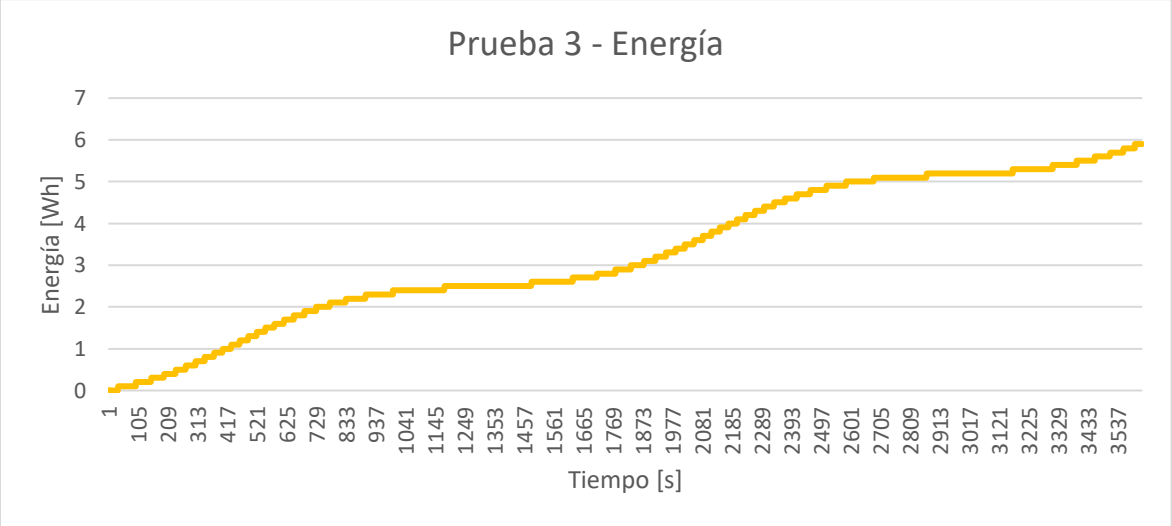


Figura 45. Gráfica de la energía registrada en la tercera prueba.

7 CONCLUSIONES Y TRABAJO FUTURO

Se diseñó y desarrolló una aplicación con base en un teléfono celular para recibir y almacenar las variables eléctricas de voltaje, corriente, potencia y energía que suministra un watthorímetro digital. Desde la aplicación es posible detectar dispositivos bluetooth para así poder encontrar y emparejarse con el watthorímetro deseado. Al momento de realizar una conexión exitosa, se muestra una actividad en donde se reflejan los datos recibidos. Esto se hace a través de medidores circulares acompañados con el número exacto de la medida, de forma que la información es de fácil interpretación para el usuario. A la par que los datos se reflejan en pantalla, esto se va almacenando en una base de datos desde la que es posible descargarlos para su posterior análisis. Todo el proceso de recepción de datos se realiza apoyándose en un servicio que permite que la tarea se siga realizando incluso si la aplicación pierde el foco.

La aplicación funciona adecuadamente, sin embargo, es recomendable implementar algunas modificaciones que podrían resultar útiles. Por ejemplo, de momento, la función para rotar la pantalla esta deshabilitada, ya que la recepción de datos se detiene cuando se hace el cambio. También se consideró hacer que la base de datos fuera un recurso en línea, de forma que asegurara la uniformidad de los datos entre usuarios. Además, esta aplicación no cuenta con un medio para consultar los datos dentro de la misma, por lo que tener una vista de la información almacenada en cada registro, e incluso su gráfica, sería conveniente.

8 REFERENCIAS

Belmonte Fernández, O., 2005. *Introducción al lenguaje de programación Java. Una guía básica.* s.l.:s.n.

Castillo Hernández, J., Caballero Ruiz, A. & Ruiz Huerta, L., 2020. Prototipo para el control electrónico de un motor BLDC usado en un vehículo eléctrico. *Pistas Educativas,* 42(136), pp. 317-336.

Castillo Hernández, J., Caballero Ruiz, A., Ruiz-Huerta, L. & De Gortari Briseño, J., 2019. Desarrollo del prototipo de un watthorímetro digital. *Pistas Educativas,* 41(134), p. 119.134.

Castillo Hernández, J., Castillo Martínez, M. A., Quintana Thierry, S. & Damián Zamacona, J. R., 2022. Aplicación para registrar el consumo eléctrico de corriente directa. *Congreso de Instrumentación SOMI XXXVI. Sociedad Mexicana de Instrumentación. Año 8, No 01, 1aSOMI36_43.*

Castillo Hernández, J., Kemper Valdever, N. C. & Sovero Ancheyta, G., 2013. Medición inalámbrica del consumo eléctrico para el ahorro de energía en edificios. *X-SEMETRO-2013, 10th International Congress on Electrical Metrology, Instituto Nacional de la Tecnología Industrial,* p. ID164.

Castillo Hernández, J., Marcos J., A., Bañuelos Saucedo, M. & Quintana Thierry, S., 2007. Implementación de un algoritmo de modulación vectorial en un FPGA para control de un motor de inducción trifásico.

de Gortari Briseño, J., 2019. *Sistema inalámbrico para la medición de energía de un auto eléctrico.* México. Cd. Mx.: Universidad Nacional Autónoma de México, Facultad de Ingeniaría.

Google, 2020. *Aspectos fundamentales de la aplicación.* [En línea] Available at: https://developer.android.com/guide/components/fundamentals?hl=es-419

Google, 2020. *Cómo interpretar el ciclo de vida de una actividad.* [En línea] Available at: https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es-419

Google, 2021. *Descripción general de los servicios.* [En línea] Available at: https://developer.android.com/guide/components/services?hl=es-419

Google, 2023. *Guía de arquitectura de apps.* [En línea] Available at: https://developer.android.com/topic/architecture?hl=es_419

Google, 2023. *Introducción a Android Studio.* [En línea] Available at: https://developer.android.com/studio/intro?hl=es-419

Google, 2023. *Permisos en Android.* [En línea] Available at: https://developer.android.com/guide/topics/permissions/overview?hl=es_419

Gradle Inc., 2022. *What is Gradle?.* [En línea] Available at: https://docs.gradle.org/current/userguide/what_is_gradle.html

Kai OS Technologies, 2021. *Our mission.* [En línea] Available at: https://www.kaiostech.com/company/

Kleczkowski, P., YALÇIN, S. & González, I., 2019. *CustomGauge.* [En línea] Available at: https://github.com/pkleczko/CustomGauge

Linux Foundation, 2012. *About.* [En línea] Available at: https://www.tizen.org/about

Mozilla Corporation, 2022. *Introducción a XML.* [En línea] Available at: https://developer.mozilla.org/es/docs/Web/XML/XML_introduction

Open Handset Alliance, s.f. *Android.* [En línea] Available at: http://www.openhandsetalliance.com/android_overview.html

Oracle Corporation, 2022. *¿Qué es la tecnología Java y por qué la necesito?.* [En línea]
Available at: https://www.java.com/es/download/help/whatis_java.html

Silberschatz, A., Galvin, P. B. & Gagne, G., 2004. *Fundamentos de sistemas operativos.* s.l.:Mc Graw Hill.

SQLite, 2023. *What Is SQLite?.* [En línea]
Available at: https://www.sqlite.org/index.html

Statcounter, 2023. *Operating System Market Share Worldwide Apr 2021 - Apr 2023.* [En línea]
Available at: https://gs.statcounter.com/os-market-share#monthly-202104-202304

TIOBE Software BV, 2023. *TIOBE Index for April 2023.* [En línea]
Available at: https://www.tiobe.com/tiobe-index/

Wolf , G., Ruiz, E., Borgero, F. & Meza, E., 2015. *Fundamentos de sistemas operativos.* México, Cd. Mx.: Universidad Nacional Autónoma de México, Instituto de Investigaciones Económicas: Facultad de Ingeniaeria.

9 APÉNDICE

9.1 Código

A continuación, se exponen las partes más relevantes del proyecto, sin embargo, para verlo en su totalidad, es posible acceder a él a través de: https://github.com/AlejandraCastillo/MedidorBluetooth

9.1.1 MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MainActivity";

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

        Objects.requireNonNull(getSupportActionBar()).setTitle(" ");

        BluetoothUtils bluetoothUtils = new BluetoothUtils(this, this);

        if (!bluetoothUtils.bluetoothAdapterExist()){
            Button connectButton = findViewById(R.id.b_conectar);
            connectButton.setVisibility(View.INVISIBLE);
            Button downloadButton = findViewById(R.id.b_descargas);
            downloadButton.setVisibility(View.INVISIBLE);
            TextView tvBienvenida = findViewById(R.id.tv_bienvenido);
            tvBienvenida.setVisibility(View.INVISIBLE);
            TextView tvNoBluetooth = findViewById(R.id.tv_no_bluetooth);
            tvNoBluetooth.setVisibility(View.VISIBLE);
        }

//        SQLiteActions actions = new SQLiteActions(this);
//        actions.borrarBD();
    }

    public void onClickConectar(View view) {
        Intent intent = new Intent(this, ConnectActivity.class);
        startActivity(intent);
    }

    public void onClickDescargar(View view) {
        Intent intent = new Intent(this, DownloadActivity.class);
        startActivity(intent);
    }

}
```

9.1.2 ConnectActivity.java

```
public class ConnectActivity extends AppCompatActivity {
    private static final String TAG = "ConnectActivity";

    private BluetoothUtils bluetoothUtils;

    private SwitchCompat swBluetooth;
    private RecyclerView rvBondedDevices;
    private ProgressBar progressBar;
    private ProgressBar pbBluetooth;
    private Button buttonBuscar;

    private boolean discoveryFlag = false;
    private boolean enableBTFlag = false;

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

Objects.requireNonNull(getSupportActionBar()).setTitle("Bluetooth");

        swBluetooth = findViewById(R.id.sw_enable_bt);
        progressBar = findViewById(R.id.progress_bar);
        pbBluetooth = findViewById(R.id.pb_buetooth);
        rvBondedDevices = findViewById(R.id.rv_bondedDevices);
        RecyclerView rvDiscoveryDevices =
findViewById(R.id.rv_discoveryDevices);
        buttonBuscar = findViewById(R.id.b_Busqueda);

        bluetoothUtils = new BluetoothUtils( this, this);

        LinearLayoutManager bondedDevicesManager = new
LinearLayoutManager(this);
        rvBondedDevices.setLayoutManager(bondedDevicesManager);
        rvBondedDevices.setAdapter(bluetoothUtils.adapterBondedDevices);

        LinearLayoutManager discoveryDevicesManager = new
LinearLayoutManager(this);
        rvDiscoveryDevices.setLayoutManager(discoveryDevicesManager);

rvDiscoveryDevices.setAdapter(bluetoothUtils.adapterDiscoveryDevices);

        //Verificar el estado del bluetooth
        if(bluetoothUtils.bluetoothAdapterIsEnable()){
            swBluetooth.setChecked(true);
            enableBTFlag=true;
            bluetoothUtils.getBondedDevices();
        }
```

```
        //Switch
        swBluetooth.setOnCheckedChangeListener((compoundButton,
isChecked) -> {
            if(isChecked) {
                pbBluetooth.setVisibility(View.VISIBLE);
                swBluetooth.setVisibility(View.INVISIBLE);
                bluetoothUtils.enableBluetoothAdapter();
            }
            else {
                pbBluetooth.setVisibility(View.VISIBLE);
                swBluetooth.setVisibility(View.INVISIBLE);
                bluetoothUtils.disableBluetoothAdapter();
            }
        });

        //Registrar el broadcast
        IntentFilter filter = new IntentFilter();
        filter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
        filter.addAction(BluetoothDevice.ACTION_FOUND);
        filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
        filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        registerReceiver(receiver, filter);
    }

    public final BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            switch (action){
                case BluetoothAdapter.ACTION_STATE_CHANGED:
                    bluetoothStateManager(intent);
                    break;
                case BluetoothAdapter.ACTION_DISCOVERY_STARTED:
                    Log.i(TAG, "onReceive: Iniciar busqueda");
                    discoveryFlag = true;
                    buttonBuscar.setText(R.string.cancelar_busqueda);
                    progressBar.setVisibility(View.VISIBLE);
                    break;
                case BluetoothDevice.ACTION_FOUND:
                    BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {

bluetoothUtils.checkPermission(Manifest.permission.BLUETOOTH_CONNECT);
                    }
                    Log.i(TAG, "onReceive: Dispositivo " +
device.getName() + " encontrado");
                    bluetoothUtils.addNewDevice(device);
                    break;
                case BluetoothAdapter.ACTION_DISCOVERY_FINISHED:
                    Log.i(TAG, "onReceive: Cancelar busqueda");
                    discoveryFlag = false;
```

```
                    progressBar.setVisibility(View.INVISIBLE);
                    buttonBuscar.setText(R.string.inciar_busqueda);
                    break;
            }

        }
    };

    private void bluetoothStateManager(Intent intent){
        int state = intent.getIntExtra(
                BluetoothAdapter.EXTRA_STATE,
                BluetoothAdapter.ERROR);

        switch (state) {
            case BluetoothAdapter.STATE_ON:
                swBluetooth.setChecked(true); //Si el estado del BT es
cambiado fuera de la app
                if(bluetoothUtils.getBondedDevices())

rvBondedDevices.setAdapter(bluetoothUtils.adapterBondedDevices);
                pbBluetooth.setVisibility(View.INVISIBLE);
                swBluetooth.setVisibility(View.VISIBLE);
                enableBTFlag=true;
                break;
            case BluetoothAdapter.STATE_OFF:
                swBluetooth.setChecked(false);
                bluetoothUtils.adapterBondedDevices.clear();
                bluetoothUtils.adapterDiscoveryDevices.clear();
                pbBluetooth.setVisibility(View.INVISIBLE);
                swBluetooth.setVisibility(View.VISIBLE);
                enableBTFlag=false;
                break;
        }

    }

    public void onClickBuscar(View view) {
        if(enableBTFlag){
            String permiso = Manifest.permission.ACCESS_FINE_LOCATION;
            if (ContextCompat.checkSelfPermission(this, permiso)
                    != PackageManager.PERMISSION_GRANTED) {
                requestPermissionLauncher.launch(permiso);
            }
            else
                bluetoothUtils.discovery(discoveryFlag);
        }
        else{
            Toast.makeText(this, "El bluetooth está apagado",
Toast.LENGTH_SHORT).show();
        }
    }

    private final ActivityResultLauncher<String>
```

```
requestPermissionLauncher = registerForActivityResult(
            new ActivityResultContracts.RequestPermission(),
            new ActivityResultCallback<Boolean>() {
                @Override
                public void onActivityResult(Boolean result) {
                    if (result) {
                        bluetoothUtils.discovery(discoveryFlag);
                    } else {
                        Log.i(TAG, "onActivityResult: Permiso denegado");
                    }
                }
            }
    );

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.i(TAG, "onDestroy()");
        //Se elimina el resgistro del broadcast al destruir el Activity
        unregisterReceiver(receiver);
    }

}
```

9.1.3 MessageReceiverActivity.java

```
public class MessageReceiverActivity extends AppCompatActivity {
    private static final String TAG = "MessageReceiver";

    public static TextView tvVoltage;
    public static TextView tvCorriente;
    public static TextView tvPotencia;
    public static TextView tvEnergia;

    public static CustomGauge gaugeVoltaje;
    public static CustomGauge gaugeCorriente;
    public static CustomGauge gaugePotencia;
    public static CustomGauge gaugeEnergia;

    Intent serviceIntent;
    boolean isBounded;
    MyService myService;

    @SuppressLint("SetTextI18n")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_message_receiver);

        Log.d(TAG, "onCreate");

        tvVoltage = findViewById(R.id.tv_voltaje);
```

```
        tvCorriente = findViewById(R.id.tv_corriente);
        tvPotencia = findViewById(R.id.tv_potencia);
        tvEnergia = findViewById(R.id.tv_energia);

        gaugeVoltaje = findViewById(R.id.gauge_voltaje);
        gaugeCorriente = findViewById(R.id.gauge_corriente);
        gaugePotencia = findViewById(R.id.gauge_potencia);
        gaugeEnergia = findViewById(R.id.gauge_energia);

        SQLiteActions actions = new SQLiteActions(this);
        int registro = actions.getLastRegister() + 1;
        Objects.requireNonNull(getSupportActionBar()).setTitle("Registro:
" + registro + "   Fecha: " + actions.getDate("dd/MM/yy"));

        myService = new MyService();
        serviceIntent = new Intent(this, MyService.class);
        startService(serviceIntent);
        bindService(serviceIntent, connection, BIND_AUTO_CREATE);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
        unbindService(connection);
        stopService(serviceIntent);
    }

    private final ServiceConnection connection = new ServiceConnection()
{
        @Override
        public void onServiceConnected(ComponentName componentName,
IBinder iBinder) {
            MyService.MyBinder binder = (MyService.MyBinder) iBinder;
            myService = binder.getService();
            isBounded = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            isBounded = false;
        }
    };

}
```

9.1.4 DownloadActivity.java

```
public class DownloadActivity extends AppCompatActivity implements
OnClickListenerDownload {

    private static final String TAG = "DownloadActivity";
```

```
    private SQLiteActions actions;

    ActivityResultLauncher<Intent> descargaLauncher;

//    private static final String REGISTRO_ID = "registro_id";
    private int mRegistroID;

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

Objects.requireNonNull(getSupportActionBar()).setTitle("Descargas");

        RecyclerView recyclerView = findViewById(R.id.rv_descargas);

        actions = new SQLiteActions(this);
        ArrayList<TablaRegistro> listTablaRegistro;

        listTablaRegistro = actions.readTablaRegistro();

        LinearLayoutManager manager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(manager);
        RecycleViewDownloadAdapter downloadAdapter = new
RecycleViewDownloadAdapter(listTablaRegistro, this);
        recyclerView.setAdapter(downloadAdapter);

        descargaLauncher = registerForActivityResult(
                new ActivityResultContracts.StartActivityForResult(),
                result -> {
                    if(result.getResultCode() == Activity.RESULT_OK){
                        Intent data = result.getData();
                        if(data != null && data.getData() != null){
                            Log.i(TAG, "onCreate: registro " +
mRegistroID);
                            actions.createDocument(data.getData(),
mRegistroID);
                        }
                    }
                }
        );
    }

    @Override
    public void onClick(TablaRegistro registro) {
        Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("text/CSV");
        int registro_id = registro.getRegistroID();
        String nombre = actions.getFechaRegistro(registro_id) + "_reg" +
registro_id + ".CSV";
        intent.putExtra(Intent.EXTRA_TITLE, nombre);
```

```
            mRegistroID = registro_id;

            descargaLauncher.launch(intent);
        }

}
```

9.1.5 MyService.java

```
public class MyService extends Service {
    private static final String TAG = "MyService";

    private static final int NOTIFICATION_ID = 1;
    public static final int MESSAGE_READ = 2;
    public static final int CONNECTION_LOST = 3;
    private static final String CHANNEL_ID = "100";

    private final IBinder myBinder = new MyBinder();
    private boolean isBind = false;

    BluetoothSocket socket;
    private SQLiteActions actions;
    private int registro;

    ExecutorService executorService;

    public MyService() {
    }

    @Override
    public void onCreate() {
        super.onCreate();
        Log.d(TAG, "onCreate");

        socket = ShareSocket.getSocket();
        actions = new SQLiteActions(this);
        registro = actions.addNewRegister();

        startForeground(NOTIFICATION_ID, showNotification("Contenido "));
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(TAG, "onStartCommand");
        doTask();
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
        ShareSocket.closeSocket();
```

```
        executorService.shutdown();
    }

    @Override
    public IBinder onBind(Intent intent) {
        Log.d(TAG, "onBind");
        isBind = true;
        return myBinder;
    }

    @Override
    public boolean onUnbind(Intent intent) {
        Log.d(TAG, "onUnbind");
        isBind = false;
        return super.onUnbind(intent);
    }

    public class MyBinder extends Binder {
        public MyService getService(){
            return MyService.this;
        }
    }

    private Notification showNotification(String content){
        //Solo Google Oreo o versiones superiores
        ((NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE))
                .createNotificationChannel(
                        new NotificationChannel(
                                CHANNEL_ID,
                                "ForegroundService",
                                NotificationManager.IMPORTANCE_HIGH
                        )
                );

        return new NotificationCompat.Builder(this, CHANNEL_ID)
                .setContentTitle("Registro " + registro)
                .setContentText(content)
                .setOnlyAlertOnce(true)
                .setOngoing(true)
                .setSmallIcon(R.drawable.ic_arrow_24)
                .build();
    }

    private void updateNotification(String data){
        Notification notification = showNotification(data);
        NotificationManager notificationManager =
                (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
        notificationManager.notify(NOTIFICATION_ID, notification);
    }

    private void doTask(){
```

```
        executorService = Executors.newSingleThreadExecutor();

        AtomicReference<String> line = new AtomicReference<>();
        String regex = "1?2?3::\\d*(-\\d*\\.\\d*){4}::456";

        InputStream tmpInStream = null;
        try{
            tmpInStream = socket.getInputStream();
        } catch (IOException e){
            Log.e(TAG, "Error al crear el input stream", e);
        }
        InputStream mmInStream = tmpInStream;
        BufferedReader buffer = new BufferedReader(new
InputStreamReader(mmInStream));

        executorService.execute(() -> {
                while (socket.isConnected()){
                    try {
                        line.set(buffer.readLine().replaceAll("\\s",
""));
                        Log.i(TAG, "doTask: " + line);
                        if(line.get().matches(regex)){
                            handler.obtainMessage(MESSAGE_READ, line)
                                    .sendToTarget();
                        }
                        Log.d(TAG, "doTask: " + line);
                    } catch (IOException e){
                        handler.obtainMessage(CONNECTION_LOST)
                                .sendToTarget();
                        break;
                    }
                }
            }
        );
    }

    private final Handler handler = new Handler(Looper.myLooper()) {
        @Override
        @SuppressLint("SetTextI18n")
        public void handleMessage(@NotNull Message msg){
            switch (msg.what){
                case MESSAGE_READ:
                    String readMessage = msg.obj.toString();

                    int index = readMessage.indexOf("::");
                    String aux = readMessage.substring(index + 2);
                    index = aux.indexOf("::");
                    aux = aux.substring(0, index);

                    TablaDatos row = new TablaDatos();

                    //Tiempo
                    index = aux.indexOf('-');
```

```
                    long t = Long.parseLong(aux.substring(0, index));
                    row.setTiempo(t);
                    aux = aux.substring(index + 1);

                    //Voltage
                    index = aux.indexOf('-');
                    double v = Double.parseDouble(aux.substring(0,
index));
                    row.setVoltaje(v);
                    aux = aux.substring(index + 1);

                    //Corriente
                    index = aux.indexOf('-');
                    double c = Double.parseDouble(aux.substring(0,
index));
                    row.setCorriente(c);
                    aux = aux.substring(index + 1);

                    //Potencia
                    index = aux.indexOf('-');
                    double p = Double.parseDouble(aux.substring(0,
index));
                    row.setPotencia(p);

                    //Energia
                    double e = Double.parseDouble(aux.substring(index +
1));


                    row.setEnergia(e);
                    row.setRegistroID(registro);
                    actions.addnewDataRow(row);

                    updateNotification(row.printRow(TAG));

                if(isBind){
                    MessageReceiverActivity.gaugeVoltaje.setValue(v);
                    MessageReceiverActivity.gaugeCorriente.setValue(c);
                    MessageReceiverActivity.gaugePotencia.setValue(p);
                    MessageReceiverActivity.gaugeEnergia.setValue(e);

                    DecimalFormat formato = new DecimalFormat("00.00");

MessageReceiverActivity.tvVoltage.setText(formato.format(v) + " V");

                    formato = new DecimalFormat("0.00");

MessageReceiverActivity.tvCorriente.setText(formato.format(c) + " A");

MessageReceiverActivity.tvEnergia.setText(formato.format(e) + " Wh");

                    formato = new DecimalFormat("000.00");

MessageReceiverActivity.tvPotencia.setText(formato.format(p) + " W");
```

```
                }

                    break;

                case CONNECTION_LOST:
                    Log.i(TAG, "Conexión perdida");
                    Toast.makeText(getBaseContext(), "Conexión perdida",
Toast.LENGTH_SHORT).show();
                    if(isBind){
                        MessageReceiverActivity.gaugeVoltaje.setValue(0);

MessageReceiverActivity.gaugeCorriente.setValue(0);

MessageReceiverActivity.gaugePotencia.setValue(0);
                        MessageReceiverActivity.gaugeEnergia.setValue(0);

                        MessageReceiverActivity.tvVoltage.setText("--.--
");
                        MessageReceiverActivity.tvCorriente.setText("--.-
-");
                        MessageReceiverActivity.tvEnergia.setText("--.--
");
                        MessageReceiverActivity.tvPotencia.setText("--.--
");
                    }
            }

        }
    };

}
```