



Universidad
Nacional
de Córdoba



Ingeniería en Computación

Informe Final de Proyecto Integrador

Captura de Señales en Dispositivos Móviles para la Detección de Contactos entre Personas

Autor

Ramiro Fernando DETKE

DNI 38122152

ramiro.fer.det@gmail.com

☎ 0351-574-5305

Director

Dr. Jorge M. FINOCHIETTO

2020

Dedicatoria

Para Mami, Papi, Abi y Enana

Agradecimientos

A mis padres, que fueron y son mi ejemplo a seguir. A mi hermana que es y será siempre mi cómplice y una gran amiga. A mi Abi que siempre me cuidó y me contuvo. Las personas que estuvieron apoyándome en cada paso que di durante toda mi vida. Gracias porque sin ellos a mi lado acompañándome con amor y comprensión no habría llegado a cumplir este sueño.

A todo el equipo del Laboratorio de Comunicaciones Digitales, quienes estuvieron presentes con buena voluntad y predisposición contribuyendo y compartiendo gran parte de mi formación, incluyendo este trabajo.

A mis amigos y compañeros de facultad, con los que viví infinidad de momentos.

A mis profesores, que compartieron sus conocimientos, experiencias y que guiaron mi formación durante toda la carrera, y en especial a dos grandes personas que hicieron la diferencia: Miguel Ferreras gracias por ser mi inspiración y despertar en mí la vocación de ser Ingeniero, y Jorge Finochietto quien además de ser el director de este proyecto me ha mostrado que para ser un buen profesional hay que combinar los conocimientos con la pasión por lo que uno hace confiando que en este camino no estamos solos.

Índice

| | |
|---|----|
| Índice de Tablas | 8 |
| Índice de Figuras | 9 |
| Resumen | 12 |
| Palabras Clave | 12 |
| Área Temática y Asignaturas | 13 |
| Siglas y Convenciones | 14 |
| Capítulo 1: Introducción | 15 |
| Presentación del Problema | 16 |
| Objetivos | 17 |
| Metodología de Desarrollo | 18 |
| Estructura del documento | 18 |
| Capítulo 2: Marco Teórico | 19 |
| Detección y Trazabilidad de Contactos | 19 |
| Dispositivos de Comunicación | 19 |
| Wi-Fi | 19 |
| Bluetooth | 20 |
| Sensores | 22 |
| Aplicación Android | 23 |
| Permisos | 24 |
| Actividades | 25 |
| Servicios | 26 |
| Notificaciones | 26 |
| Intents | 27 |
| Broadcasts | 28 |
| Patrones de diseño | 28 |
| Observer | 29 |
| Singleton | 29 |

| | |
|--|----|
| Data Access Object | 30 |
| Repository | 30 |
| Arquitectura de un sistema | 31 |
| Arquitectura Model-View-ViewModel | 31 |
| Arquitectura sugerida para Aplicaciones Android | 32 |
| Capítulo 3: Desarrollo del Proyecto | 33 |
| Riesgos | 34 |
| Identificación y Análisis | 34 |
| Planificación | 34 |
| Definición de Requerimientos | 35 |
| Detalle en alto nivel de funcionalidades | 35 |
| Detalle de requerimientos funcionales | 36 |
| Detalle de requerimientos no funcionales | 39 |
| Etapas del Desarrollo | 39 |
| Selección de herramientas | 40 |
| Desarrollo de la Aplicación | 41 |
| Arquitectura | 41 |
| Modelo y base de datos | 42 |
| Descripción del proceso de ejecución de un Experimento | 44 |
| Concepto de Ventana | 45 |
| Módulos de recolección | 46 |
| Módulo de transmisión (Baliza BLE) | 52 |
| Módulos de exportación de datos | 53 |
| Interfaces gráficas de usuario | 54 |
| Pruebas | 58 |
| Capítulo 4: Resultados | 61 |
| Bluetooth Low Energy | 64 |
| Wi-Fi | 68 |

| | |
|---|----|
| Sensores | 70 |
| Capítulo 5: Conclusiones | 72 |
| Trabajos Futuros | 73 |
| Anexo A: Detalle de Pruebas | 75 |
| Anexo B: Manual para el usuario | 83 |
| Anexo C: Algoritmos para graficar datos de prueba | 89 |
| Bluetooth Low Energy | 89 |
| Wi-Fi | 91 |
| Sensores | 92 |
| Referencias | 93 |

Índice de Tablas

| | |
|--|----|
| Tabla 1: Riesgos identificados en este proyecto. Fuente: Elaboración propia..... | 34 |
| Tabla 2: Planificación de medidas para cada peligro identificado. Fuente: Elaboración propia | 35 |
| Tabla 3: Detalle de funcionalidades de la aplicación. Fuente: Elaboración propia | 35 |
| Tabla 4: Detalle de requerimientos funcionales. Fuente: Elaboración propia | 36 |
| Tabla 5: Detalle de requerimientos no funcionales. Fuente: Elaboración propia | 39 |
| Tabla 6: Referencia de colores para clases de pruebas. Fuente: Elaboración propia | 58 |
| Tabla 7: Lista de clases con descripciones de pruebas implementadas. Fuente: Elaboración propia .. | 59 |
| Tabla 8: Configuración de experimento | 61 |
| Tabla 9: Descripción de pruebas para BluetoothLeAdvertiseScheduler. Fuente: Elaboración propia . | 75 |
| Tabla 10: Descripción de pruebas para BluetoothCsvFileWriterTest. Fuente: Elaboración propia..... | 75 |
| Tabla 11: Descripción de pruebas para BluetoothLeCsvFileWriterTest. Fuente: Elaboración propia . | 75 |
| Tabla 12: Descripción de pruebas para SensorCsvFileWriterTest. Fuente: Elaboración propia | 76 |
| Tabla 13: Descripción de pruebas para WifiCsvFileWriterTest. Fuente: Elaboración propia | 76 |
| Tabla 14: Descripción de pruebas para JsonFileWriterTest. Fuente: Elaboración propia | 76 |
| Tabla 15: Descripción de pruebas para ExperimentRepresentationTest. Fuente: Elaboración propia . | 77 |
| Tabla 16: Descripción de pruebas para BluetoothLeRepositoryTest. Fuente: Elaboración propia..... | 77 |
| Tabla 17: Descripción de pruebas para BluetoothRepositoryTest. Fuente: Elaboración propia..... | 77 |
| Tabla 18: Descripción de pruebas para DeviceRepositoryTest. Fuente: Elaboración propia..... | 78 |
| Tabla 19: Descripción de pruebas para ConfigurationRepositoryTest. Fuente: Elaboración propia ... | 78 |
| Tabla 20: Descripción de pruebas para SensorRepositoryTest. Fuente: Elaboración propia | 79 |
| Tabla 21: Descripción de pruebas para SourceTypeRepositoryTest. Fuente: Elaboración propia | 79 |
| Tabla 22: Descripción de pruebas para WifiRepositoryTest. Fuente: Elaboración propia | 79 |
| Tabla 23: Descripción de pruebas para WindowRepositoryTest. Fuente: Elaboración propia | 80 |
| Tabla 24: Descripción de pruebas para ExperimentRepositoryTest. Fuente: Elaboración propia | 80 |
| Tabla 25: Descripción de pruebas para ShareToolsTest. Fuente: Elaboración propia | 81 |
| Tabla 26: Descripción de pruebas para RunRepositoryTest. Fuente: Elaboración propia | 81 |
| Tabla 27: Descripción de pruebas para BluetoothScanSchedulerTest. Fuente: Elaboración propia | 82 |
| Tabla 28: Descripción de pruebas para SensorsScanSchedulerTest. Fuente: Elaboración propia | 82 |
| Tabla 29: Descripción de pruebas para WifiScanSchedulerTest. Fuente: Elaboración propia..... | 82 |
| Tabla 30: Descripción de pruebas para BluetoothLeScanSchedulerTest. Fuente: Elaboración propia | 82 |

Índice de Figuras

| | |
|---|----|
| Figura 1: Formato de un Beacon Frame Wi-Fi. | 20 |
| Figura 2: Formato de un paquete Bluetooth. | 20 |
| Figura 3: Formato de un Frame de Advertising BLE. | 21 |
| Figura 4: Ejemplo de declaración de permisos en el manifiesto de la aplicación. Fuente: Elaboración propia | 24 |
| Figura 5: Ejemplos de diálogo para que el usuario otorgue un permiso. Fuente: Elaboración propia . | 24 |
| Figura 6: Ejemplo de declaración de una actividad en el manifiesto de la aplicación. Fuente: Elaboración propia..... | 25 |
| Figura 7: Ciclo de vida simplificado de una actividad..... | 26 |
| Figura 8: Ícono de notificación al lado izquierdo de la barra de estado. Fuente: Elaboración propia .. | 27 |
| Figura 9: Detalles de notificaciones en el panel. Fuente: Elaboración propia | 27 |
| Figura 10: Estructura del patrón Observer. Fuente: Elaboración propia..... | 29 |
| Figura 11: Actor del patrón Singleton. Fuente: Elaboración propia..... | 30 |
| Figura 12: Estructura del patrón Data Access Object. Fuente: Elaboración propia | 30 |
| Figura 13: Representación del patrón Repository. Fuente: Elaboración propia | 31 |
| Figura 14: Interacción de los componentes de la arquitectura MVVM. Fuente: Adaptado de [22]..... | 32 |
| Figura 15: Arquitectura sugerida en la documentación de Android. | 32 |
| Figura 16: Proceso de desarrollo del proyecto. Fuente: Elaboración propia | 33 |
| Figura 17: Arquitectura de alto nivel del sistema. Fuente: Elaboración propia | 41 |
| Figura 18: Arquitectura con detalle de la aplicación. Fuente: Elaboración propia | 42 |
| Figura 19: Esquema Entidad-Relación para el modelo de datos de la Aplicación. Fuente: Elaboración propia | 43 |
| Figura 20: Diagrama de Tablas implementado en la base de datos relacional. Fuente: Elaboración propia | 44 |
| Figura 21: Diagrama de secuencia en alto nivel del comportamiento esperado para la recolección de datos. Fuente: Elaboración propia | 45 |
| Figura 22: Representación del concepto de una ventana. Fuente: Elaboración propia..... | 46 |
| Figura 23: Diagrama de clases simplificado para la recolección de datos. Fuente: Elaboración propia | 47 |
| Figura 24: Diagrama de clases simplificado de los recolectores Wi-Fi y Bluetooth. Fuente: Elaboración propia..... | 48 |
| Figura 25: Comportamiento de recolectores que implementan la interfaz BroadcastReceiver. Fuente: Elaboración propia..... | 49 |

| | |
|--|----|
| Figura 26: Diagrama de clases simplificado del recolector Bluetooth Low Energy. Fuente: Elaboración propia..... | 50 |
| Figura 27: Comportamiento del recolector Bluetooth Low Energy. Fuente: Elaboración propia | 50 |
| Figura 28: Diagrama de clases simplificado del recolector de Sensores. Fuente: Elaboración propia..... | 51 |
| Figura 29: Comportamiento del recolector de Sensores. Fuente: Elaboración propia..... | 52 |
| Figura 30: Diagrama de clases del módulo para transmitir en modo Baliza Bluetooth Low Energy. Fuente: Elaboración propia..... | 53 |
| Figura 31: Diagrama de clases de módulos para escritura de archivos CSV y JSON. Fuente: Elaboración propia..... | 54 |
| Figura 32: Estructura del módulo para manejo de códigos QR. Fuente: Elaboración propia | 54 |
| Figura 33: Casos de uso de la aplicación para dispositivos móviles. Fuente: Elaboración propia | 55 |
| Figura 34: Flujos de usuario para interacción con la interfaz gráfica de la aplicación. Fuente: Elaboración propia..... | 55 |
| Figura 35: Secuencia de vistas de la aplicación. Fuente: Elaboración propia | 57 |
| Figura 36: Captura del último reporte de corrida de pruebas. Fuente: Elaboración propia | 60 |
| Figura 37: Captura del último reporte de cobertura de código. Fuente: Elaboración propia | 60 |
| Figura 38: Captura de pantalla de configuración de un experimento | 61 |
| Figura 39: Captura del archivo de configuración de experimento. Fuente: Elaboración propia..... | 62 |
| Figura 40: Captura de pantalla para el detalle del experimento configurado | 63 |
| Figura 41: Captura de pantalla para detalles de la corrida 1 para cada dispositivo (E6 y G7) | 63 |
| Figura 42: RSSI de las señales capturadas por el dispositivo E6. Fuente: Elaboración propia | 65 |
| Figura 43: RSSI de las señales capturadas por el dispositivo G7. Fuente: Elaboración propia..... | 66 |
| Figura 44: Histograma de RSSI capturado por E6 para corridas a 1m y a 0,4m. Fuente: Elaboración propia | 67 |
| Figura 45: Histograma de RSSI capturado por G7 para corridas a 1m y a 0,4m. Fuente: Elaboración propia | 67 |
| Figura 46: Cantidad de detecciones de cada red para cada dispositivo con 0,4m de separación entre ellos. Fuente: Elaboración propia..... | 69 |
| Figura 47: Cantidad de detecciones de cada red para cada dispositivo con 1m de separación entre ellos. Fuente: Elaboración propia | 69 |
| Figura 48: Nivel de luminosidad para cada dispositivo (separación de 1m). Fuente: Elaboración propia | 70 |
| Figura 49: Nivel de luminosidad para cada dispositivo (separación de 0,4m). Fuente: Elaboración propia | 71 |
| Figura 50: Aceleración en cada eje para cada dispositivo. Fuente: Elaboración propia | 71 |
| Figura 51: Ícono de la aplicación en el menú. Fuente: Elaboración propia | 83 |

| | |
|---|----|
| Figura 52: Recorte superior de pantalla principal Fuente: Elaboración propia | 83 |
| Figura 53: Pantalla de detalle del dispositivo Fuente: Elaboración propia | 84 |
| Figura 54: Recorte inferior de pantalla principal Fuente: Elaboración propia | 84 |
| Figura 55: Secuencia de imágenes para la creación de un experimento Fuente: Elaboración propia .. | 85 |
| Figura 56: Secuencia de imágenes para el escaneo de un experimento Fuente: Elaboración propia ... | 85 |
| Figura 57: Secuencia de imágenes para el detalle de un experimento Fuente: Elaboración propia | 86 |
| Figura 58: Pantalla para la programación de una nueva corrida. Fuente: Elaboración propia | 87 |
| Figura 59: Pantalla de detalle de una corrida Fuente: Elaboración propia | 88 |

Resumen

En el contexto de la pandemia del COVID-19 la detección del contacto estrecho entre las personas resulta crucial para identificar y romper la cadena de transmisión del virus. La Organización Mundial de la Salud (OMS) define como contacto estrecho a aquellas personas que se encuentren a menos de 2 metros de distancia por al menos 15 minutos. Dado que la mayoría de las personas suele llevar consigo un dispositivo móvil (en general, un teléfono celular) es posible utilizar éstos para la detección de contactos y de esta forma poder registrar de manera sistemática y objetiva dicha información.

En este trabajo se presenta el diseño e implementación de una aplicación para dispositivos móviles con sistema operativo Android para recolectar datos de diferentes periféricos de comunicación y de sensores, con la finalidad de poder procesarlos y analizarlos para determinar contactos entre las personas que poseen estos dispositivos. En el desarrollo se plantean cuatro grandes actividades: la primera es el diseño e implementación de un modelo de datos que represente a la información del dominio del problema, en segunda instancia el diseño e implementación de los diferentes módulos de recolección, seguidamente en tercer lugar se desarrolla un módulo de transmisión bajo el protocolo Bluetooth Low Energy y finalmente se diseñan e implementan módulos para exportar los datos en formatos procesables. A lo largo de todo el proceso de desarrollo se realizan verificaciones de las funcionalidades con pruebas unitarias y de integración automatizadas.

Se utilizaron herramientas provistas por Google para el desarrollo de aplicaciones móviles como por ejemplo Android Studio, la documentación del sistema operativo Android disponible de manera digital en Internet entre otros.

Al final del proceso de desarrollo se realizó una prueba a modo de demostración del uso de la aplicación, con la cual se evidencia la utilidad de la misma para el procesamiento y análisis de datos. Finalmente se concluye que el desarrollo de una aplicación móvil de estas características permite tener control sobre los datos que se pueden recolectar como así también sobre los mecanismos y tiempos de recolección. Además, los resultados muestran que los datos obtenidos son procesables con algoritmos implementados en cualquier lenguaje y sin la necesidad de librerías adicionales.

Palabras Clave

Detección de contactos, sensores, Wi-Fi, Bluetooth, BLE - *Bluetooth Low Energy*

Área Temática y Asignaturas

Área Temática: Comunicaciones, Aplicaciones móviles

Asignaturas: Comunicaciones de Datos, Redes de computadoras, Ingeniería de software, Bases de datos

Siglas y Convenciones

| | |
|-----------------|--|
| COVID-19 | Virus del SARS-CoV-2 |
| Wi-Fi | Wireless-Fidelity |
| IEEE | Institute of Electrical and Electronics Engineers |
| BSS | Basic Service Set |
| GAP | Generic Access Profile |
| GATT | Generic Attribute Profile |
| DAO | Data Access Object |
| DTO | Data Transfer Object |
| MVVM | Model-View-ViewModel |
| RF | Requerimiento Funcional |
| RNF | Requerimiento No Funcional |
| QR | Quick Response |
| JSON | JavaScript Object Notation |
| CSV | Comma-Separated Value |
| GSON | Librería para mapeo de Objetos Java a notación en formato JSON |
| BLE | Bluetooth Low Energy |
| SQL | Structured Query Language |
| JaCoCo | Java Code Coverage |
| JUnit | Java Unitary Tests |

Capítulo 1: Introducción

Dado el contexto producido por la pandemia del COVID-19 y aprovechando que actualmente la mayoría de las personas disponen de un dispositivo móvil que llevan consigo la mayor parte del tiempo, cobra interés la idea de poder determinar los contactos estrechos (Definidos por la Organización Mundial de la Salud como los encuentros entre personas con una separación de menos de 2 metros de distancia y por al menos 15 minutos) a través del análisis de las señales que captan estos dispositivos.

Al día de hoy, se desarrollan diferentes proyectos en esta área por parte de diferentes países, cada una con un enfoque ligeramente diferente pero todas con el mismo objetivo: *notificar la exposición al virus*. Algunas de las propuestas cuentan con protocolos y aplicaciones como:

- BlueTrace [49]: Es un protocolo en el cual se realizan intercambios de mensajes de descubrimiento BLE periódicamente. Un dispositivo central descubre dispositivos periféricos. La central solicita las características del periférico y este le responde con un mensaje que contiene un identificador temporal, la central almacena este identificador y responde con un mensaje que en el contenido porta su identificador junto con el indicador de potencia de señal (RSSI) del mensaje recibido desde el periférico, esto es así para que el dispositivo periférico conozca con que potencia recibió el mensaje la central. Cuando se confirma que un paciente dio positivo a la prueba de COVID-19, se propagan los identificadores que transmitió su dispositivo para que cada teléfono determine localmente si estuvo expuesto al virus debido a un contacto con el paciente infectado.
- DP-3T [50]: Similar al protocolo anterior, también se realizan intercambios de identificadores temporales y se registran localmente aquellos captados por los escaneos. Este protocolo se basa en la interfaz de programación de aplicaciones (*Application Programming Interface*, API) que propusieron Google y Apple para Notificación de Exposición [2]. Como explican en su documentación [5] utilizan un puntaje como nivel de exposición, el cual se calcula en base a la cantidad de contactos que se tuvo con dispositivos pertenecientes a pacientes infectados. Si este puntaje está por encima de un umbral se notifica al usuario y se le recomienda que se haga una prueba y realice aislamiento.

Por otro lado, como parte de una propuesta para desarrollar una solución en este campo, presentada desde el Laboratorio de Comunicaciones Digitales de esta unidad académica y, lugar donde se desarrolló el presente trabajo, es necesario recolectar información de las señales que captan los dispositivos móviles para ayudar a comprender y caracterizar el contexto en el que se encuentran estos, de manera que se pueda mejorar la precisión de la detección de contactos. Es aquí donde toma importancia el desarrollo del presente trabajo, ya que una aplicación de las características que se

proponen, facilitaría el proceso de recolección y procesamiento de los datos para su posterior análisis. Este proyecto también permite incorporar flexibilidad en lo que respecta al ajuste de parámetros de las instancias de recolección y/o transmisión de datos.

Presentación del Problema

Uno de los principales problemas a la hora de determinar si dos personas estuvieron en contacto es conocer las condiciones en las que se podría haber producido el encuentro, esto implicaría responder preguntas como ¿Por cuánto tiempo compartieron un espacio? ¿A qué distancia se encontraban estas personas? ¿El ambiente era abierto o cerrado? ¿Habían paredes de por medio? Podríamos continuar construyendo preguntas de este estilo para cada situación que se nos ocurra. Una primera aproximación a obtener una respuesta a estas preguntas sería recolectar datos que permitan determinar esa información y que este proceso sea preferentemente automático, confiable y rápido.

Actualmente casi cualquier persona dispone de un teléfono móvil inteligente, estos dispositivos constantemente están registrando e interactuando con diferentes señales de comunicación, las cuales dependen directamente del entorno físico en el que se propagan. Los teléfonos celulares también cuentan con una serie de sensores que perciben características de su entorno y del propio estado del dispositivo. Con esta descripción podemos proponer a estos aparatos tecnológicos como una fuente relevante de datos que nos podrían ayudar a responder las preguntas que se planteaban al inicio.

Dado que para poder caracterizar los encuentros se requiere un gran volumen de datos, sería interesante tener una solución que permita recolectarlos directamente desde los dispositivos móviles en términos de diferentes experimentos bajo diferentes condiciones de recolección, transmisión y distancias entre ellos.

De acuerdo a todo lo mencionado anteriormente es que en este trabajo se propone el desarrollo de una aplicación móvil capaz de capturar tanto información de diferentes señales de comunicación como de sensores y almacenarlos de manera persistente y bajo un modelo de datos que permita su posterior exportación y análisis mediante algoritmos externos a la aplicación. Se pretende que la aplicación desarrollada sea utilizada con fines experimentales y por tanto no se requiere un grado avanzado de desarrollo en su interfaz gráfica pero sí lo mínimo como para que cualquier persona involucrada en tal tarea pueda llevarla a cabo.

Objetivos

Como **objetivo general** de la propuesta se plantea desarrollar una aplicación móvil que recolecta datos de periféricos de comunicación y sensores, los cuales deben almacenarse de manera persistente localmente en la memoria del teléfono.

La aplicación debe ser capaz tanto de monitorear diferentes periféricos del dispositivo y capturar estados de los mismos como de configurar el dispositivo para que opere como una baliza Bluetooth, de aquí se desprende el **primer objetivo específico** que involucra el diseño e implementación de módulos de programa para tal fin.

Los datos capturados deben ser almacenados de manera persistente de forma que puedan ser recuperados posterior a la ejecución de los experimentos las veces que sea necesario independientemente del tiempo o del reinicio del dispositivo. Por ello se plantea como **segundo objetivo específico** el diseño de un modelo de datos y la implementación de una base de datos con las interfaces necesarias para realizar inserciones, modificaciones o borrado de datos.

Dado que la aplicación se utilizará principalmente para realizar experimentos, esta debe permitir configurar tanto los parámetros de recolección de datos como los asociados a la visibilidad del dispositivo en modo baliza. Estos experimentos pueden involucrar más de un dispositivo por lo que debe tener un mecanismo para compartir las configuraciones entre dispositivos de manera rápida. El **tercer objetivo específico** propone diseñar e implementar módulos de programa para estas tareas.

El **cuarto objetivo específico** involucra el diseño e implementación de interfaces gráficas de usuario para poder programar, configurar y visualizar estados de los experimentos que se vayan a realizar con la aplicación.

Dado que la aplicación únicamente recolecta datos y que el procesamiento de estos se deja para algún algoritmo externo, para el **quinto objetivo específico** se plantea desarrollar módulos de programa para la exportación de los datos a archivos en algún formato convencional para su posterior procesamiento y análisis.

Finalmente el **último objetivo específico** es desarrollar pruebas unitarias y de integración automatizadas que permitan verificar las funcionalidades de la aplicación.

Metodología de Desarrollo

Para este proyecto se optó por utilizar una metodología basada en técnicas de desarrollo iterativo, incremental y de verificación por pruebas.

Se plantea desarrollar los módulos de la aplicación y del modelo de datos en pequeños ciclos sucesivos de tiempo, de manera que cada funcionalidad pueda verificarse aisladamente mediante pruebas exclusivas para esa funcionalidad, esto lo hace iterativo. Conforme avanza el desarrollo del sistema se integran los diferentes módulos al funcionamiento de la aplicación final, constituyendo un progreso incremental, el cual también es verificado mediante pruebas de integración.

Estructura del documento

El presente informe se encuentra organizado en cinco capítulos.

En el *Capítulo 2*, se introduce al lector en el marco teórico necesario para comprender el desarrollo realizado. Se describe la detección de contactos, los dispositivos de comunicación involucrados, sensores, las aplicaciones Android, algunos patrones de diseño de software y también algunos patrones de arquitectura de sistemas.

En el *Capítulo 3* se describe el desarrollo del proyecto, desde la definición de requerimientos de la aplicación y el diseño de módulos de software que respondan a estos hasta el diseño de interfaces gráficas y pruebas automatizadas utilizadas.

En el *Capítulo 4*, se presentan los resultados de la utilización de la aplicación final obtenida del desarrollo del trabajo. Se describen y analizan los datos recolectados en un experimento.

Finalmente en el *Capítulo 5* se presentan las conclusiones y los potenciales trabajos futuros para mejorar algunos aspectos del funcionamiento de la aplicación.

Capítulo 2: Marco Teórico

Detección y Trazabilidad de Contactos

La trazabilidad de contactos es una herramienta utilizada en epidemiología que se basa en el conocimiento de las redes de interacción física tejidas entre las personas que, en su versión manual se puede ver dificultada la tarea de reconstruirlas generalmente debido a restricciones de privacidad o la dependencia en la memoria de las personas. Por esta razón es que las trazas de comunicación que se pueden obtener de los dispositivos móviles de las personas se convierten en una buena fuente de información para reconstruir los contactos que tuvieron las mismas [6].

Dispositivos de Comunicación

Recientes reportes prevén que para 2023 dos tercios de la población mundial tendrán acceso a Internet y a la par también la cantidad de dispositivos para ese entonces será más de tres veces la población, teniendo alrededor de 3,6 dispositivos conectados a la red por persona [1]. Este crecimiento es la razón por la cual cada vez cobra más importancia realizar estudios en materia de comunicación y en particular, cómo dentro del marco de la pandemia por el COVID-19 estos dispositivos pueden ayudarnos a enfrentar tal situación.

Wi-Fi

Wi-Fi es una tecnología estándar de comunicación inalámbrica presente en la gran mayoría de los dispositivos a partir del año 2000 aproximadamente [2], particularmente es una marca de la Wi-Fi Alliance [3] que es la organización comercial que cumple con los estándares IEEE 802.11 [4].

Las versiones más difundidas de esta tecnología trabajan en bandas de frecuencia de los 2.4GHz y 5GHz. A lo largo de los años y con cada revisión de las versiones, fue mejorando el rendimiento en términos de velocidades de transferencia y consumo energético como se menciona en un resumen de funcionalidades introducidas exclusivamente para esto último [5].

En particular, para este proyecto es de interés estudiar qué información de las redes Wi-Fi se puede conocer desde lo que es visible para un teléfono móvil inteligente y cómo se puede almacenar esta información de manera que sea útil para determinar contactos entre dispositivos.

Descubrimiento de redes Wi-Fi

Para anunciar la presencia de redes del estándar IEEE 802.11 se utilizan *Management Frames*, particularmente un tipo especial de estos que se llaman *Beacon Frames* [10]. Estos *frames* se

transmiten periódicamente en intervalos descritos por un campo dentro del mismo *frame*. Además portan información sobre los parámetros del set de servicios básicos (BSS) así las estaciones pueden estar a la escucha de estos y detectar las características de los puntos de acceso. Cada *frame* tiene el formato como el que se muestra en la Figura 1 adaptada de [10].

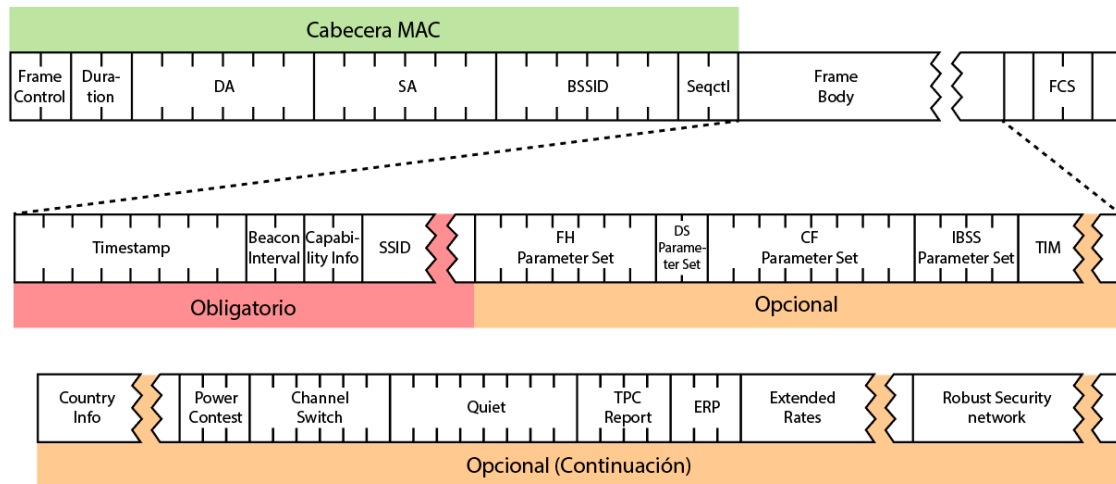


Figura 1: Formato de un Beacon Frame Wi-Fi.

Bluetooth

La tecnología Bluetooth admite que varios dispositivos se conecten unos con otros con un enlace de radio de corto alcance [7]. Está conformado por un *stack* completo compuesto por protocolos específicos de Bluetooth junto a otros que no son exclusivamente de Bluetooth.

Formato del paquete

Los paquetes de Bluetooth tienen un formato fijo (Figura 2 adaptada de [7]). El mismo está compuesto por un código de acceso, una cabecera de paquete y la carga útil del mensaje.

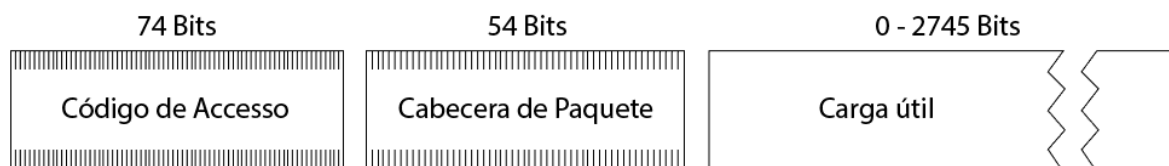


Figura 2: Formato de un paquete Bluetooth.

Perfiles de acceso genérico

Los perfiles de acceso genérico (*Generic Access Profile*, GAP) definen cómo se van a descubrir los dispositivos Bluetooth y aseguran que las conexiones se van a poder realizar sin importar el fabricante.

o el tipo de aplicación. Se puede intercambiar información por Bluetooth para descubrir el tipo de aplicación que soporta cada dispositivo [7].

Low Energy

Bluetooth Low Energy está diseñado para funcionar con muy poca energía. Entrega una funcionalidad confiable operando a la banda de frecuencia de 2.4GHz realizando la transmisión de datos a través de 40 canales. Además proporciona gran flexibilidad para los desarrolladores permitiendo configurar en que canal de capa física se desea transmitir consiguiendo así velocidades desde los 125 Kbps hasta los 2 Mbps, múltiples niveles de potencia de transmisión desde 1 mW hasta 100 mW [8]. Una vez establecida la comunicación se utilizan los perfiles de atributos genéricos (GATT) para enviar pequeños fragmentos de datos, denominados *atributos*.

Data Frames

Los *Data Frames* son un tipo de mensajes que se transmiten en una comunicación Bluetooth Low Energy, particularmente nos interesa conocer la estructura de un tipo de estos que se llama *Beacon Frame* y es el que se utiliza para publicitar un dispositivo para que este pueda ser descubierto por otros. En la Figura 3 (adaptada de [11]) se puede observar el formato de estos mensajes como así también la presencia de un campo nombrado como *Advertise Data*, este es de gran interés desde el punto de vista de los desarrolladores ya que es el que se puede utilizar para transmitir información personalizada. El resto de los campos generalmente son completados por diferentes partes del Stack Bluetooth.

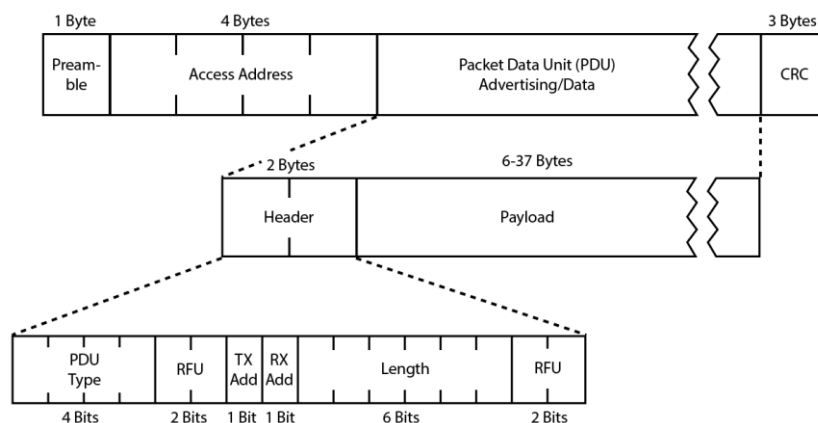


Figura 3: Formato de un Frame de Advertising BLE.

Roles en la comunicación Bluetooth Low Energy

En el contexto de una comunicación Bluetooth Low Energy los dispositivos pueden tomar los siguientes roles [9]:

- En el descubrimiento:

- Central: es el dispositivo que realiza los escaneos en busca de publicidad.
- Periférico: es el dispositivo que se publicita, es decir que se hace visible para los demás.
- Con una conexión ya establecida:
 - Cliente y Servidor GATT: Determina cómo se comunican entre sí los dispositivos.

Sensores

Actualmente muchos dispositivos móviles traen incorporados sensores que son capaces de proporcionar datos con alta precisión y exactitud. Podríamos supervisar tres categorías de sensores principales [12]:

- Sensores de movimiento: permiten medir fuerzas de aceleración y rotación. Pueden ser acelerómetros, sensores de gravedad, giroscopios, entre otros.
- Sensores ambientales: captan parámetros ambientales, como temperatura y presión del aire, iluminación o nivel de humedad. Pueden ser barómetros, fotómetros y termómetros.
- Sensores de posición: miden la posición física del dispositivo y pueden ser sensores de orientación o magnetómetros.

Haciendo foco en la información que se puede acceder desde la interfaz de programación de aplicaciones para Android [12], podemos realizar lecturas sobre los siguientes tipos de sensores:

- Acelerómetros: Mide la influencia de todas las fuerzas de aceleración que se le aplican a un dispositivo en cada uno de los tres ejes físicos (x, y, z), incluida la fuerza de gravedad.
- Sensores de gravedad: Miden únicamente la influencia de la fuerza de gravedad que se aplica a un dispositivo en cada eje físico (x, y, z).
- Sensores de aceleración lineal: Miden la fuerza de aceleración aplicada a un dispositivo en los tres ejes físicos sin incluir la fuerza de gravedad.
- Sensores de presión: Miden la presión del aire del ambiente.
- Sensores de iluminación: Miden el nivel de luz ambiental en lux.
- Sensores de humedad relativa: Miden el valor de porcentaje de humedad relativa del ambiente.
- Sensores de temperatura ambiente: Miden la temperatura ambiente aproximada de la habitación en la que se encuentra el dispositivo.
- Sensores de temperatura interna: Miden la temperatura del dispositivo.
- Sensores de campo magnético: Mide el campo geomagnético ambiental de los tres ejes físicos.

- Sensores de orientación: Miden los grados de rotación de un dispositivo alrededor de los tres ejes físicos. Se puede leer una matriz de inclinación en conjunto con una matriz de rotación. Este sensor es virtual (está basado en Software y depende de otros sensores como el de gravedad o el de campo magnético).
- Sensores de proximidad: Miden en cm la proximidad de un objeto con respecto a la pantalla de visualización del dispositivo.
- Sensores de vectores de rotación: Miden la orientación de un dispositivo mediante los tres elementos del vector de rotación del dispositivo.
- Giroscopios: Miden la velocidad de rotación de un dispositivo alrededor de cada eje físico.

Aplicación Android

Para poder desarrollar una aplicación para dispositivos móviles con Android como sistema operativo hay que entender algunos conceptos fundamentales [13] sobre estas:

- Como Android es un sistema basado en Linux y es multiusuario, se explota esta capacidad haciendo que cada aplicación se comporte como un usuario diferente. Esto implica que el sistema operativo le asigna un identificador de usuario a cada aplicación y este es el único que lo conoce.
- El sistema operativo establece permisos de acceso para cada uno de los archivos asociados a una aplicación asegurando que solo esta sea capaz de trabajar con ellos.
- Cada proceso tiene su propia máquina virtual brindando total aislamiento en su ejecución respecto de otras aplicaciones.

Android implementa algo que se llama *principio de mínimo privilegio*. Quiere decir que cada aplicación accede únicamente a aquellos recursos que le son estrictamente necesarios para poder llevar a cabo las tareas para las cuales fue implementada y nada más. De esta manera se tiene un entorno seguro en el que una aplicación no puede tener permisos excesivos y por ende convertirse en una amenaza general para todo el sistema. Aun así existen mecanismos mediante los cuales las aplicaciones pueden compartir datos entre ellas:

- Dos aplicaciones podrían compartir el mismo identificador de usuario Linux y de esta forma acceder archivos de una a la otra.
- Una aplicación puede solicitar permiso para acceder a ciertos datos del dispositivo y el usuario debe concederlos explícitamente.

Permisos

El mecanismo de permisos de Android tiene como principal objetivo proteger la privacidad del usuario [14]. Es por eso que al momento de desarrollar una aplicación es estrictamente necesario publicitar que permisos requiere esta en su manifiesto tal como se muestra en la Figura 4.

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.CAMERA" />
```

Figura 4: Ejemplo de declaración de permisos en el manifiesto de la aplicación. Fuente: Elaboración propia

Podemos categorizar permisos como *normales* a aquellos que no presentan un gran riesgo a la privacidad del usuario o al funcionamiento del dispositivo. Estos se declaran en el manifiesto pero no requiere que el usuario los conceda explícitamente, por el contrario el sistema operativo los otorga automáticamente.

Por otro lado están los permisos *riesgosos* qué, como su nombre indica, podrían afectar gravemente la privacidad del usuario o el funcionamiento del dispositivo. Para estos casos es cuando el usuario tiene que otorgar el permiso explícitamente. Esta petición la debe realizar el programador en alguna sección de la aplicación en la que se requiera acceder a una funcionalidad que dependa del permiso en cuestión. La llamada al sistema operativo se verá reflejada al usuario en un diálogo emergente similar a los que se muestran en la Figura 5.



Figura 5: Ejemplos de diálogo para que el usuario otorgue un permiso. Fuente: Elaboración propia

Actividades

Las Actividades son componentes esenciales de cualquier aplicación y sirven como punto de entrada para la interacción con el usuario, como así también para que éste navegue a través de la aplicación.

Las actividades están pensadas para admitir el inicio de la aplicación en diferentes posibles vistas [15]. Por ejemplo, el usuario al abrir la aplicación desde el inicio podría ser dirigido a una pantalla inicial configurada por defecto, pero si abre la aplicación desde una llamada producida por otra aplicación diferente podría acceder directamente a una sección en que realice la tarea vinculada a tal llamada.

La actividad *por defecto* que mencionamos anteriormente generalmente se especifica como *actividad principal* y es la primera pantalla que aparece cuando el usuario inicia la aplicación. Dentro de esta se configuran componentes que pueden llevar a otras actividades y así producir el flujo de usuario a lo largo de la aplicación.

Cada actividad debe ser declarada en el manifiesto de la aplicación como se aprecia en la Figura 6.

```
<activity android:name=".MainActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Figura 6: Ejemplo de declaración de una actividad en el manifiesto de la aplicación. Fuente: Elaboración propia

Cuando un usuario navega a través de la aplicación las actividades pasan por diferentes estados, al conjunto de estados por los que pasa una actividad es a lo que se le llama el *ciclo de vida* de la actividad. Para cada transición de estado hay un método de *devolución de llamada* que puede ser personalizado por el desarrollador para que la actividad se comporte de determinada forma ante tales cambios. El ciclo de vida de una actividad está representado de manera simplificada por la Figura 7 (adaptada de [15]), donde además se ven los métodos de devolución de llamada que pueden ser personalizados por el programador.

La implementación de los métodos del ciclo de vida depende de la complejidad de la actividad, pero es sumamente importante comprender cada uno de ellos para garantizar que la aplicación se comporte exactamente como esperan los usuarios.

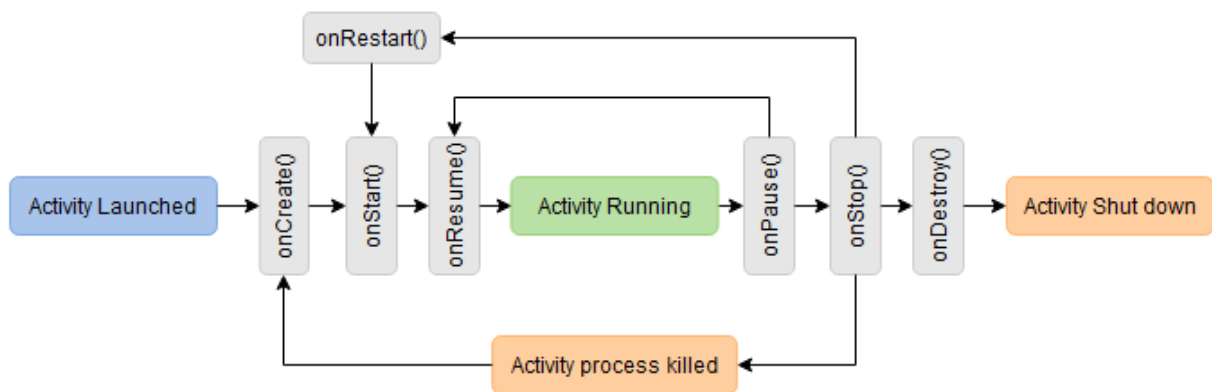


Figura 7: Ciclo de vida simplificado de una actividad.

Servicios

Los Servicios son componentes de aplicación pensados para correr tareas que demandan más tiempo y no exponen una interfaz gráfica al usuario. Una vez comenzada su ejecución puede que el servicio siga corriendo por algún tiempo, incluso si el usuario se cambia a otra aplicación [16].

Existen tres tipos de servicios que se pueden implementar en una aplicación:

- *Foreground Services*: Son servicios que realizan operaciones que de alguna forma son perceptibles por el usuario. Este tipo de servicio debe mostrar una *notificación*.
- *Background Services*: Realizan operaciones que no son directamente visibles para el usuario.
- *Bound Services*: Sirven para ofrecer arquitecturas de tipo *cliente-servidor* que permiten a los componentes interactuar con el servicio, enviar consultas, recibir resultados e incluso implementar comunicación entre procesos. Los procesos enlazados se ejecutan por el tiempo que algún componente se encuentre enlazado al mismo.

Algo para remarcar es que los servicios no necesariamente trabajan en otro hilo diferente al principal, de hecho para que esto suceda hay que declarar este comportamiento en el manifiesto del servicio.

Notificaciones

Una Notificación es un mensaje que muestra el sistema operativo fuera de la interfaz gráfica de la aplicación y sirve para proporcionar información extra sobre eventos o estados de esta. Generalmente se presentan en la barra de estado o de manera más detallada en el panel de notificaciones [17].

Cuando se envía una notificación el comportamiento natural es mostrar un ícono en la barra de estado como se muestra en la Figura 8.

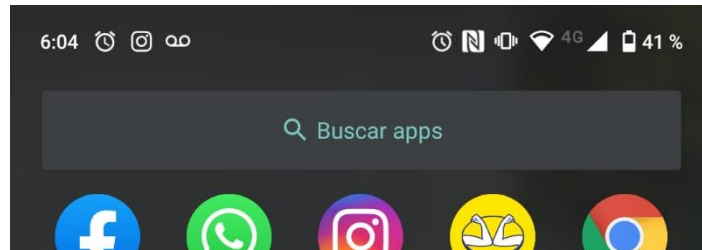


Figura 8: Ícono de notificación al lado izquierdo de la barra de estado. Fuente: Elaboración propia

Además como se muestra en la Figura 9, al deslizar hacia abajo la barra de estado se despliega el panel de notificaciones, donde se pueden ver más detalles y realizar acciones sobre estas.

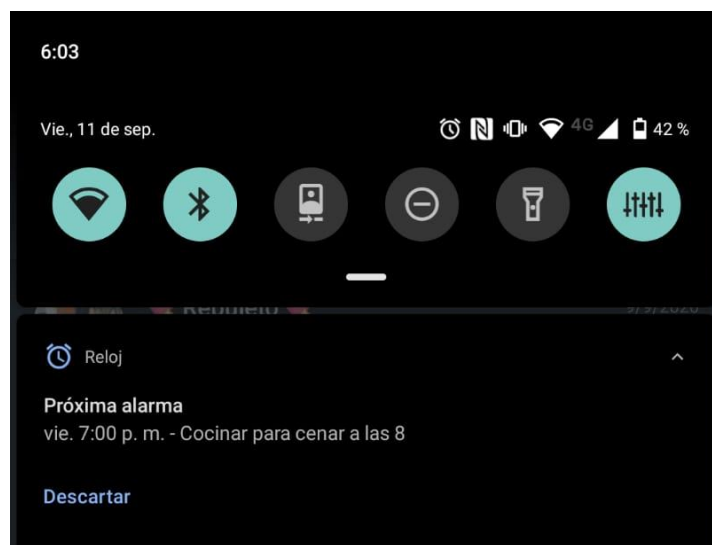


Figura 9: Detalles de notificaciones en el panel. Fuente: Elaboración propia

Intents

Las Intents son elementos que se utilizan para encapsular mensajes y solicitar acciones sobre componentes de la aplicación [18]. Tienen tres usos principales o más comunes:

- **Iniciar una actividad:** Al incluir una *Intent* en la iniciación de una actividad se puede usar para enviarle información que esta podría utilizar para construirse. Además se puede esperar que la actividad recientemente iniciada devuelva un resultado a la que la llamó y la respuesta también se comunica por medio de una *Intent*.
- **Iniciar un servicio:** funciona de manera similar a las actividades bajo cualquier tipo de servicio descrito anteriormente.
- **Transmitir emisiones:** Dado que las emisiones son avisos que cualquier aplicación puede recibir o emitir, las *Intents* son una interfaz estandarizada para comunicar información entre estas.

Broadcasts

Dentro de las aplicaciones Android existe un mecanismo llamado *emisión* (Broadcast) [19] que se utiliza para enviar o recibir mensajes desde y hacia aplicaciones. Se comporta similar al patrón de diseño de publicación-suscripción. Todos los mensajes de emisión viajan encapsulados en *Intents*.

Generalmente una aplicación se registra para recibir emisiones del sistema operativo, pero estas también pueden emitir sus propios mensajes de emisión.

Para que una aplicación pueda capturar las emisiones se deben declarar receptores en el manifiesto o bien, registrarlos en el contexto. En cualquier caso, el desarrollador debe crear subclases de `BroadcastReceiver` [20] y en esta sobrescribir el método `onReceive` que es el encargado de procesar el mensaje de emisión capturado.

Para enviar emisiones, la interfaz de programación de Android ofrece tres mecanismos. El primero envía los mensajes a un receptor por vez, lo que permite que se pueda propagar el resultado al siguiente o anular por completo la emisión para que no se transmita a otros. El orden en que se transmiten puede controlarse con un atributo de prioridad del filtro de *Intent*. El segundo mecanismo envía las emisiones a todos los receptores en un orden no especificado, este es el modo normal de operación y el más eficiente pero implica que los receptores no pueden leer los resultados de otros. Finalmente el último método limita las emisiones únicamente a los receptores que pertenecen a la misma aplicación y resulta de utilidad cuando no se necesita enviar información entre aplicaciones ya que es mucho más eficiente y libera al desarrollador de problemas de seguridad en torno a las demás aplicaciones.

Patrones de diseño

En general un patrón de diseño tiene cuatro elementos como se menciona en [23], comenzando por un *nombre* que describe el problema de diseño, sus soluciones y consecuencias en una o a lo sumo dos palabras. El *problema* describe en qué situaciones se puede aplicar el patrón y explica su contexto. La *solución* explica qué elementos intervienen en el diseño con sus respectivas responsabilidades, relaciones y colaboraciones, en general no busca describir concretamente la implementación o el diseño sino más bien la *plantilla* o guía a seguir para resolver el problema. Por último tenemos las *consecuencias* que son los resultados de aplicar el patrón, son altamente importantes para poder evaluar el impacto de utilizar un patrón u otro en un sistema.

Entonces, en resumen podemos decir que un patrón de diseño es una abstracción de los aspectos más comunes de una estructura de diseño que es útil para recrear componentes de programa reutilizables.

Observer

El patrón Observer, como se describe en [23], define una dependencia en un único sentido entre objetos de manera que el cambio de estado de uno se propaga y actualiza automáticamente a aquellos interesados en conocerlo.

Este patrón es aplicable cuando un cambio de un objeto afecta a otros y se desconoce cuántos objetos necesitan ser modificados. También es aplicable cuando un objeto necesita notificar cambios a otros sin asumir quienes son estos.

En la Figura 10 se pueden observar los actores que interactúan en este patrón para conseguir un comportamiento como el que se describió anteriormente. Los roles en el patrón son *subject* y *observer*, donde el primero es el que guarda registro de quienes son los que están observando a la espera de cambios. *Subject* tiene métodos para registrar y cancelar el registro de un *observer* como así también un método para notificarle a estos cuando ocurre un cambio. Por otro lado cada *observer* tiene un método que actúa cuando recibe la notificación, de esta manera puede actuar ante el cambio notificado.

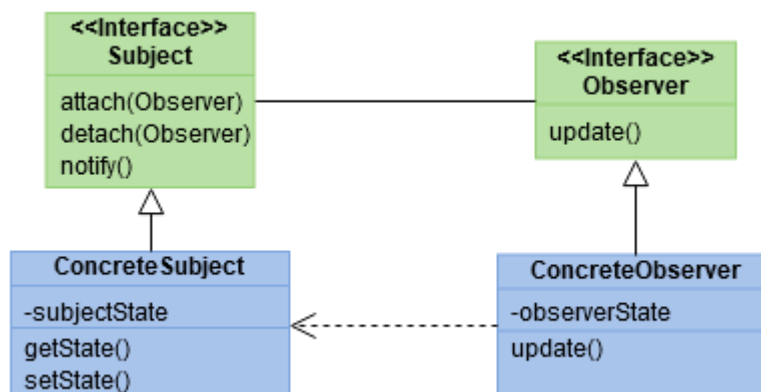


Figura 10: Estructura del patrón Observer. Fuente: Elaboración propia

Singleton

El patrón *Singleton* asegura que solo exista una única instancia de una clase y además provee acceso a esta.

Como se menciona en [23] es un patrón aplicable en casos donde debe existir una única instancia de una clase y debe ser accesible para los clientes con una interfaz bien definida. En este patrón existe un solo actor como se puede apreciar en la Figura 11, que es el encargado de instanciar internamente la clase con el método `singletonOperation` y proveer acceso a esta por medio del método `getInstance`.

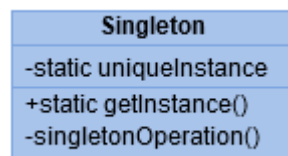


Figura 11: Actor del patrón Singleton. Fuente: Elaboración propia

Data Access Object

Dado que en muchos sistemas el acceso a datos puede darse desde distintos componentes, el código para accederlos queda distribuido y el acceso a datos codificado en componentes de negocios generando cierto acople a la fuente de datos. De aquí se desprende la utilidad de este patrón ya que está pensado para separar el acceso a la fuente de datos de las lógicas de negocio, generando así un desacoplamiento que resuelve el problema anteriormente descrito.

Tal y como se menciona en [24] este patrón tiene dos funciones principales:

- Contiene la lógica para el acceso a datos.
- Gestiona la conexión con la fuente de datos para poder obtenerlos y guardarlos.

La estructura general del patrón se ve representada por la figura a continuación.

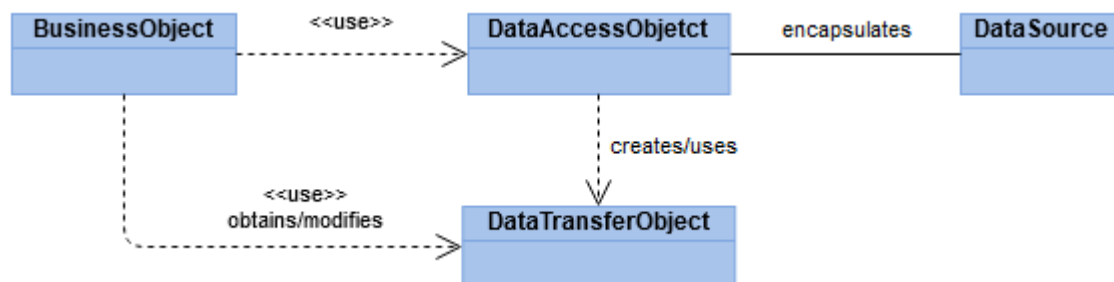


Figura 12: Estructura del patrón Data Access Object. Fuente: Elaboración propia

Repository

Basado en el patrón DAO, este permite introducir una capa de abstracción denominada *Repository* y es la que se considera como fuente real de datos [26] para los objetos de lógica de negocios. *Repository* puede obtener los datos de diferentes fuentes (Figura 13) como bases de datos locales, remotas o memoria caché, y es éste el que en base a los datos que recibe y a *Data Mappers* construye los modelos adecuados para ser utilizados por los objetos de lógica de negocios.

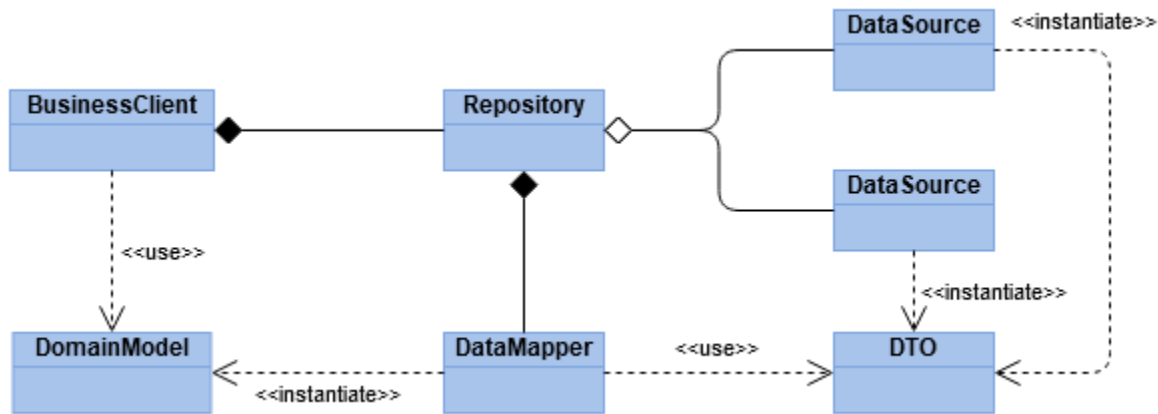


Figura 13: Representación del patrón Repository. Fuente: Elaboración propia

Arquitectura de un sistema

Se puede pensar la arquitectura de un sistema como una combinación de cuatro elementos que tienen que estar presentes como se menciona en [21]. Primero tiene que existir una *estructura* que describa el estilo de arquitectura con el que se implementa un sistema (como por ejemplo micro servicios, por capas o de micro kernel). Lo segundo es hablar de *características* de la arquitectura, estas definen un criterio de éxito para el sistema y que está estrechamente relacionado a la funcionalidad del sistema. Otro aspecto de relevancia son las *decisiones* de la arquitectura, es decir un conjunto de reglas por el cual debe guiarse la construcción del sistema. Finalmente el último factor son los *principios de diseño*, estos se diferencian de las decisiones en el sentido de que son más bien una guía más que una regla.

Arquitectura Model-View-ViewModel

La arquitectura de software *Model-View-ViewModel* está basada en componentes que permiten separar la presentación, de la lógica de información que se va a presentar y del modelo que contiene esta información.

En este patrón de arquitectura hay tres componentes esenciales como mencionamos antes, en [22] los describe como:

- *Model*: es la representación abstracta de los datos que describen el problema en cuestión.
- *View*: representa la interfaz gráfica de una aplicación.
- *ViewModel*: es un mediador entre el modelo y la vista, es el encargado de transformar los datos del modelo para que la vista pueda mostrarlos.

La principal ventaja de este patrón de arquitectura es que las *views* conocen a las *ViewModel* a través de un patrón *observer*, como se aprecia en la Figura 14, facilitando la comunicación de información desde la última a la primera y también permitiendo una modularización mayor entre los componentes

ya que las *views* se desprecupan de cómo se construye la información que tienen que mostrar mientras que las *ViewModel* se desprecupan de cómo comunicar esta información.



Figura 14: Interacción de los componentes de la arquitectura MVVM. Fuente: Adaptado de [22]

Arquitectura sugerida para Aplicaciones Android

Basados en el patrón MVVM, la documentación de desarrollo de aplicaciones para Android [25] sugiere seguir un esquema como el representado por la Figura 15 (adaptada de 25). Las ventajas de utilizar este tipo de arquitectura es que permite el desacoplamiento de las vistas de la aplicación con respecto a las fuentes de datos, por lo que se podría implementar una aplicación con una o varias fuentes de datos independientes como así también realizar sincronización entre ellas.

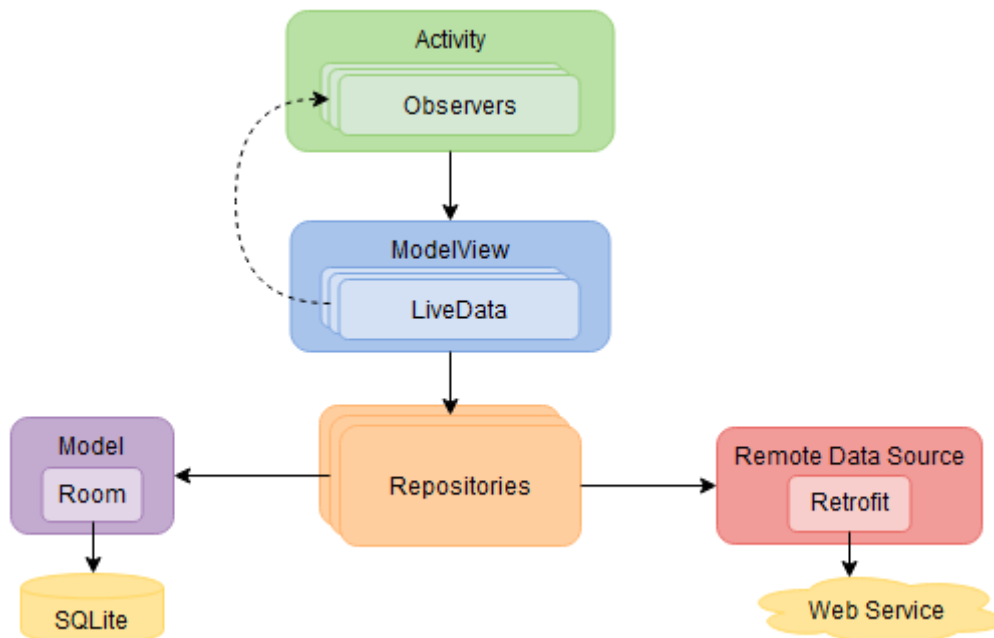


Figura 15: Arquitectura sugerida en la documentación de Android.

Capítulo 3: Desarrollo del Proyecto

Para el desarrollo del proyecto se siguió un esquema como el de la Figura 16, donde cada uno de los ítems del diagrama se encuentra descrito por un título dentro del presente capítulo.

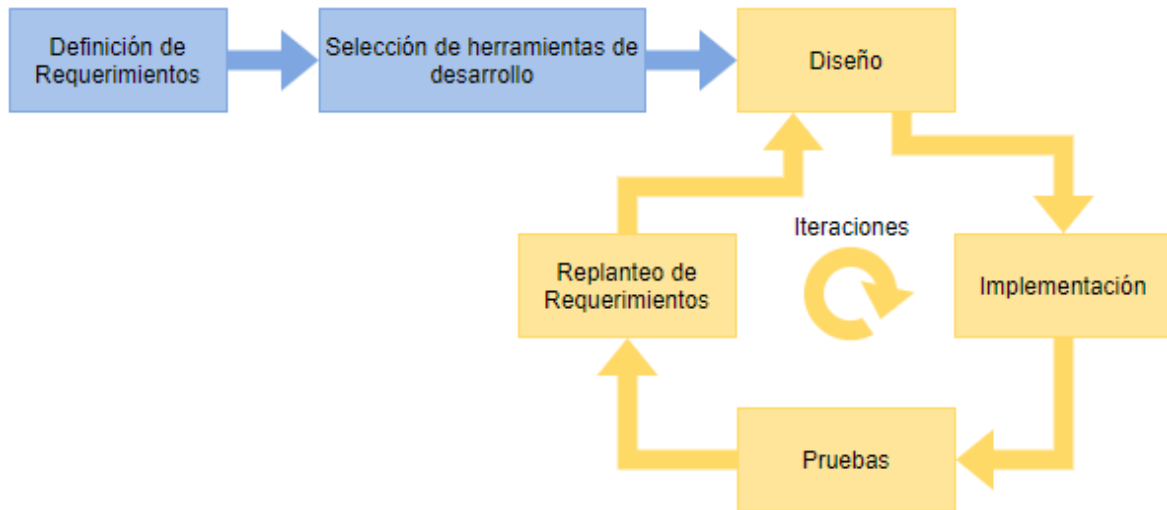


Figura 16: Proceso de desarrollo del proyecto. Fuente: Elaboración propia

Este esquema de desarrollo parte de la *Definición de los requerimientos*, que implica el análisis de la problemática y diálogo con el equipo de investigación. De aquí se desprenden los diseños que se van a implementar posteriormente.

Con los requerimientos definidos se pasa a una etapa de *Evaluación y selección de herramientas* para el desarrollo de la aplicación que mejor se adapten a los requerimientos.

La etapa de desarrollo de la aplicación se llevó a cabo en *Iteraciones* que involucran fases de *Diseño*, *Implementación* y *Pruebas* donde para determinar la finalización se fijó un criterio basado en verificación de funcionalidades asociadas a requerimientos mediante pruebas, es decir que cuando una prueba da un resultado de éxito se acepta un requerimiento como verificado y el desarrollo finaliza cuando todos los requerimientos están en tal estado.

Riesgos

Los riesgos fueron identificados, analizados y se planificaron medidas en base a lo estudiado. A continuación se detalla lo visto para cada instancia.

Identificación y Análisis

A partir del diálogo con el equipo de investigación y estudiando el dominio del problema se identificaron los posibles peligros en la aplicación, en la Tabla 1 se detallan estos y se especifican las probabilidades de riesgo y que nivel de aceptabilidad se adopta en cada caso.

Tabla 1: Riesgos identificados en este proyecto. Fuente: Elaboración propia

| Peligro identificado | Probabilidad de Riesgo | Riesgo estimado | Aceptabilidad |
|---|------------------------|-----------------|---------------|
| Ingreso de valores inválidos para las configuraciones de escaneo | Alto | Alto | Inaceptable |
| Solapamiento de recolectores sobre un mismo recurso de Hardware | Medio | Medio | Aceptable |
| Desbordamiento de memoria durante la recolección de datos | Bajo | Medio | Aceptable |
| Acceso simultaneo a la Base de Datos | Alto | Alto | Inaceptable |
| Cierre de la aplicación durante la ejecución de un Experimento | Alto | Alto | Inaceptable |
| Bloqueo de ejecución de la aplicación por escritura de datos en memoria no volátil. | Alto | Alto | Inaceptable |

Planificación

Para cada uno de los peligros identificados y, teniendo en cuenta la probabilidad de riesgo, el nivel de riesgo y la aceptabilidad, es que se planificaron medidas para mitigar estos peligros. En la Tabla 2 se pueden observar las medidas planificadas para cada peligro.

Tabla 2: Planificación de medidas para cada peligro identificado. Fuente: Elaboración propia

| Peligro identificado | Medida planificada |
|---|--|
| Ingreso de valores inválidos para las configuraciones de escaneo | Implementar controles sobre los valores ingresados en cada formulario |
| Solapamiento de recolectores sobre un mismo recurso de Hardware | Utilizar mecanismos provistos por el Sistema Operativo que resuelvan el acceso a los recursos |
| Desbordamiento de memoria durante la recolección de datos | No planificado |
| Acceso simultaneo a la Base de Datos | Utilizar un motor de Bases de Datos que maneje múltiples accesos de manera simultanea |
| Cierre de la aplicación durante la ejecución de un Experimento | Implementar la ejecución de los Experimentos con algún mecanismo que mantenga vivo el hilo de esta tarea. |
| Bloqueo de ejecución de la aplicación por escritura de datos en memoria no volátil. | Utilizar mecanismos de ejecución de tareas no bloqueantes para implementar la escritura de datos en la memoria no volátil. |

Definición de Requerimientos

Detalle en alto nivel de funcionalidades

En la Tabla 3 se describen las funcionalidades a alto nivel que debe tener la aplicación al final del desarrollo.

Tabla 3: Detalle de funcionalidades de la aplicación. Fuente: Elaboración propia

| Código | Descripción |
|--------|---|
| CONF | Configuración de experimentos asociados a un dispositivo en términos de períodos de recolección y transmisión de datos. |
| RUN | Programación y ejecución de corridas futuras de experimentos |
| WIFI | Recolección, almacenamiento y exportación de datos de redes Wi-Fi |
| BLUE | Recolección, almacenamiento y exportación de datos de dispositivos Bluetooth |
| BLE | Recolección, almacenamiento y exportación de datos de dispositivos Bluetooth Low Energy |
| SENS | Recolección, almacenamiento y exportación de datos de Sensores incorporados en el dispositivo |
| BALIZA | Visibilidad del dispositivo mediante Bluetooth Low Energy (Baliza BLE) |
| UI | Interfaz gráfica |

Detalle de requerimientos funcionales

De la problemática que se pretende resolver y del diálogo con el equipo de investigación se extraen los requerimientos detallados en la Tabla 4. Estos requerimientos están estrechamente relacionados a las funcionalidades descritas en la sección anterior.

Tabla 4: Detalle de requerimientos funcionales. Fuente: Elaboración propia

| Código | Descripción |
|--------|---|
| RF-00 | La aplicación debe almacenar de manera persistente un nombre en código, un identificador único, marca y modelo del dispositivo. |
| RF-01 | La interfaz de la aplicación debe permitir modificar el nombre en código del dispositivo. |
| RF-02 | La interfaz de la aplicación debe permitir generar un nuevo identificador único para el dispositivo. |
| RF-03 | La aplicación debe almacenar de manera persistente un código y descripción de un experimento. |
| RF-04 | La aplicación debe almacenar de manera persistente etiquetas que caracterizan un experimento. |
| RF-05 | La aplicación debe almacenar de manera persistente el tiempo activo, tiempo inactivo y cantidad de ventanas de recolección de datos de redes Wi-Fi. |
| RF-06 | La aplicación debe almacenar de manera persistente el tiempo activo, tiempo inactivo y cantidad de ventanas de recolección de datos de dispositivos Bluetooth. |
| RF-07 | La aplicación debe almacenar de manera persistente el tiempo activo, tiempo inactivo y cantidad de ventanas de recolección de datos de dispositivos Bluetooth Low Energy. |
| RF-08 | La aplicación debe almacenar de manera persistente el tiempo activo, tiempo inactivo y cantidad de ventanas de recolección de datos de Sensores incorporados en el dispositivo. |
| RF-09 | Cada configuración de recolección o transmisión debe asociarse únicamente a un experimento. |
| RF-10 | La aplicación debe almacenar de manera persistente el tiempo activo, tiempo inactivo, cantidad de ventanas, potencia e intervalos de transmisión de datos en modo Baliza BLE. |
| RF-11 | La interfaz de la aplicación debe validar que cada experimento tenga asociado como mínimo una configuración de recolección o transmisión. |
| RF-12 | La interfaz de la aplicación debe validar que cada experimento tenga asociado como máximo una configuración de recolección de datos de redes Wi-Fi. |

| | |
|-------|---|
| RF-13 | La interfaz de la aplicación debe validar que cada experimento tenga asociado como máximo una configuración de recolección de datos de dispositivos Bluetooth. |
| RF-14 | La interfaz de la aplicación debe validar que cada experimento tenga asociado como máximo una configuración de recolección de datos de dispositivos Bluetooth Low Energy. |
| RF-15 | La interfaz de la aplicación debe validar que cada experimento tenga asociado como máximo una configuración de recolección de datos de Sensores. |
| RF-16 | La interfaz de la aplicación debe validar que cada experimento tenga asociado como máximo una configuración de transmisión de datos en modo Baliza BLE. |
| RF-17 | La interfaz de la aplicación debe permitir asociar etiquetas nuevas o previamente creadas a un experimento. |
| RF-18 | La aplicación debe recolectar datos de redes Wi-Fi en ventanas de tiempo determinadas por períodos activos e inactivos de recolección. Este proceso se debe repetir por una cantidad determinada de veces. |
| RF-19 | La aplicación debe almacenar de manera persistente los siguientes datos de una red Wi-Fi: BSSID, marca de tiempo de la lectura, ancho del canal, frecuencias centrales primaria y secundaria, frecuencia, nivel de señal y si la red es Passpoint. |
| RF-20 | La aplicación debe recolectar datos de dispositivos Bluetooth en ventanas de tiempo determinadas por períodos activos e inactivos de recolección determinados por una configuración. Este proceso se debe repetir por una cantidad determinada de veces. |
| RF-21 | La aplicación debe almacenar de manera persistente la siguiente información de dispositivos Bluetooth: dirección física, marca de tiempo de la lectura, clase Bluetooth, estado de vinculación y tipo de dispositivo. |
| RF-22 | La aplicación debe recolectar datos de dispositivos Bluetooth Low Energy en ventanas de tiempo determinadas por períodos activos e inactivos de recolección determinados por una configuración. Este proceso se debe repetir por una cantidad determinada de veces. |
| RF-23 | La aplicación debe almacenar de manera persistente la siguiente información de dispositivos Bluetooth Low Energy: dirección física, RSSI, potencia de transmisión, marca de tiempo de la lectura, identificador del set de publicidad, capa física primaria, capa física secundaria, intervalo de publicidad, si el dispositivo es conectable y si el dispositivo opera en modo <i>legacy</i> . |
| RF-24 | La aplicación debe recolectar datos de Sensores en ventanas de tiempo determinadas por períodos activos e inactivos de recolección determinados por una configuración. Este proceso se debe repetir por una cantidad determinada de veces. |
| RF-25 | La aplicación debe almacenar de manera persistente los siguientes datos de Sensores: tipo, índice del dato, valor del dato. |

| | |
|-------|--|
| RF-26 | La aplicación debe transmitir mensajes de publicidad Bluetooth Low Energy en ventanas de tiempo determinadas por períodos activos e inactivos, a potencia e intervalos de transmisión determinados por la configuración. Este proceso se debe repetir por una cantidad determinada de veces. |
| RF-27 | La aplicación debe almacenar de manera persistente la programación de una corrida de un experimento. |
| RF-28 | La interfaz de la aplicación debería validar que la corrida de un experimento se programe a futuro en cualquier caso. |
| RF-29 | El proceso de recolección de datos debe ejecutarse aun cuando la interfaz de la aplicación se encuentra cerrada. |
| RF-30 | La aplicación debe permitir exportar los datos en archivos con formato CSV. |
| RF-31 | Una vez finalizado el proceso de recolección de datos, se podrían exportar automáticamente los datos recolectados. |
| RF-32 | La aplicación debe permitir exportar experimentos en formato JSON. |
| RF-33 | La aplicación podría permitir la carga de información referida a un experimento a través del escaneo de un código QR. |
| RF-34 | La aplicación podría generar un código QR que contenga información referida a un experimento. |
| RF-35 | La interfaz de la aplicación debe permitir visualizar un listado de experimentos guardados. |
| RF-36 | La interfaz de la aplicación debe permitir visualizar el detalle de un experimento y este debe contener: código QR asociado, descripción, configuraciones asociadas, etiquetas asociadas y corridas programadas. |
| RF-37 | La interfaz de la aplicación debe permitir visualizar el estado de ejecución de la corrida de un experimento. |

Detalle de requerimientos no funcionales

El desarrollo de la aplicación se encuentra condicionado por ciertos requerimientos no funcionales detallados en la Tabla 5. Estos tienen que ver con restricciones del sistema operativo, recursos de hardware, entre otros.

Tabla 5: Detalle de requerimientos no funcionales. Fuente: Elaboración propia

| Código | Descripción |
|--------|---|
| RNF-00 | La aplicación debe correr en dispositivos con Android 9 o superior. |
| RNF-01 | Los datos exportados deben almacenarse en una memoria no volátil. |
| RNF-02 | La aplicación debe implementarse en Java o Kotlin. |
| RNF-03 | La aplicación debe tener una interfaz gráfica basada en Material Design. |
| RNF-04 | La aplicación debe poder ejecutarse en sistemas con 2GB de memoria RAM como mínimo. |
| RNF-05 | La aplicación debe poder ejecutarse en sistemas con 4 núcleos como mínimo. |

Etapas del Desarrollo

El desarrollo se llevó a cabo en **X** etapas. En la **X** etapa se diseñaron e implementaron cada uno de los módulos de recolección de manera individual, también se diseñó e implementó el mecanismo de planificación de estos de recolectores. Una vez que se finalizó el desarrollo de estos módulos y que se verificó su correcto funcionamiento, se diseñó e implementó el modelo de datos.

Posteriormente, en la **X** etapa se diseñó e implementó el módulo de transmisión que utiliza Bluetooth Low Energy para que el dispositivo opere como baliza aprovechando el mismo mecanismo de planificación de la etapa anterior.

En la **X** etapa se trabajó sobre los módulos de exportación de datos, comenzando por el diseño e implementación del módulo para exportar datos en formato CSV y seguidamente, haciendo uso de la reutilización de código y la modularidad del diseño propuesto se implementó el módulo para exportar en formato JSON.

En cada etapa se logró un entregable que verificaba un conjunto de requerimientos y que mitigaba uno o más peligros de los identificados. También es importante mencionar que en cada etapa se desarrollaron las vistas correspondientes a cada una de las funcionalidades asociadas a la etapa en concreto.

Selección de herramientas

A partir de los requerimientos detallados en la sección anterior se realizó una investigación y evaluación de herramientas y recursos de software necesarios para poder realizar el desarrollo y, se concluyó que los óptimos son los siguientes:

- Gitlab [33]: sistema de control de versiones basado en GIT. Se elige esta herramienta por sobre otras del mismo tipo por la posibilidad de crear repositorios privados teniendo todas las funcionalidades para realizar integración continua dentro de la misma plataforma y de manera gratuita. (A futuro se planea utilizar integración continua para agilizar el desarrollo).
- Visual Paradigm Online [34]: set de herramientas para desarrollar diagramas. Comparado con otras herramientas, ofrece la posibilidad de guardar y abrir los modelos en/desde de Google Drive, además posee plantillas para realizar diagramas de todos los tipos especificados en UML con la posibilidad de personalizar al detalle el estilo visual de los diagramas, es gratuito y no tiene límites en la cantidad de diagramas o esquemas que se pueden realizar.
- Entorno de desarrollo Android Studio 4 [31]: Incluyendo todos los kits de desarrollo de software y pruebas para sistemas operativos Android 9 y superiores, simulador de dispositivos, depuradores, etc. Se eligió este IDE por la integración de todas la herramientas para desarrollar y depurar aplicaciones para Android sin la necesidad de instalar *plugins* adicionales.
- Room [27 - 28]: Librería para implementación y prueba de bases de datos locales. Seleccionada porque introduce una capa de abstracción que permite enfocarse en la implementación del esquema de la base de datos y las consultas correspondientes para operar con los datos.
- QR Generator [29]: Librería para generación de códigos QR. Se eligió por ser una librería de código abierto y que cumple con las necesidades para personalizar la generación de códigos QR.
- Code Scanner [30]: Librería para implementar lectores de códigos QR. Se eligió esta librería por incluir una Actividad de Android lista para escanear y obtener los datos codificados en el código QR.
- Material Design [32]: Librería de componentes de interfaz gráfica para el desarrollo de interfaces de usuario. Se optó por estos componentes de interfaz gráfica porque respetan el estándar de diseño sugerido por Google para sus aplicaciones y por lo tanto para aplicaciones de sistemas operativos Android.
- GSON [42]: Librería para el mapeo de Objetos Java a notación en formato JSON. Se la eligió porque constituye una solución completa para la transformación de objetos sin la necesidad de

manipular cada uno de los atributos de estos para obtener una cadena de caracteres en formato JSON.

Desarrollo de la Aplicación

En primera instancia se propuso un esquema general de la arquitectura de la aplicación basada en los conceptos de arquitectura por capas descritos en el marco teórico y la arquitectura sugerida en la documentación de desarrollo de aplicaciones Android.

El desarrollo de la aplicación se planteó en cinco grandes etapas, cada una compuesta por iteraciones con fases de *Diseño*, *Implementación* y verificación con *Pruebas*. Estas están relacionadas con las siguientes áreas:

- Arquitectura general y de la aplicación.
- Modelo de datos.
- Módulos de recolección.
- Módulo de transmisión (Baliza BLE).
- Módulos de exportación de datos.
- Interfaces gráficas.

A lo largo de esta sección se documenta el proceso de desarrollo que se siguió para cada una.

Arquitectura

Como primer boceto del diseño se propuso una arquitectura general del sistema, la cual está representada en la Figura 17.

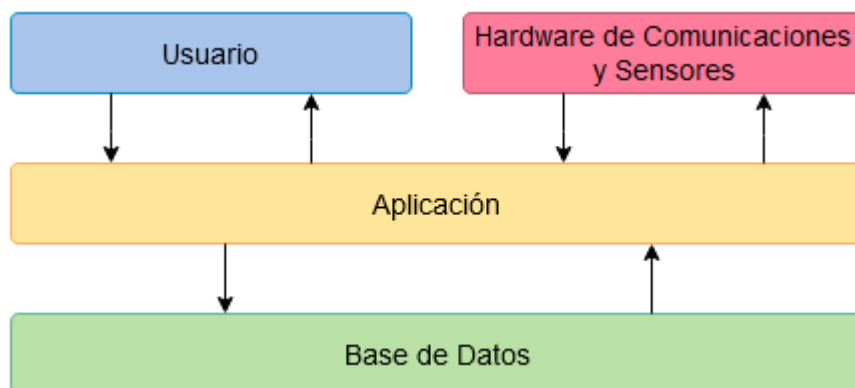


Figura 17: Arquitectura de alto nivel del sistema. Fuente: Elaboración propia

Teniendo en cuenta la arquitectura de alto nivel, los requerimientos definidos y la teoría expuesta referida a arquitecturas de sistema y arquitecturas sugeridas para aplicaciones Android, se plantea una arquitectura de aplicación como la de la Figura 18.

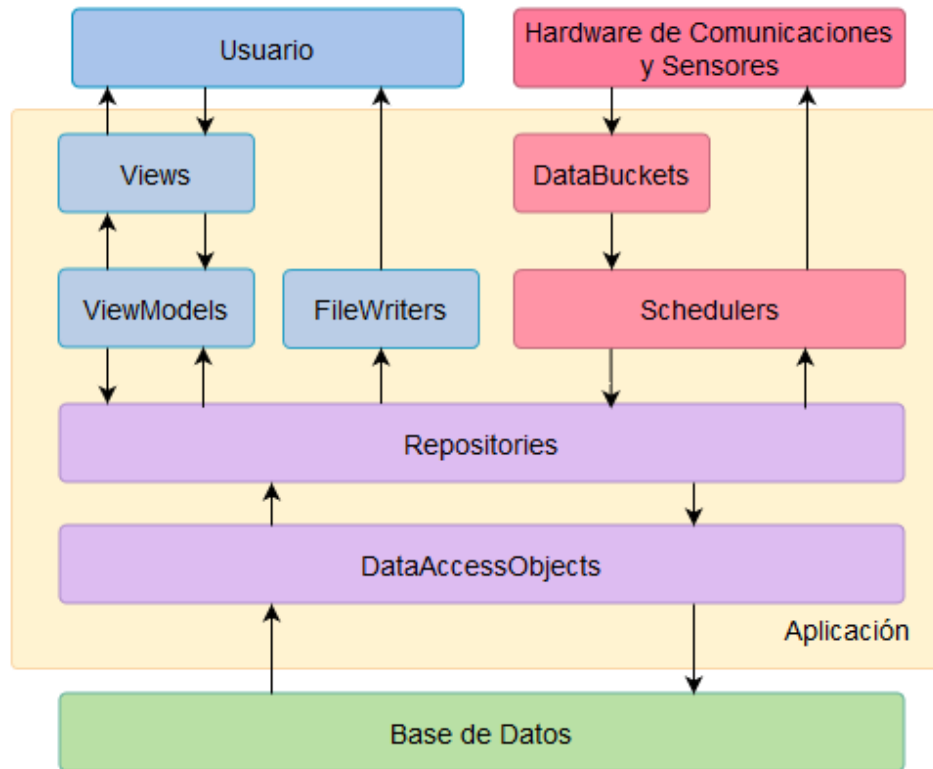


Figura 18: Arquitectura con detalle de la aplicación. Fuente: Elaboración propia

Modelo y base de datos

Antes de hablar del modelo que representa a los datos del problema, es preciso definir algunos términos relacionados a la problemática y que se desprenden directamente del diálogo con el equipo de investigación y de los requerimientos definidos al principio del presente trabajo:

- Un **dispositivo** es la plataforma en la que se instalará la aplicación. El mismo tiene un sistema operativo, un conjunto de periféricos de comunicación y sensores. Particularmente, para este trabajo se trabajarán con dispositivos móviles con Android como sistema operativo.
- Un **experimento** es una o más instancias de recolección y/o transmisión de datos de periféricos de comunicación y sensores presentes en el dispositivo. Un experimento está definido como un set de configuraciones.
- Para facilitar la caracterización de un experimento se le asociará a cada uno de estos, una o más **etiquetas**.
- La **configuración** de cada módulo de recolección y/o transmisión (Wi-Fi, Bluetooth, etc.) determina cada cuánto tiempo se encenderá el escaneo, por cuánto tiempo estará encendido y cuántas veces se repetirá este proceso.
- Una **corrida** es una instancia de un experimento de ejecución. Se programan a futuro y pueden ejecutarse de a una por vez.

- Cada período activo de recolección es lo que definimos como una **ventana** de recolección.
- Cada ventana de datos tiene asociada uno o más **registros** de datos. La información que almacena cada registro depende del módulo (Wi-Fi, Bluetooth, etc.) que lo esté recolectando.

A partir de la descripción de los elementos que intervienen en el modelo de datos y cómo se relacionan entre sí, se construye un *Diagrama Entidad-Relación* representado por la Figura 19, el cual se implementó en la base de datos respetando el diagrama de tablas de la Figura 20.

La implementación se realizó sobre el motor de bases de datos relacionales SQLite, que es soportado por sistemas operativos Android para ser utilizado como sistema de almacenamiento persistente. Igualmente, el esquema que se propuso se puede implementar en cualquier otro sistema de gestión de bases de datos relacionales, permitiendo así una ventajosa flexibilidad para posibles trabajos futuros o el escalamiento del sistema a un paradigma distribuido.

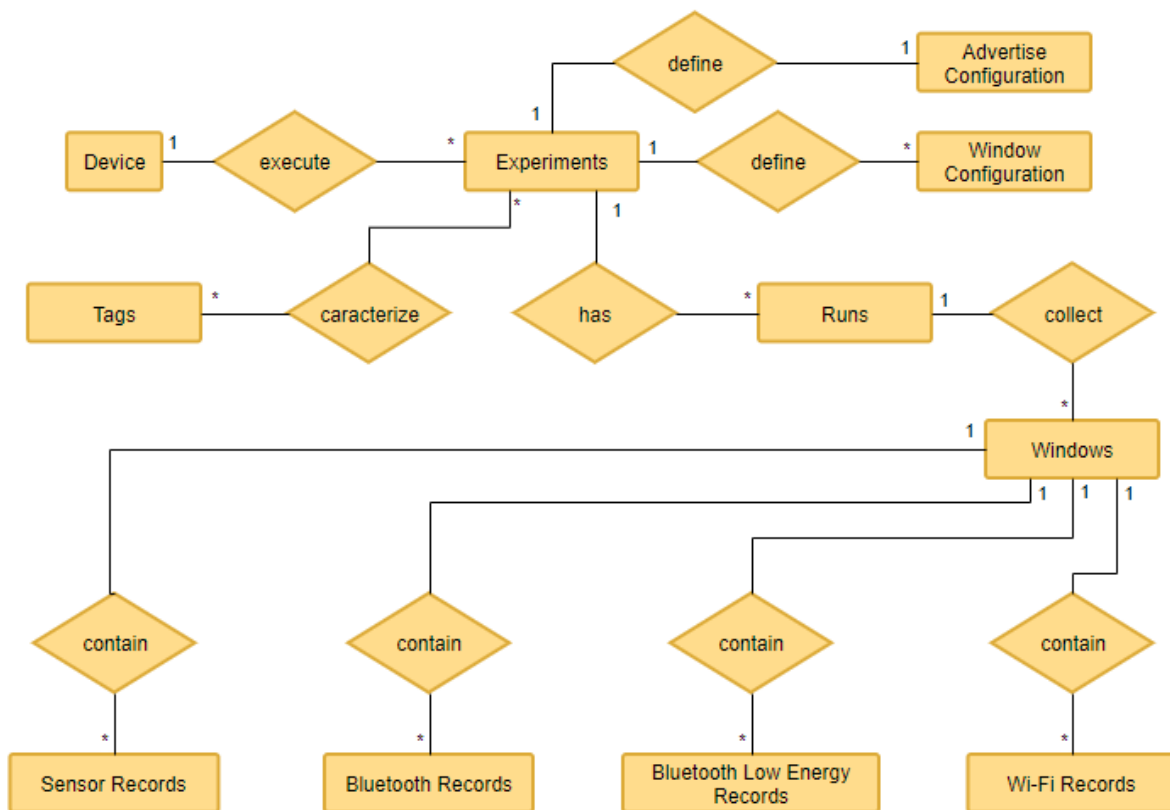


Figura 19: Esquema Entidad-Relación para el modelo de datos de la Aplicación. Fuente: Elaboración propia

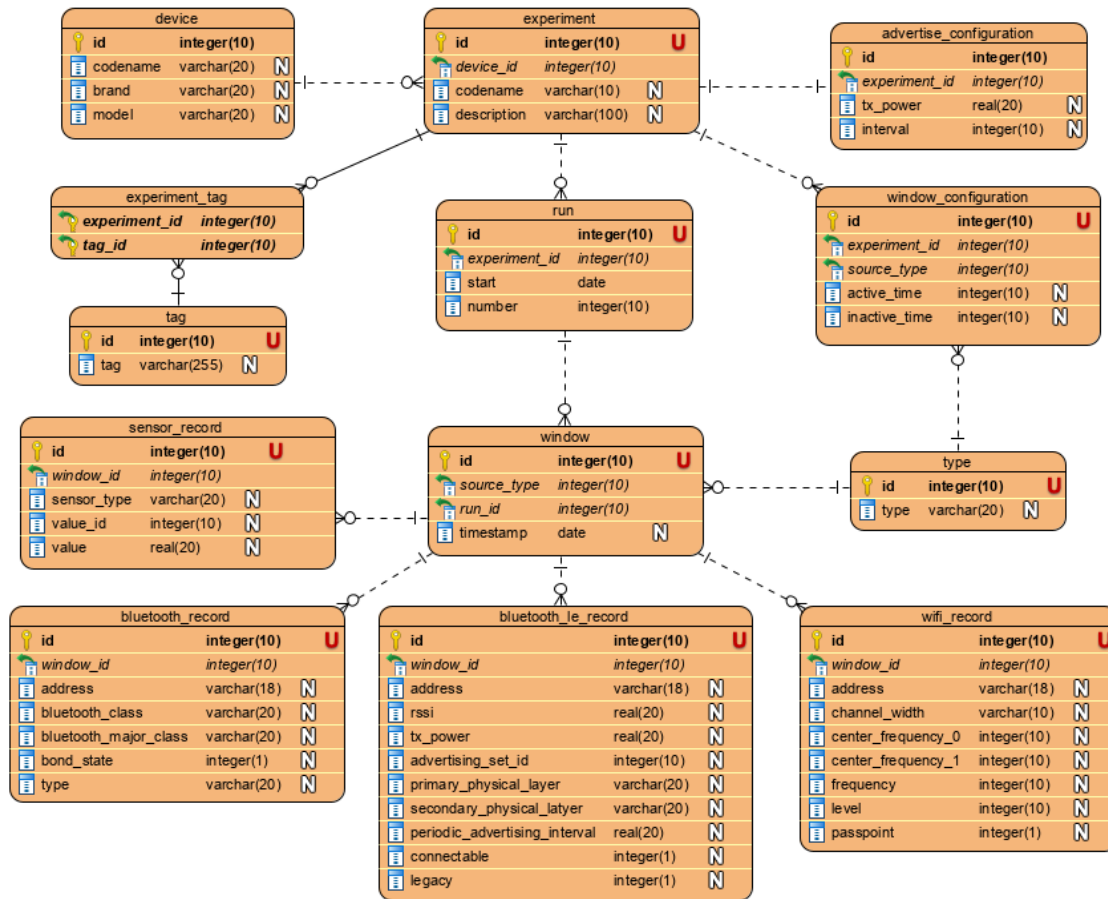


Figura 20: Diagrama de Tablas implementado en la base de datos relacional. Fuente: Elaboración propia

Para facilitar la implementación de la base de datos con sus interfaces para guardar y consultar información se utilizó una librería llamada Room [27 - 28], que permite mediante codificación en lenguaje Java [41] describir las entidades, cómo se relacionan y las interfaces con las que se acceden a los datos mediante consultas SQL embebidas. También es necesario describir el esquema en una clase dedicada a tal fin, y es esta la que se utiliza para obtener los objetos de acceso a datos para cualquier tipo de consulta que se requiera realizar.

Descripción del proceso de ejecución de un Experimento

Partiendo de la premisa que los experimentos se programan como ejecuciones a futuro y que las mismas deben llevarse a cabo aún cuando la aplicación se encuentra cerrada (su proceso no fue iniciado o no está en primer plano) se planteó un diseño utilizando el gestor de alarmas [40] y los servicios en primer plano [16] que provee el sistema operativo Android. A continuación, en la Figura 21, se representa el comportamiento a alto nivel de la corrida de un experimento con un diagrama de secuencias.

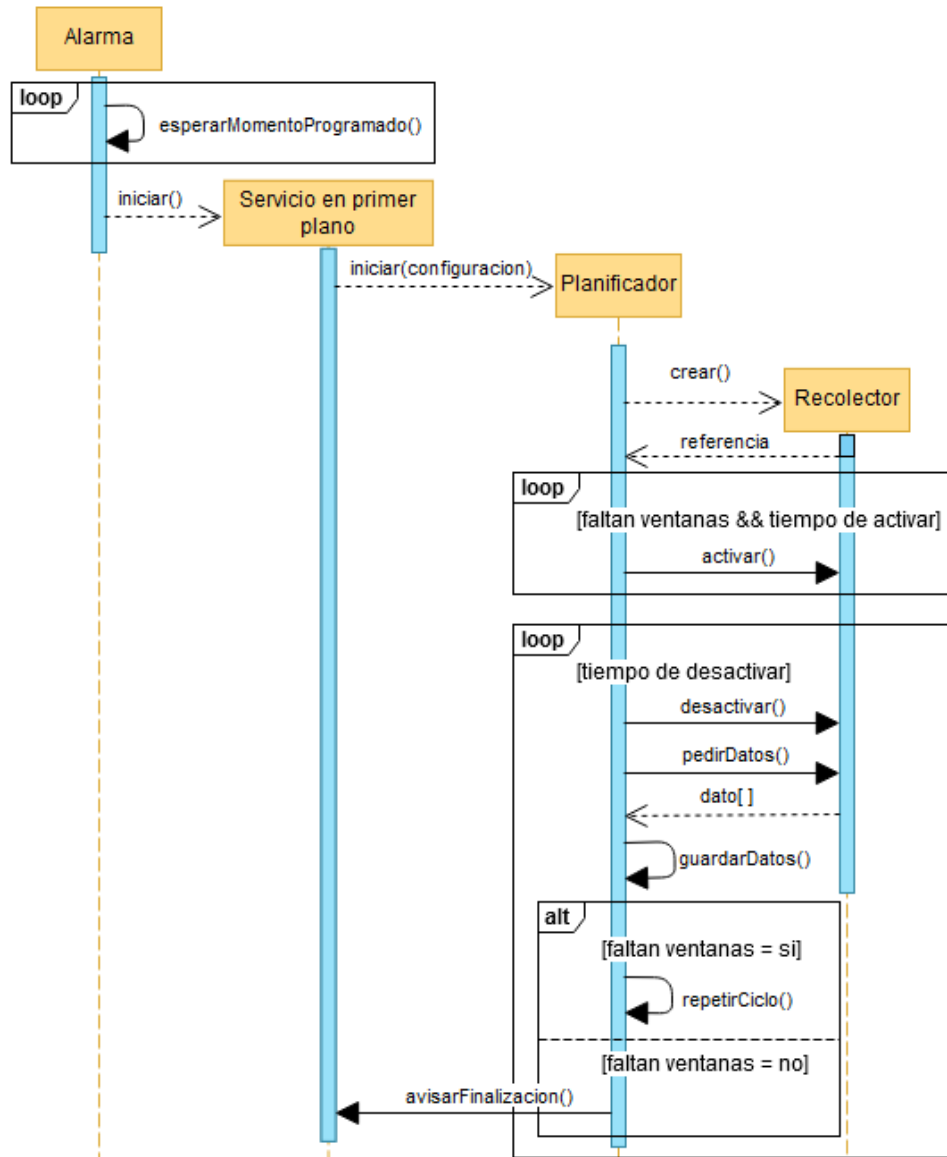


Figura 21: Diagrama de secuencia en alto nivel del comportamiento esperado para la recolección de datos. Fuente: Elaboración propia

Concepto de Ventana

Dentro de la aplicación los planificadores trabajan en *ventanas de tiempo*, las mismas están definidas por lo que es el *tiempo activo* y el *tiempo inactivo* de trabajo (Figura 22). Durante cada período *activo*, el recolector y/o transmisor está trabajando, es decir recolectando o transmitiendo muestras. Por el contrario, en el período *inactivo*, el recolector y/o el transmisor no están ejecutando ninguna tarea.

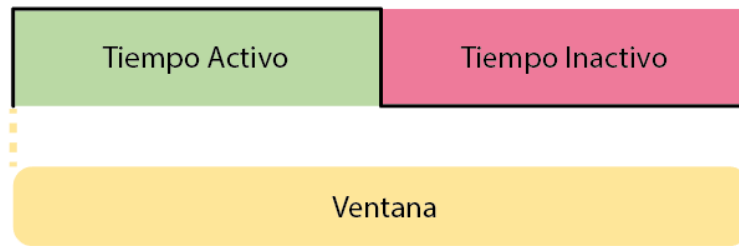


Figura 22: Representación del concepto de una ventana. Fuente: Elaboración propia

De esta manera, mientras transcurre el tiempo activo de la ventana cada recolector puede obtener una o más muestras del tipo de dato que le corresponde. Algo similar sucede con los transmisores, estos pueden enviar una o más muestras de su tipo de dato durante su período activo de trabajo.

Módulos de recolección

Para los módulos de recolección del sistema se plantean dos componentes esenciales en el diseño:

- *Schedulers* (Planificadores): Son los encargados de interpretar la configuración de ventana y en base a esta controlar en qué instantes se activa o desactiva la recolección y por cuántos ciclos se realizarán estas acciones.
- *Data Buckets* (Recolectores): Estos componentes se encargan de consultar las interfaces del sistema operativo para obtener información de los diferentes periféricos de comunicación y sensores. Se mantienen trabajando durante el transcurso del tiempo activo indicado por los *Schedulers* y los datos que recolectan se almacenan temporalmente en memoria hasta que se entra en el intervalo de tiempo inactivo, en ese momento los *Schedulers* indican que los datos se escriban a la base de datos mediante un proceso asíncrono para evitar solapamientos de ventanas.

A partir del diagrama de la Figura 21 se puede profundizar en la estructura de software que se implementó con estos dos componentes que mencionamos recién, para facilitar la visualización se construyó un diagrama de clases para comprender qué tipo de relaciones hay y qué otros elementos ocupan un lugar en el proceso de recolección, este diagrama está representado de manera simplificada en la Figura 23 a continuación.

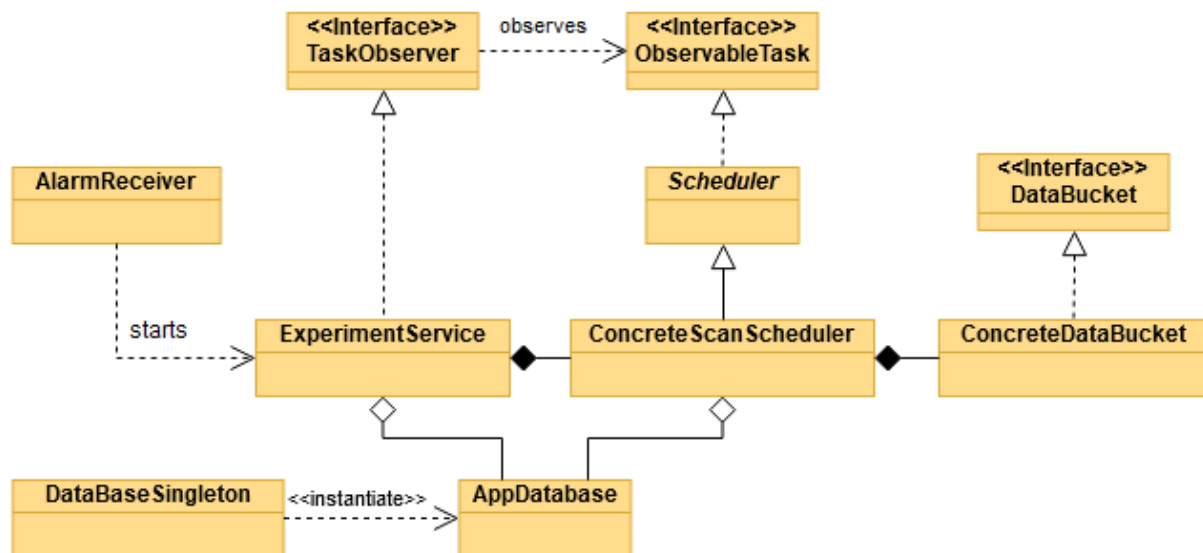


Figura 23: Diagrama de clases simplificado para la recolección de datos. Fuente: Elaboración propia

En la figura anterior, se visualizan elementos etiquetados bajo los nombres de **ConcreteScanScheduler** y **ConcreteDataBucket**, estos representan clases concretas que implementan las funcionalidades de un **Scheduler** y un **DataBucket** respectivamente, pueden referirse a recolectores de datos de redes Wi-Fi, dispositivos Bluetooth, dispositivos Bluetooth Low Energy o bien de Sensores.

Es importante notar la presencia de las interfaces que implementan los **Schedulers** y el servicio de experimentos. Las interfaces **TaskObserver** y **ObservableTask** permiten implementar un patrón *Observer* como el que se describió en el marco teórico del presente documento, de esta manera cuando un recolector termina su tarea puede notificar al servicio de experimentos y este último actualizar su mapa de estados de cada una de las tareas que están corriendo.

Finalmente cuando todos los recolectores notifican su finalización, el servicio de experimentos inicia la exportación de datos a archivos y posteriormente finaliza su actividad.

Recolectores de datos Wi-Fi y Bluetooth

Basado en la documentación de Android sobre **BroadcastReceivers** [19] y las guías para desarrolladores en materia de operaciones sobre los adaptadores Wi-Fi [35] y Bluetooth para escaneos clásicos [36], se implementaron los recolectores para estos tipos de datos siguiendo la estructura de la Figura 24.

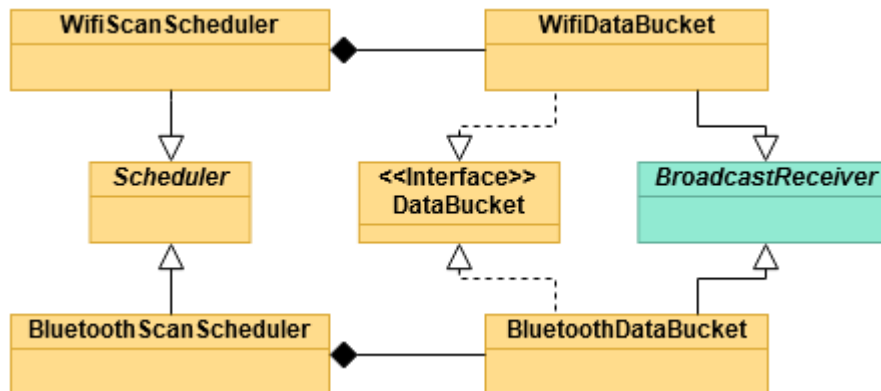


Figura 24: Diagrama de clases simplificado de los recolectores Wi-Fi y Bluetooth. Fuente: Elaboración propia

Para poder obtener los datos de redes Wi-Fi es necesario registrar en el sistema operativo el *BroadcastReceiver* correspondiente, particularmente para que capture mensajes del tipo `SCAN_RESULTS_AVAILABLE_ACTION` [37]. Estos mensajes de *broadcast* son emitidos por el sistema operativo cada vez que el adaptador de Wi-Fi termina de realizar un escaneo y expone en su interfaz una lista con los resultados, la cual se itera para obtener la información y así poder generar los objetos de transferencia de datos (*DataTransferObjects*) que posteriormente se utilizan para realizar las inserciones en la base de datos mediante los *Repositories*.

En el caso del recolector de datos de dispositivos Bluetooth Clásico es necesario registrar el *BroadcastReceiver* para capturar mensajes de tipo `ACTION_FOUND` [38]. Mensajes de este tipo son emitidos por el sistema operativo cada vez que se descubre un nuevo dispositivo Bluetooth en el entorno. Para cada dispositivo descubierto se procesa su información y se crea el correspondiente objeto de transferencia de datos que será encolado finalmente en una lista para su posterior inserción en la base de datos de manera análoga a los datos de Wi-Fi.

La Figura 25 describe de manera simplificada el comportamiento de estos recolectores ante la recepción de mensajes de *broadcast* emitidos por el sistema operativo.

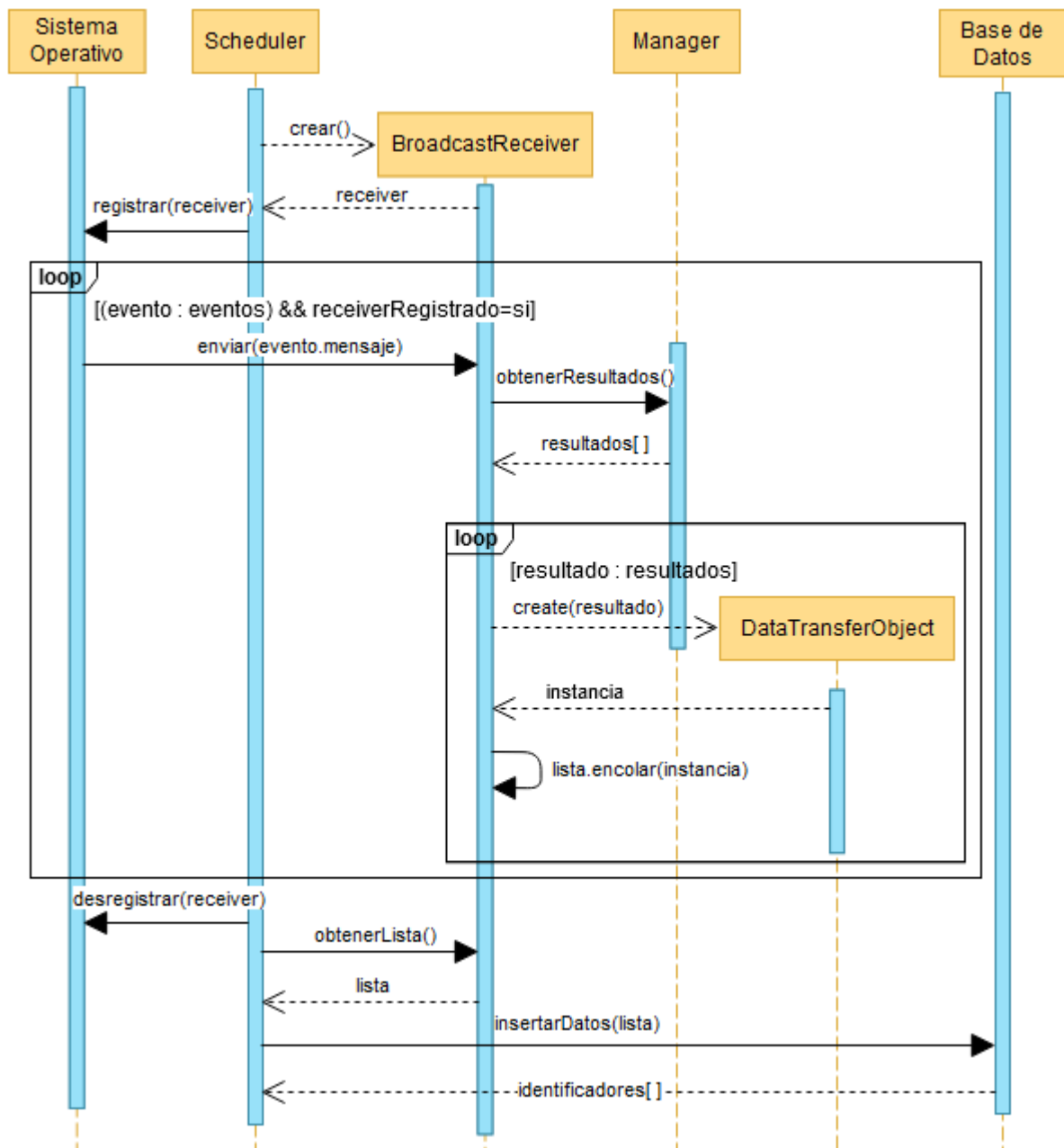


Figura 25: Comportamiento de recolectores que implementan la interfaz BroadcastReceiver. Fuente: Elaboración propia

Nota: Los objetos de transferencia de datos asociados en cada caso son WifiRecord y BluetoothRecord donde cada instancia de estos representa un registro de datos del tipo que describe su nombre. Particularmente lo que se inserta en la base de datos a través de los repositorios son estas instancias de las clases entidades que conforman el esquema de la base de datos.

Recolector de datos Bluetooth Low Energy

Tomando como referencia la documentación referida al manejo de adaptadores Bluetooth Low Energy descrita en [9] y construyendo un esquema análogo a los recolectores anteriores, este se implementó con el esquema del diagrama de la Figura 26.

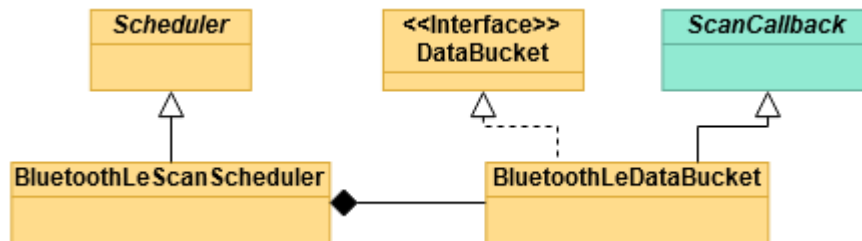


Figura 26: Diagrama de clases simplificado del recolector Bluetooth Low Energy. Fuente: Elaboración propia

De forma similar a los recolectores implementados con `BroadcastReceivers`, este utiliza una clase que extiende a `ScanCallback` [39] para procesar los datos de cada nuevo dispositivo descubierto.

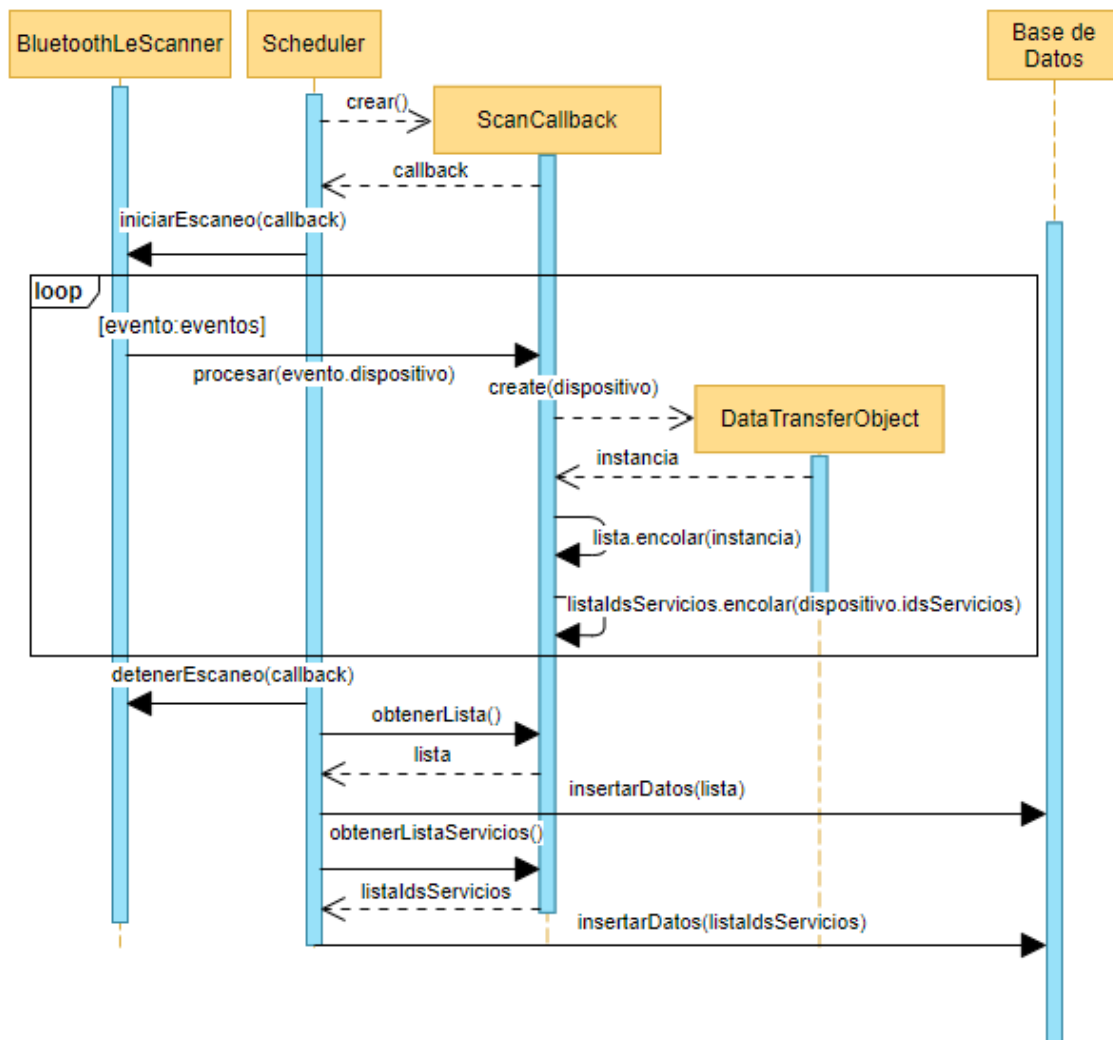


Figura 27: Comportamiento del recolector Bluetooth Low Energy. Fuente: Elaboración propia

La particularidad de los escaneos de Bluetooth Low Energy es que resulta de interés conocer cuáles son los servicios que publica cada dispositivo, o al menos conocer los identificadores de estos. Por lo tanto se añadió un paso extra (observable en el comportamiento descrito por la Figura 27) en el procesamiento que está relacionado con la iteración sobre los diferentes identificadores que porta el resultado del dispositivo descubierto y la generación de los objetos de transferencia de datos asociados a estos identificadores.

Recolector de datos de Sensores

Para este recolector se tomó como base la documentación disponible en [12] que describe cómo implementar lectura de Sensores disponibles en el dispositivo y se lo adaptó al diseño con *Schedulers* y *DataBuckets* de manera similar a los demás recolectores. El diseño final de este recolector se encuentra representado por la Figura 28.

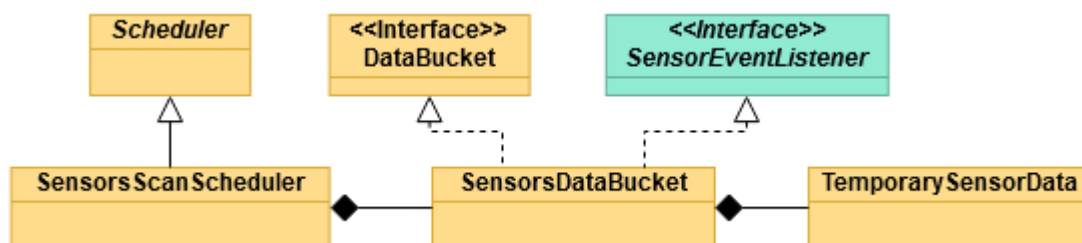


Figura 28: Diagrama de clases simplificado del recolector de Sensores. Fuente: Elaboración propia

Debido a la manera en la que se comportan los sensores y la interfaz que brinda el sistema operativo para poder capturar los valores es que se implementó un componente de valor temporario de sensor, este se encarga de ir registrando cada cambio del valor de un sensor de manera que posteriormente, al momento de consultar por esta información se obtenga el dato con el valor del último cambio que experimentó.

El comportamiento descrito anteriormente se puede ver reflejado en el diagrama de secuencia diseñado para este recolector. Además, en el diagrama de la Figura 29 también puede observarse que a diferencia de los demás recolectores, este utiliza un mapa donde las claves son el tipo de sensor y el valor es una instancia de *TemporaryDataSensor*. Esta implementación permite que a la hora de procesar un cambio disparado por un evento, se pueda filtrar la búsqueda de qué valor temporal se debe actualizar mediante el tipo de sensor como clave.

La implementación del registro mediante un mapa, también asegura que solo se capturan y/o actualizan cambios de sensores si y sólo si están disponibles en el dispositivo.

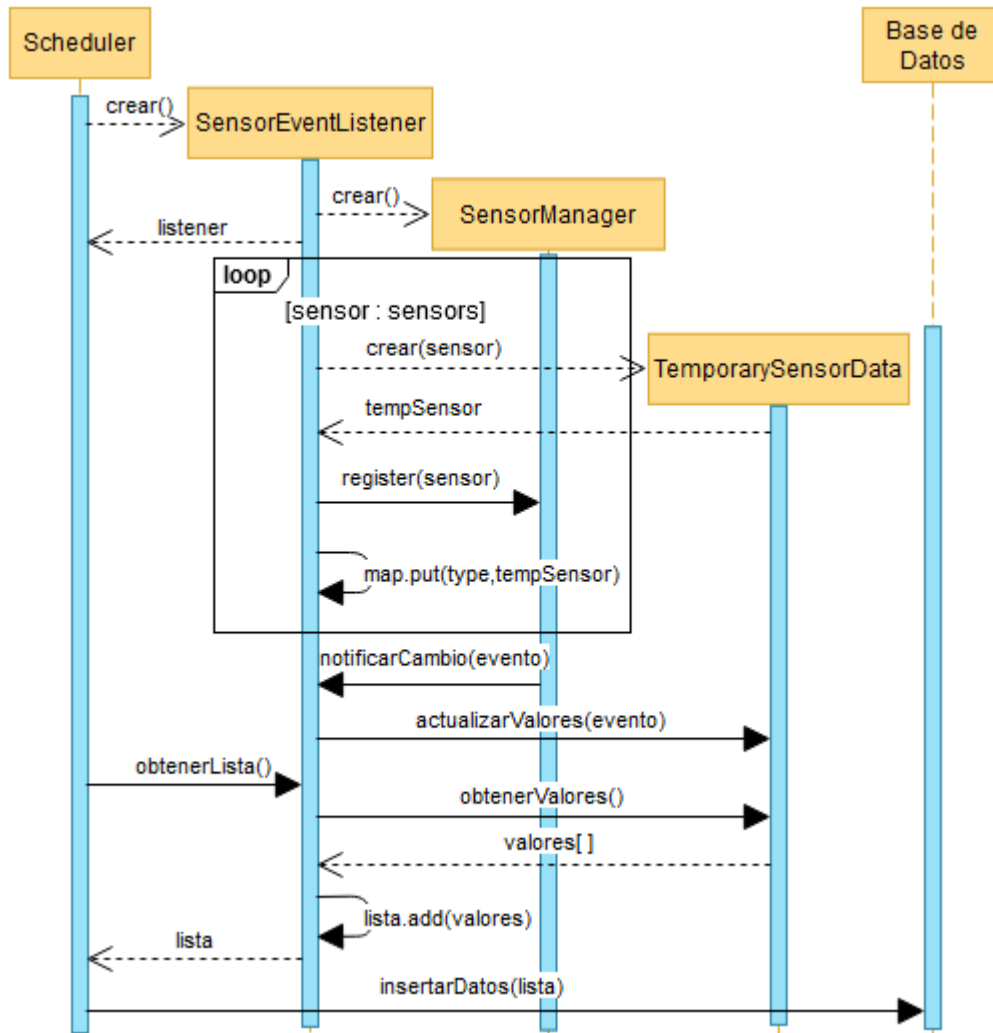


Figura 29: Comportamiento del recolector de Sensores. Fuente: Elaboración propia

Módulo de transmisión (Baliza BLE)

Para realizar la publicación del dispositivo de manera que pueda ser descubierto por otros mediante Bluetooth Low Energy, se diseñó un esquema que utiliza un Scheduler para activar y desactivar la transmisión de manera análoga a los recolectores, con la diferencia de que en este módulo no es necesario guardar datos de la información transmitida ya que siempre es la misma.

El módulo de transmisión requiere dos tipos de configuraciones, una es la de ventana y la otra es la que contiene información referida a la potencia con la que va a transmitir y el intervalo entre mensajes de publicidad.

Con estas consideraciones en mente, se diagramaron las clases quedando un esquema como el representado por la Figura 30.

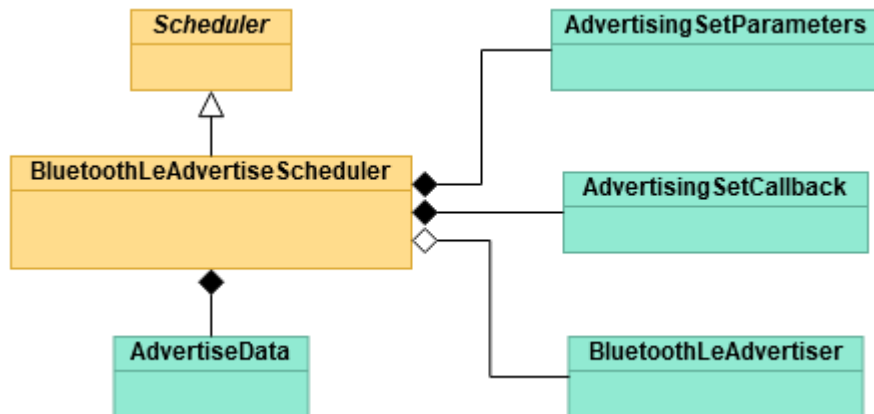


Figura 30: Diagrama de clases del módulo para transmitir en modo Baliza Bluetooth Low Energy. Fuente: Elaboración propia

Como referencia para el diseño se siguieron las guías [44 - 45] que describen el proceso de implementación de funcionalidades de *advertising* sobre Bluetooth Low Energy

Módulos de exportación de datos

Para los módulos de exportación se implementó una clase abstracta **FileWriter** que se encarga de realizar de manera asíncrona las tareas de escritura de datos en archivos en la memoria externa del dispositivo. Esto fue diseñado así para no bloquear los hilos principales de la aplicación durante este proceso y que de esta manera pueda continuar operando normalmente con otras tareas como por ejemplo, recolecciones programadas.

Como se aprecia en la Figura 31 se implementaron dos clases abstractas de nombre **CsvFileWriter** y **JsonFileWriter** que se encargan de la manipulación de datos CSV [43] y JSON respectivamente. Cada **FileWriter** mencionado opera sobre objetos que implementen las interfaces **CsvExportable** y **JsonExportable**, las cuales proveen métodos para interpretar los objetos bajo los formatos correspondientes, de esta manera los **FileWriters** conocen cómo deben escribir cada instancia de un objeto en el archivo de texto.

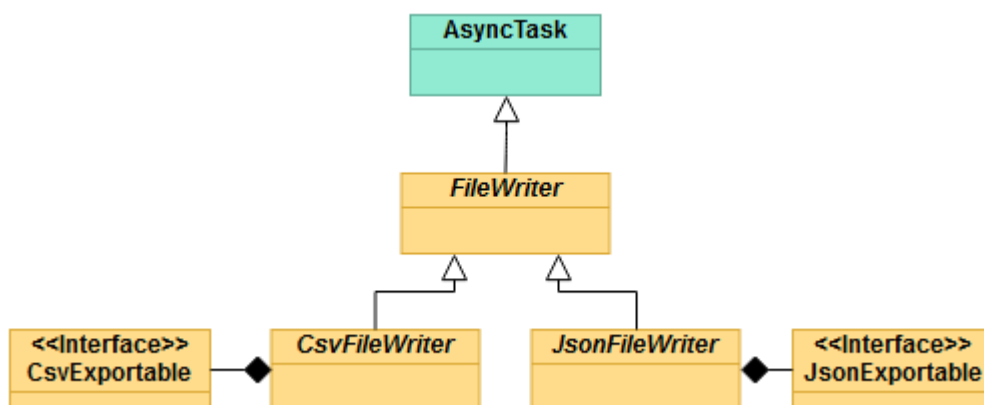


Figura 31: Diagrama de clases de módulos para escritura de archivos CSV y JSON. Fuente: Elaboración propia

Cada clase concreta que extienda a `CsvFileWriter` o bien a `JsonFileWriter` debe implementar los métodos para definir nombre de archivo, la ruta en la que se va a escribir el archivo y que datos serán escritos al mismo.

Particularmente las clases que implementan la interfaz `JsonExportable` utilizan la librería `GSON` [42] para convertir objetos a cadenas de caracteres que los representan en formato JSON.

Finalmente el último módulo que se implementó es para la obtención de códigos QR. Este se encarga de obtener una representación en mapa de bits para un objeto de manera que pueda ser mostrado como una imagen de código QR.

Para tener mayor flexibilidad y recurriendo a la reusabilidad de código se tomó como interfaz el formato JSON para codificar los objetos en `Strings` y luego estas cadenas en mapas de bits. La estructura de este módulo se ve representada por la Figura 32.

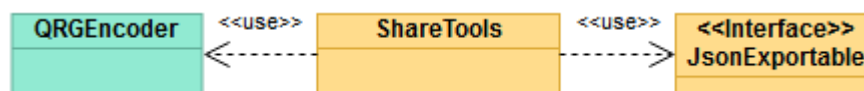


Figura 32: Estructura del módulo para manejo de códigos QR. Fuente: Elaboración propia

La clase `ShareTools` implementa métodos estáticos para las conversiones y actualmente se utiliza al momento de generar un código QR para compartir experimentos o bien para interpretar la cadena de caracteres resultante de escanear un código QR. Esta interpretación se utiliza principalmente para autocompletar los campos de configuración de un experimento nuevo.

Interfaces gráficas de usuario

A partir de los requerimientos y del diálogo con el equipo de investigación se desprenden los siguientes posibles casos de uso representados por la Figura 33.

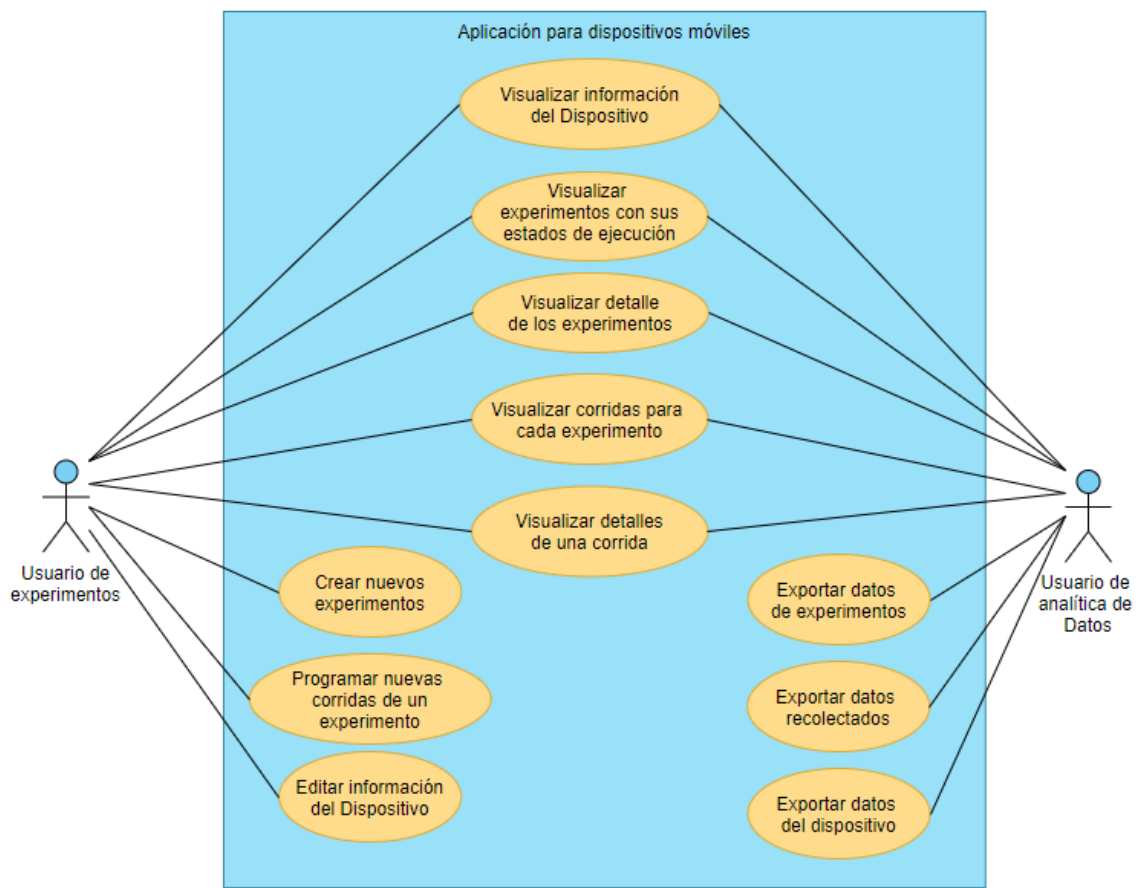


Figura 33: Casos de uso de la aplicación para dispositivos móviles. Fuente: Elaboración propia

Dados estos casos de uso, se propuso un flujo de utilización de la aplicación que resulte intuitivo para el usuario. Este flujo se ve representado en el diagrama de la Figura 34 que se muestra a continuación.

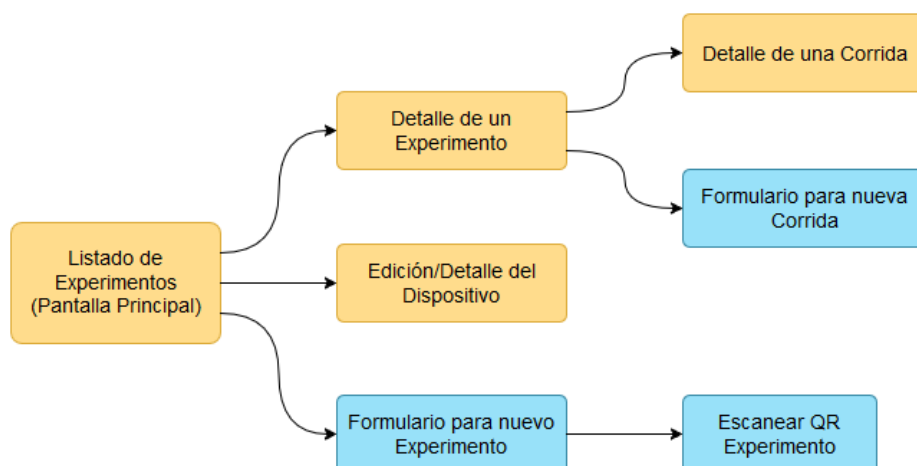


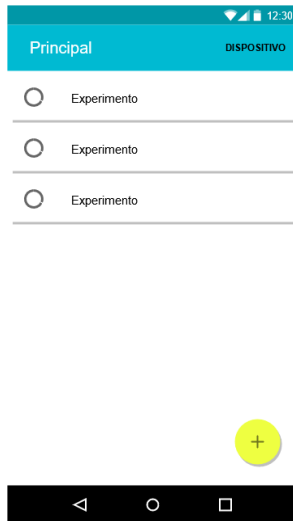
Figura 34: Flujos de usuario para interacción con la interfaz gráfica de la aplicación. Fuente: Elaboración propia

Cada uno de los bloques de la figura anterior tiene asociado una vista para que el usuario pueda interactuar con la aplicación.

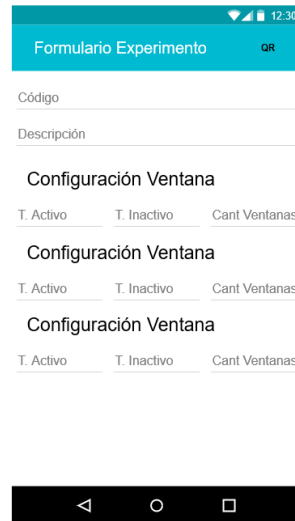
Detalle de vistas diseñadas

A continuación se describen los diseños de cada una de las vistas y sus funcionalidades:

- La pantalla principal (Figura 35.a) muestra un listado de experimentos donde cada elemento lleva a la vista de detalle del mismo, un botón que permite acceder al formulario para crear un nuevo experimento y finalmente un botón que da acceso a la vista para ver o modificar información del dispositivo.
- La vista de creación de nuevos experimentos es la que se ve representada por la Figura 35.b y tiene una sección dedicada para cargar información particular sobre el experimento, como así también las configuraciones de cada tipo de ventana para los diferentes recolectores o transmisores.
- La Figura 35.c representa la vista para el detalle de un experimento, en la cual se puede visualizar el código QR que codifica ese experimento, la descripción, las etiquetas, las configuraciones asociadas de cada recolector o transmisor, también se ve el listado de corridas programadas para ese experimento donde cada elemento de esta lista de acceso a la vista de detalle de la corrida seleccionada, finalmente también tiene un botón para programar nuevas corridas.
- En la vista de detalle de una corrida se pueden observar las cantidades de muestras recolectadas para cada tipo de datos del experimento como se aprecia en la Figura 35.d. También cuando una corrida se está ejecutando se visualiza una barra de progreso indefinido en la parte superior, la cual desaparece cuando el proceso de ejecución finaliza.
- En la imagen de la Figura 35.e está representado el diseño para la vista de programación de nuevas corridas, donde se puede seleccionar la fecha y hora de inicio.
- La Figura 35.f representa el esquema para la vista de escaneo de códigos QR de experimentos, a esta se accede desde un botón disponible en la vista del formulario de creación de experimentos (botón *QR* en la Figura 35.b). Una vez escaneado el código la aplicación devolverá al usuario a la vista del formulario autocompletando los campos.
- Finalmente, la Figura 35.g corresponde a la vista de detalle o edición de información del dispositivo, donde el usuario puede cambiar el código del dispositivo o generar un nuevo identificador único.



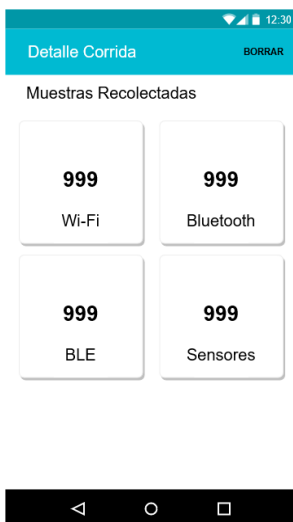
(a)



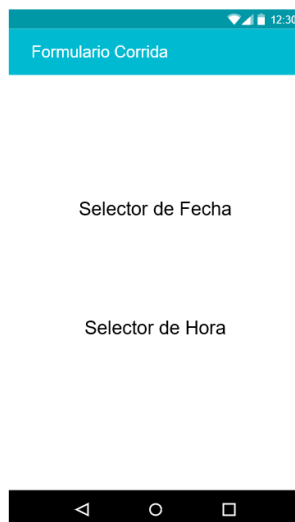
(b)



(c)



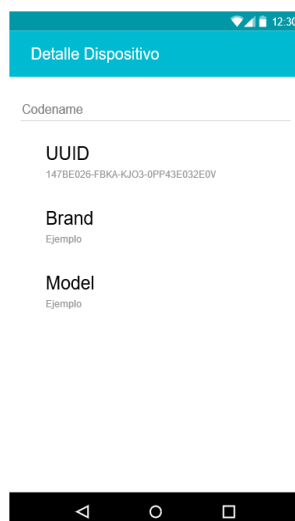
(d)



(e)



(f)



(g)

Figura 35: Secuencia de vistas de la aplicación. Fuente: Elaboración propia

Pruebas

Como metodología para verificar las funcionalidades de cada módulo de la aplicación como así también del modelo de datos, se realizaron pruebas unitarias y de integraciones automatizadas, que además están relacionadas a los requerimientos de manera que se pueda asegurar que la aplicación cumple con estos.

Dado que todos los componentes de la aplicación requieren de Hardware o de características particulares del sistema operativo, todas las pruebas son del tipo *instrumentadas*, lo que quiere decir que se compilan en el equipo de desarrollo pero se ejecutan y evalúan sus resultados en un dispositivo con Android que puede ser real o emulado.

Para implementar las pruebas automatizadas se utilizó la librería JUnit [47] en su versión 4, para correr las pruebas y generar reportes se utilizaron las herramientas que incluye Android Studio junto a JaCoCo [48] para la generación de reportes de cobertura de código.

En la Tabla 7 se pueden observar las clases implementadas con sus respectivas descripciones. Para mayor detalle puede consultarse el **Anexo A: Detalle de pruebas**.

Las clases se marcaron con colores de acuerdo a las tareas que realizan dentro de la aplicación como indica la Tabla 6 de referencia.

Tabla 6: Referencia de colores para clases de pruebas. Fuente: Elaboración propia

| Color | Tarea |
|-------|---------------------------------------|
| | Transmisión de datos |
| | Escritura de archivos |
| | Operaciones sobre modelo de datos |
| | Recolección de datos |
| | Operaciones relacionadas a códigos QR |

Tabla 7: Lista de clases con descripciones de pruebas implementadas. Fuente: Elaboración propia

| Clase | Descripción |
|-----------------------------------|---|
| BluetoothLeAdvertiseSchedulerTest | Verifica la instanciación y correcta notificación a los observadores del transmisor BLE |
| BluetoothCsvFileWriterTest | Verifican la instanciación y los métodos para escribir archivos con información de los diferentes tipos de muestras recolectados por la aplicación. |
| BluetoothLeCsvFileWriterTest | |
| SensorCsvFileWriterTest | |
| WifiCsvFileWriterTest | |
| JsonFileWriterTest | Verifica la instanciación y los métodos para escribir archivos con información de experimentos en formato JSON |
| BluetoothLeRepositoryTest | Verifican las operaciones de creación, modificación, eliminación y consulta de las diferentes entidades que pertenecen al modelo de datos implementado para la aplicación |
| BluetoothRepositoryTest | |
| ConfigurationRepositoryTest | |
| DeviceRepositoryTest | |
| ExperimentRepositoryTest | |
| RunRepositoryTest | |
| SensorRepositoryTest | |
| SourceTypeRepositoryTest | |
| WifiRepositoryTest | |
| WindowRepositoryTest | |
| BluetoothLeScanSchedulerTest | Verifican la instanciación y correcta notificación a los observadores de los recolectores de diferentes tipos de datos. |
| BluetoothScanSchedulerTest | |
| SensorsScanSchedulerTest | |
| WifiScanSchedulerTest | |
| ExperimentRepresentationTest | Verifican la instanciación y métodos provistos por las herramientas para compartir información mediante códigos QR |
| ShareToolsTest | |

En la Figura 36 puede observarse el resultado del reporte generado por Android Studio correspondiente a la última corrida de todas las pruebas implementadas.

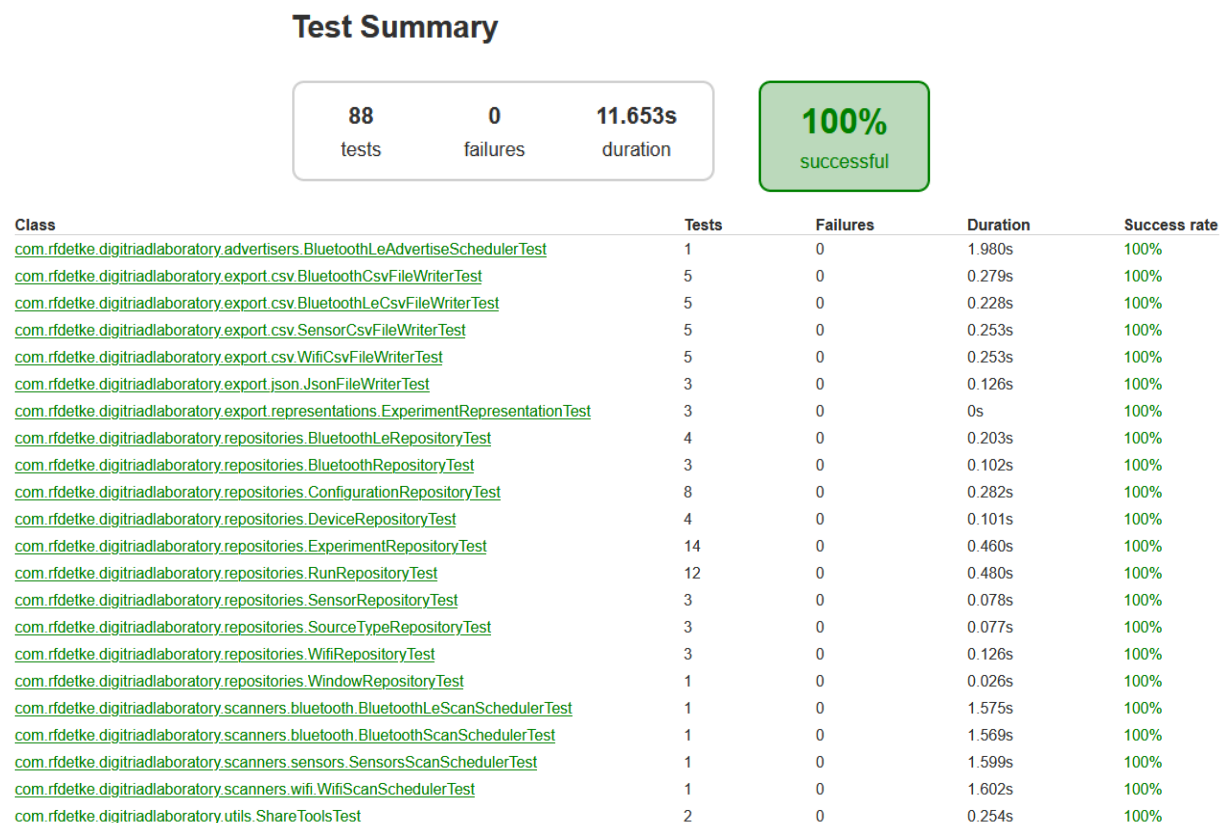


Figura 36: Captura del último reporte de corrida de pruebas. Fuente: Elaboración propia

En la Figura 37 se muestra el detalle de la cobertura de código alcanzada por las pruebas implementadas. Como se aprecia, en el caso de los *Scanners* existen ramas que no se ejecutaron, esto se debe a que gran parte del código de estos módulos depende de las señales que captura el dispositivo al momento de ejecutar la corrida de las pruebas.

app

| Element | Missed Instructions | Cov. | Missed Branches | Cov. |
|--|------------------------|------|------------------------|------|
| com.rfdetke.digitriadlaboratory.scanners.wifi | <div><div></div></div> | 94% | <div><div></div></div> | 62% |
| com.rfdetke.digitriadlaboratory.scanners | <div><div></div></div> | 95% | <div><div></div></div> | 75% |
| com.rfdetke.digitriadlaboratory.scanners.bluetooth | <div><div></div></div> | 98% | <div><div></div></div> | 81% |
| com.rfdetke.digitriadlaboratory.export.representations | <div><div></div></div> | 98% | <div><div></div></div> | 65% |
| com.rfdetke.digitriadlaboratory.utils | <div><div></div></div> | 96% | | n/a |
| com.rfdetke.digitriadlaboratory.repositories | <div><div></div></div> | 100% | | n/a |
| com.rfdetke.digitriadlaboratory.scanners.sensors | <div><div></div></div> | 100% | <div><div></div></div> | 94% |
| com.rfdetke.digitriadlaboratory.export.csv | <div><div></div></div> | 100% | <div><div></div></div> | 100% |
| com.rfdetke.digitriadlaboratory.export.json | <div><div></div></div> | 100% | | n/a |
| com.rfdetke.digitriadlaboratory.advertisers | <div><div></div></div> | 100% | | n/a |
| Total | 28 of 2.095 | 98% | 19 of 84 | 77% |

Figura 37: Captura del último reporte de cobertura de código. Fuente: Elaboración propia

Capítulo 4: Resultados

Como prueba de uso de la aplicación se realizó un experimento con dos dispositivos, se ejecutaron dos corridas, una con 1 metro de separación y otra con 40 centímetros de separación. Ambos dispositivos estaban ubicados sobre una mesa de vidrio y se programaron las corridas de manera que comiencen simultáneamente en estos. Es importante mencionar que los dispositivos son del mismo fabricante pero de diferentes gamas, lo que puede llevar a observarse diferencias en las mediciones capturadas por cada uno.

La configuración de ventanas y de transmisión para el experimento se puede ver en la Tabla 8 a continuación.

Tabla 8: Configuración de experimento

| Módulo | T. Activo[s] | T. Inactivo[s] | Ventanas | Potencia Tx[dB] | Intervalo Tx[ms] |
|----------------------|--------------|----------------|----------|-----------------|------------------|
| Recolector Wi-Fi | 20 | 40 | 10 | - | - |
| Recolector Bluetooth | 20 | 40 | 10 | - | - |
| Recolector BLE | 20 | 40 | 10 | - | - |
| Recolector Sensores | 1 | 1 | 300 | - | - |
| Transmisor BLE | 2 | 3 | 120 | -3 | 100 |

En la aplicación estos datos se verían cargados como se muestra en la Figura 38.

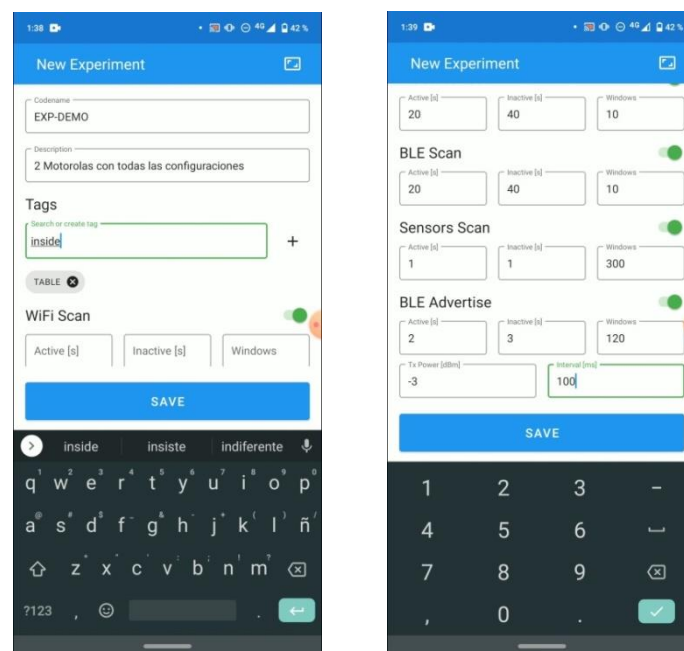


Figura 38: Captura de pantalla de configuración de un experimento

Esta misma configuración es exportada desde la aplicación en cada dispositivo a un archivo en formato JSON como el de la Figura 39 con el formato de nombre **X-
<experiment_codename>_<device_codename>.json**, además contiene información sobre el dispositivo como su nombre en código, modelo, marca e identificador único (UUID). Esta información es para identificar al dispositivo cuando se analizan los datos y poder asociarlo a estos, como se mostrará más adelante, fue de utilidad en el procesamiento que se realizó sobre los datos recolectados.

```
{
  "bluetooth": {
    "inactive": 40,
    "active": 20,
    "windows": 10
  },
  "bluetoothLe": {
    "inactive": 40,
    "active": 20,
    "windows": 10
  },
  "bluetoothLeAdvertise": {
    "inactive": 3,
    "tx_power": -3,
    "active": 2,
    "interval": 100,
    "windows": 120
  },
  "codename": "EXP-DEMO",
  "description": "2 Motorolas",
  "device": {
    "codename": "E6",
    "model": "MOTO E(6) PLUS",
    "brand": "MOTOROLA",
    "uuid": "e28ebef8-b05b-48bb-9106-3928cb5b39d6"
  },
  "sensors": {
    "inactive": 1,
    "active": 1,
    "windows": 300
  },
  "tags": [
    "TABLE",
    "INSIDE"
  ],
  "wifi": {
    "inactive": 40,
    "active": 20,
    "windows": 10
  }
}
```

Figura 39: Captura del archivo de configuración de experimento. Fuente: Elaboración propia

Estos datos del experimento, desde la aplicación se pueden encontrar en la pantalla de detalle tal y como representa la Figura 40.



Figura 40: Captura de pantalla para el detalle del experimento configurado

Cuando una corrida está en ejecución se pueden observar la cantidad de registros de cada tipo en la pantalla de detalle de corrida, la Figura 41 muestra estos valores para los dispositivos E6 (izquierda) y G7 (derecha) al final de la corrida número 1 del experimento

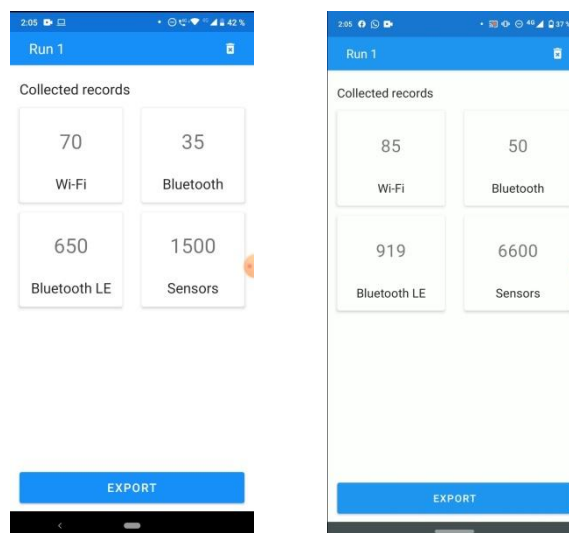


Figura 41: Captura de pantalla para detalles de la corrida 1 para cada dispositivo (E6 y G7)

Al final de la ejecución se descargaron de la memoria interna no volátil de cada teléfono los archivos exportados que contienen las muestras de las corridas, se prepararon algoritmos en Python (para más detalles revisar **Anexo C: Algoritmos para graficar datos de prueba**) los cuales leen estos archivos y procesan los datos para construir algunos gráficos. En las siguientes secciones se detallan cada uno de los gráficos obtenidos.

Bluetooth Low Energy

Para los datos recolectados sobre Bluetooth Low Energy, cada archivo cuenta con las siguientes columnas:

- **record_timestamp**: instante de tiempo en el que se descubrió el mensaje de *advertisement*.
- **run_id**: identificador de la corrida
- **window_id**: identificador de la ventana.
- **address**: dirección aleatoria del dispositivo transmisor del mensaje de *advertisement*.
- **rssi**: nivel de intensidad de señal al recibir el mensaje.
- **tx_power**: potencia con la que se transmitió el mensaje.
- **advertising_set_id**: si el mensaje forma parte de un *set* de mensajes de *advertisement* en este campo se completa el identificador del *set*.
- **primary_physical_layer**: capa física primaria utilizada para transmitir el mensaje.
- **seconary_physical_layer**: capa física secundaria utilizada para transmitir el mensaje.
- **periodic_advertising_interval**: si la transmisión se configuró como *periódica*, en este campo se encuentra el intervalo de periodicidad de la transmisión.
- **connectable**: indica si el dispositivo es conectable (**1**) o no (**0**).
- **legacy**: indica si el dispositivo transmite en modo *legacy* (**1**) o no (**0**).
- **uuid**: especifica el identificador único del dispositivo transmisor o identificador de servicio ofrecido por el dispositivo transmisor.

Para estos datos se optó por graficar, en primera instancia los niveles de RSSI conforme transcurría el experimento, luego la variabilidad de los valores de RSSI en relación a la distancia (1m y 0,4m) y por último histogramas de los valores de RSSI capturados por cada dispositivo y para cada distancia. En todos los casos se grafican únicamente las señales correspondientes a dispositivos involucrados en el experimento, es decir que los valores capturados por E6 corresponden a los transmitidos por G7 y viceversa.

Para el caso del dispositivo etiquetado como E6, se observan en la Figura 42 los valores de RSSI a 0,4m en azul y a 1m en naranja. Nótese que los cambios drásticos en las curvas son debido a los inicios y finales de períodos activos de las ventanas.

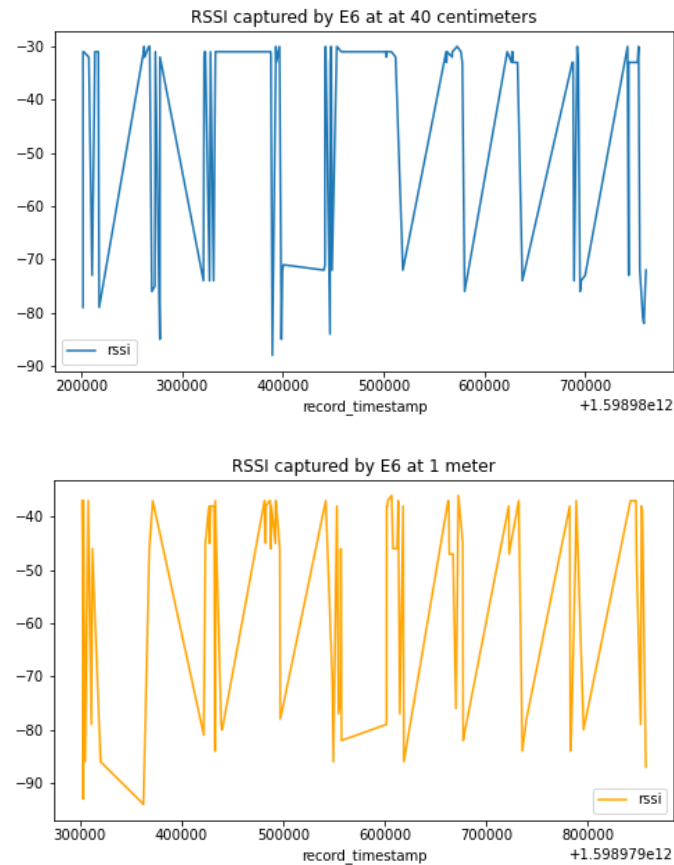


Figura 42: RSSI de las señales capturadas por el dispositivo E6. Fuente: Elaboración propia

Algo similar sucedió con las señales capturadas por el dispositivo G7 como se puede apreciar en la Figura 43.

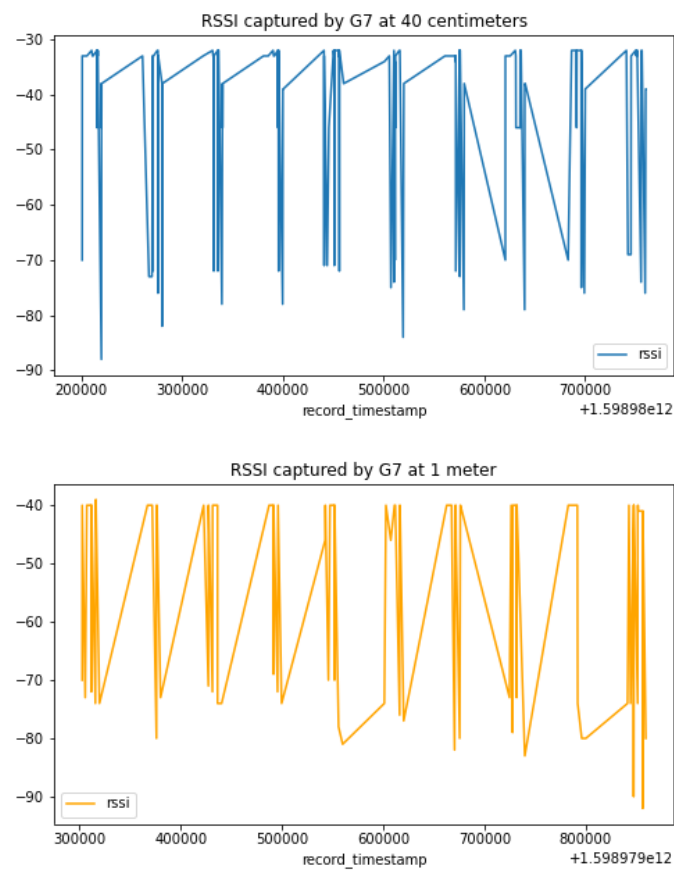


Figura 43: RSSI de las señales capturadas por el dispositivo G7. Fuente: Elaboración propia

A continuación, en la Figura 44, se muestran los histogramas elaborados para las señales capturadas por E6. En la Figura 45 se observan los gráficos análogos para el caso de los datos capturados por G7. Se desconoce el porqué de estas distribuciones, pero puede deberse a cómo se propagan las señales en el ambiente en el que se realizaron las corridas.

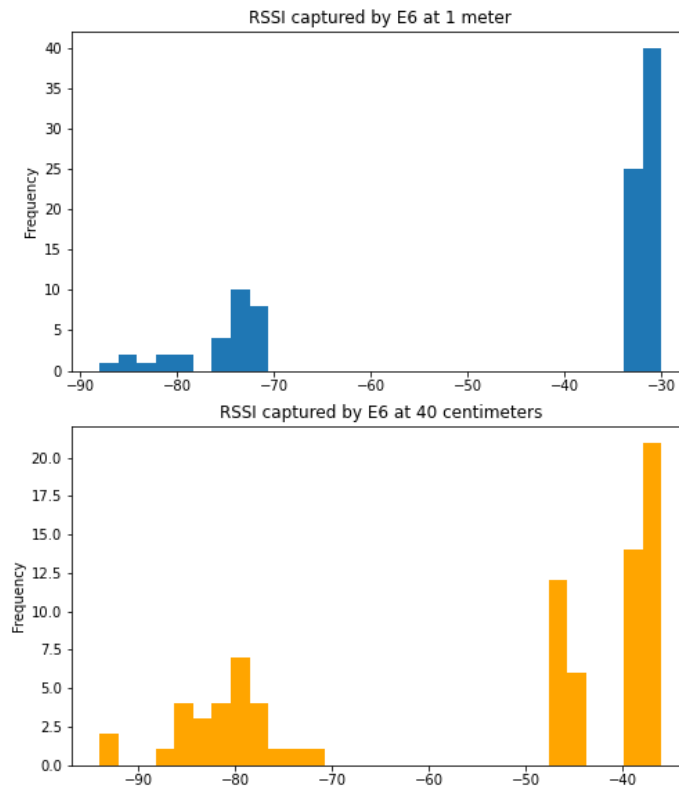


Figura 44: Histograma de RSSI capturado por E6 para corridas a 1m y a 0,4m. Fuente: Elaboración propia

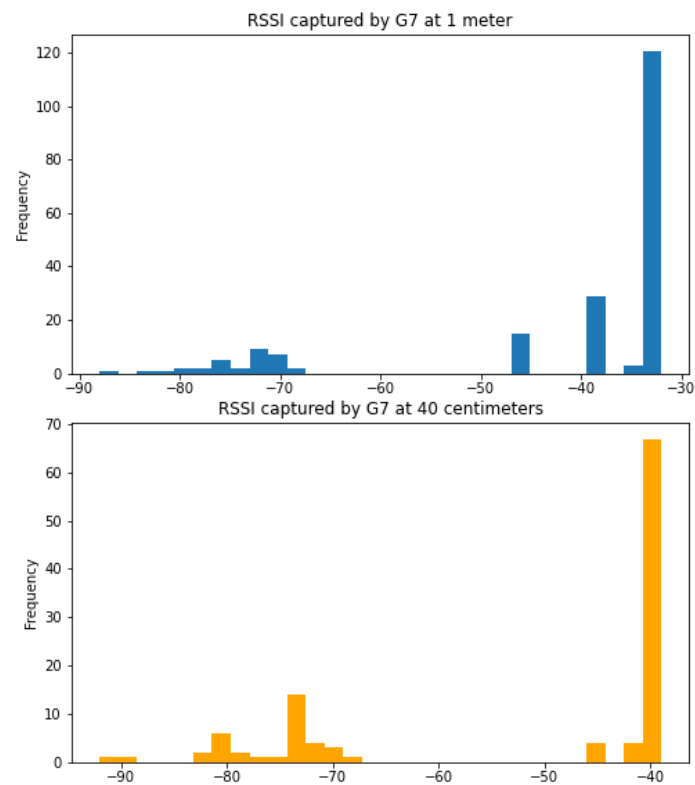


Figura 45: Histograma de RSSI capturado por G7 para corridas a 1m y a 0,4m. Fuente: Elaboración propia

Wi-Fi

Para los datos recolectados sobre Wi-Fi, cada archivo cuenta con las siguientes columnas:

- **record_timestamp**: instante en el que se capturó la presencia de esta red por última vez.
- **run_id**: identificador de la corrida.
- **window_id**: identificador de la ventana.
- **address**: dirección física del punto de acceso a la red.
- **channel_width**: ancho de banda del punto de acceso a la red. Puede ser *20 MHz*, *40 MHz*, *80 MHz*, *160 MHz* o bien *80MHz + 80MHz*.
- **center_frequency_0**: este campo tiene un valor si el ancho de banda es de 40, 80 o 160 MHz, indica la frecuencia central en MHz. Si el ancho de banda es 80MHz + 80MHz, indica la frecuencia central del primer segmento.
- **center_frequency_1**: Si el ancho de banda es 80MHz + 80MHz, indica la frecuencia central del segundo segmento.
- **frequency**: La frecuencia principal de 20 MHz (en MHz) del canal por el que el cliente se comunica con el punto de acceso.
- **level**: nivel de señal en dBm al que se capturó la presencia de la red.
- **passpoint**: indica si el tipo de red es *Passpoint* (**1**) o no (**0**).

Para este tipo de datos se decidió analizar la cantidad de veces que se detectó cada señal de una red Wi-Fi para cada dispositivo y en cada corrida. Esta información se presenta en los gráficos de barras de la Figura 46 para la corrida con 0,4m de separación entre dispositivos y de la Figura 47 para la corrida con 1m de separación.

En este caso se observa cómo claramente la gama del dispositivo influye en la cantidad de señales que es capaz de capturar, siendo G7 el de mayor gama y el que observó más redes Wi-Fi durante ambas corridas del experimento.

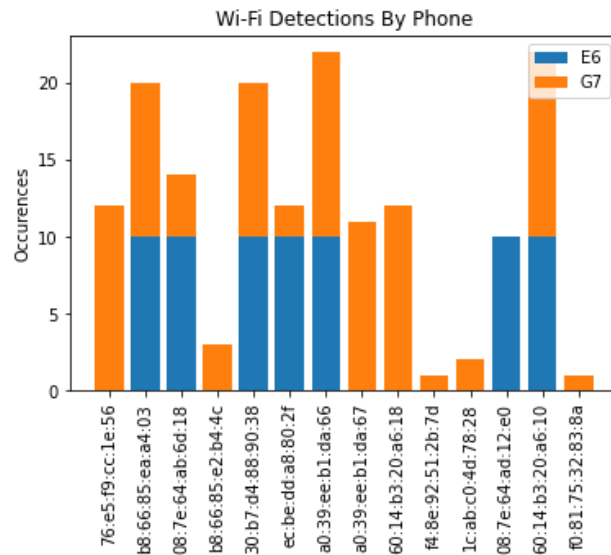


Figura 46: Cantidad de detecciones de cada red para cada dispositivo con 0,4m de separación entre ellos. Fuente: Elaboración propia

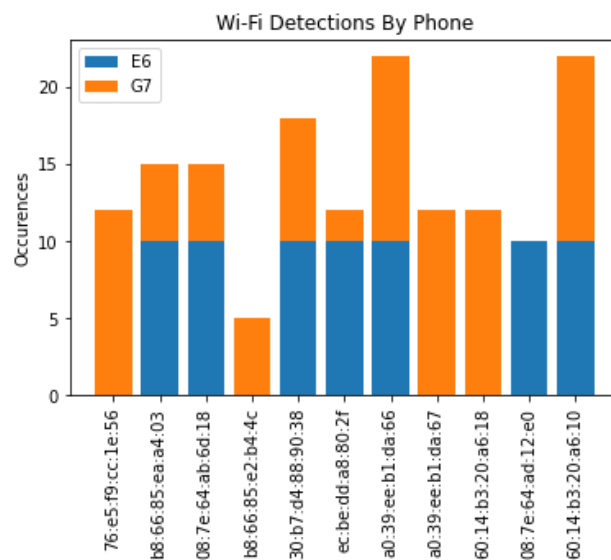


Figura 47: Cantidad de detecciones de cada red para cada dispositivo con 1m de separación entre ellos. Fuente: Elaboración propia

Sensores

Para los datos recolectados sobre Sensores, cada archivo cuenta con las siguientes columnas:

- **timestamp**: instante de tiempo en el que se consultó los valores del sensor
- **run_id**: identificador de la corrida.
- **window_id**: identificador de la ventana.
- **sensor_type**: tipo del sensor.
- **value_id**: como cada sensor puede tener hasta 6 valores, este identificador es para conocer a que posición de ese vector corresponde.
- **value**: valor del sensor medido para la posición determinada por **value_id**.

Finalmente el último conjunto de datos que se procesó fue el de los sensores, para todos los casos se grafican los valores conforme avanzaba el tiempo de ejecución del experimento.

El primer gráfico corresponde al nivel de luminosidad con los dispositivos distanciados a 1m, es el representado por la Figura 48. Si bien ambos dispositivos capturan valores aproximados debido a que se encontraban bajo la misma fuente de luz, E6 estaba ligeramente más alejado de esta y por eso puede observarse tal diferencia en los valores capturados. La caída en la luminosidad capturada por E6 puede deberse a que en ese momento se haya hecho sombra sobre el dispositivo por error o debido a circulación de personas en el ambiente donde se encontraban los dispositivos.

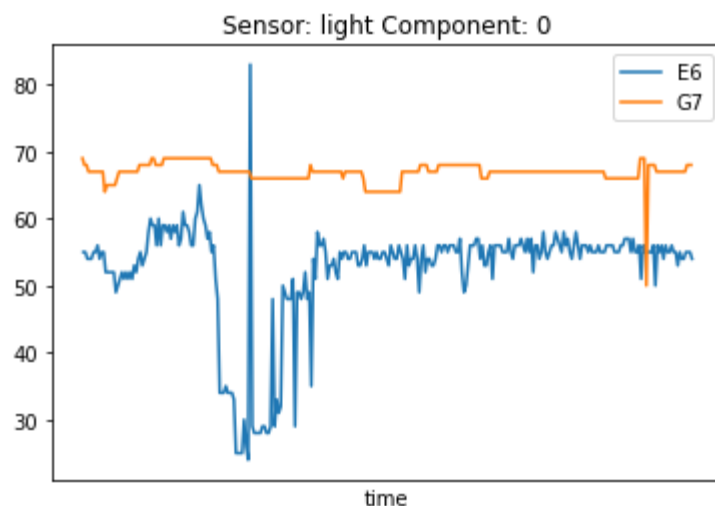


Figura 48: Nivel de luminosidad para cada dispositivo (separación de 1m). Fuente: Elaboración propia

En la Figura 49 se aprecian los niveles de luminosidad capturados durante la corrida que se realizó con los dispositivos estando a 0,4m el uno del otro, por esta razón es que las dos curvas están prácticamente una sobre la otra.

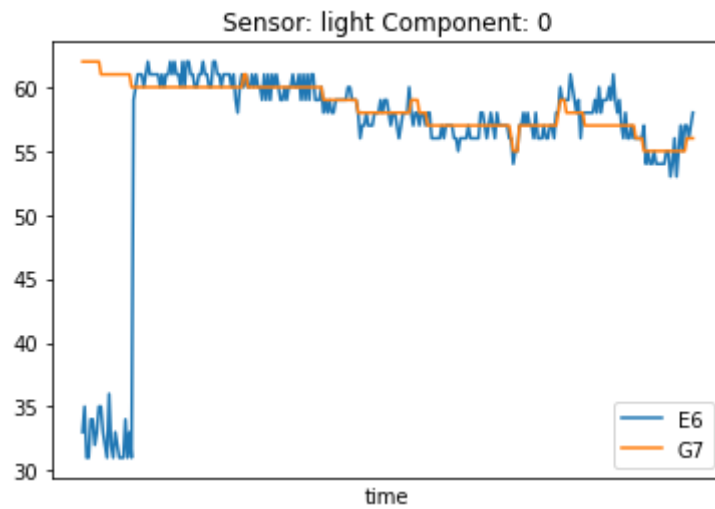


Figura 49: Nivel de luminosidad para cada dispositivo (separación de 0,4m). Fuente: Elaboración propia

Los siguientes gráficos (Figura 50) corresponden a los niveles de aceleración captados por los acelerómetros de los dispositivos, donde las componentes (0, 1, 2) corresponden a los ejes físicos (x, y, z) respectivamente como se describe en el marco teórico.

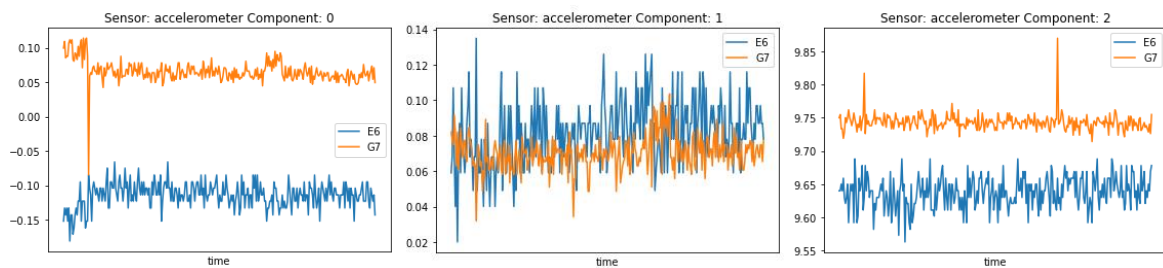


Figura 50: Aceleración en cada eje para cada dispositivo. Fuente: Elaboración propia

En todos los gráficos mostrados se aprecia una clara diferencia y es que los valores capturados por el dispositivo E6 son mucho más ruidosos que los correspondientes a los capturados por G7, esto puede deberse a la calidad de los sensores, ya que ambos dispositivos son de gamas diferentes.

Capítulo 5: Conclusiones

Al inicio de este trabajo se propuso desarrollar una aplicación móvil, para recolectar datos de periféricos de comunicación y sensores, que sería utilizada como una fuente de datos para analizar y detectar los contactos entre personas.

A lo largo del desarrollo se logró que la aplicación monitoree de manera independiente tres diferentes tipos de periféricos de comunicación (Wi-Fi, Bluetooth y Bluetooth Low Energy) en conjunto con un cuarto módulo diseñado para obtener valores de los sensores presentes en el dispositivo, para esto fue necesario diseñar un mecanismo de planificación basado en temporizadores con hilos de ejecución independientes que son gestionados por el sistema operativo y que permiten un control sobre los tiempos de recolección con gran precisión (del orden de los milisegundos), al ser gestionados por el sistema operativo, este controla el acceso a los recursos de hardware. Utilizando el mismo mecanismo, se implementó el módulo de transmisión de datos que facilita la visibilidad del dispositivo como una baliza Bluetooth Low Energy.

Dado que los datos debían ser recuperables en cualquier momento posterior a su recolección, fue de utilidad implementar una base de datos relacional basada en un modelo que represente los elementos que forman parte del dominio del problema. La implementación de interfaces para operar sobre esta, permite la abstracción respecto de cómo se guardan y/o consultan los datos, facilitando así el desarrollo ya que únicamente es necesario enfocarse en la funcionalidad de los diferentes módulos.

Podemos resaltar que la implementación de módulos específicos para la configuración de los experimentos y para compartir estas configuraciones, aporta tanto flexibilidad en la realización de los mismos como así también simplicidad a la hora de replicar un experimento en otros dispositivos.

En el desarrollo del presente trabajo se dedicó tiempo al diseño de las interfaces gráficas de la aplicación, obteniendo como resultado un flujo intuitivo para el usuario a la hora de manipularla para llevar a cabo tareas referidas a la experimentación.

Considerando que la finalidad de la aplicación es funcionar como fuente de datos para el análisis de contactos entre personas, fue de importancia diseñar de manera modular y reutilizable los mecanismos para exportar los datos, ya que permite flexibilidad a la hora de integrar nuevos módulos con diferentes tipos de datos o inclusive con diferentes formatos a los propuestos en este trabajo (CSV y JSON).

También podemos destacar que la implementación de pruebas unitarias y de integración fue de gran utilidad para verificar, en el proceso de desarrollo, que todas las funcionalidades y los requerimientos se cumplan de manera individual y trabajando en conjunto como sistema. Además, las herramientas

utilizadas para automatizar las pruebas y la generación de reportes agilizaron el proceso de desarrollo ya que brindan confiabilidad sobre los módulos ya probados.

Las pruebas de uso realizadas evidencian que la recolección y exportación de datos mediante una única aplicación facilita en gran medida las tareas de experimentación y análisis de contextos en los que se pueden encontrar los dispositivos móviles y por ende las personas.

Trabajos Futuros

Al trabajo presentado en este documento se le podrían realizar mejoras en materia de planificación de las tareas de recolección, particularmente en términos de los recolectores de datos Bluetooth. Como el dispositivo posee un único recurso de hardware de este tipo, sería interesante evaluar cómo afecta al rendimiento de la recolección la superposición de tareas de diferente tipo (escaneo de Bluetooth Clásico, escaneo de Bluetooth Low Energy y transmisión bajo este último) y posiblemente implementar un mecanismo seguro de exclusión sobre este recurso, para así realizar operaciones completas sobre el este sin que el sistema operativo sea quien tenga que gestionar el acceso al hardware.

Otro punto a resaltar es la operación de la aplicación en modo reposo, ya que actualmente la ejecución del servicio de experimentos no admite recolección de datos en este modo, debido a que el sistema operativo no le otorga a la aplicación los recursos para poder acceder a la información. Esto podría deberse a restricciones para mejorar el consumo de la batería y por eso sería importante avanzar en la investigación de este punto.

En lo que respecta a la detección de contactos, sería de interés realizar estudios en cómo los datos recolectados por diferentes experimentos pueden aportar a la precisión de la correcta detección de un encuentro. Para esto se podrían evaluar escenarios donde debe existir una comunicación en la que los dispositivos intercambien información, otros donde los dispositivos no intercambian información pero sí se hagan visibles entre ellos y finalmente un último enfoque en el que cada dispositivo haga una detección totalmente pasiva, es decir únicamente interpretando su entorno.

Anexos

Anexo A: Detalle de Pruebas

En esta sección se describe que verifica cada una de las pruebas implementadas dentro de cada clase.

BluetoothLeAdvertiseSchedulerTest

Tabla 9: Descripción de pruebas para BluetoothLeAdvertiseScheduler. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------------|--|
| scheduledDataCollection | Instanciación y notificación de observadores |

BluetoothCsvFileWriterTest

Tabla 10: Descripción de pruebas para BluetoothCsvFileWriterTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------|---|
| getContent | Que el contenido a escribir no sea nulo |
| getExportableData | Que la lista de exportables no sea nula |
| getFileName | Que el nombre del archivo no sea nulo |
| getKey | Que la clave no sea nula |
| getPath | Que el directorio para el archivo no sea nulo |

BluetoothLeCsvFileWriterTest

Tabla 11: Descripción de pruebas para BluetoothLeCsvFileWriterTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------|---|
| getContent | Que el contenido a escribir no sea nulo |
| getExportableData | Que la lista de exportables no sea nula |
| getFileName | Que el nombre del archivo no sea nulo |
| getKey | Que la clave no sea nula |
| getPath | Que el directorio para el archivo no sea nulo |

SensorCsvFileWriterTest

Tabla 12: Descripción de pruebas para SensorCsvFileWriterTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------|---|
| getContent | Que el contenido a escribir no sea nulo |
| getExportableData | Que la lista de exportables no sea nula |
| getFileName | Que el nombre del archivo no sea nulo |
| getKey | Que la clave no sea nula |
| getPath | Que el directorio para el archivo no sea nulo |

WifiCsvFileWriterTest

Tabla 13: Descripción de pruebas para WifiCsvFileWriterTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------|---|
| getContent | Que el contenido a escribir no sea nulo |
| getExportableData | Que la lista de exportables no sea nula |
| getFileName | Que el nombre del archivo no sea nulo |
| getKey | Que la clave no sea nula |
| getPath | Que el directorio para el archivo no sea nulo |

JsonFileWriterTest

Tabla 14: Descripción de pruebas para JsonFileWriterTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------|---|
| getContent | Que el contenido a escribir no sea nulo |
| getFileName | Que el nombre del archivo no sea nulo |
| getPath | Que el directorio para el archivo no sea nulo |

ExperimentRepresentationTest

Tabla 15: Descripción de pruebas para ExperimentRepresentationTest. Fuente: Elaboración propia

| Prueba | Verifica |
|----------------|---|
| objectHashCode | Que el código hash no sea nulo |
| testEquals | El método de igualdad de objetos |
| toJson | Que la cadena que representa en formato JSON al objeto no sea nula ni vacía |

BluetoothLeRepositoryTest

Tabla 16: Descripción de pruebas para BluetoothLeRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------|--|
| getAllSamples | Que la cantidad de elementos insertados sea igual a la cantidad de elementos consultados para todas las muestras insertadas para una corrida |
| getLiveCount | La notificación de los cambios en la cuenta de cantidad de muestras insertadas para una corrida |
| insertBluetoothLe | La inserción de muestras |
| insertUuids | La inserción de identificadores UUIDs |

BluetoothRepositoryTest

Tabla 17: Descripción de pruebas para BluetoothRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-----------------|--|
| getAllSamples | Que la cantidad de elementos insertados sea igual a la cantidad de elementos consultados para todas las muestras insertadas para una corrida |
| getLiveCount | La notificación de los cambios en la cuenta de cantidad de muestras insertadas para una corrida |
| insertBluetooth | La inserción de muestras |

DeviceRepositoryTest

Tabla 18: Descripción de pruebas para DeviceRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|---------------|--|
| getDevice | Que el dispositivo insertado sea el mismo que el consultado |
| getDeviceNull | Que la consulta sin haber insertado un dispositivo, devuelva un valor nulo |
| insert | La inserción de un dispositivo |
| update | La actualización de los campos de un dispositivo |

ConfigurationRepositoryTest

Tabla 19: Descripción de pruebas para ConfigurationRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|---|---|
| delete | La eliminación de una configuración |
| emptyConfigs | Que si no se realiza ninguna inserción, al consultar se devuelva una lista vacía |
| getConfigurationForExperimentByType | Que si se realiza una inserción de una configuración la misma pueda ser obtenida por su tipo |
| getConfigurationForExperimentByTypeThatNotExist | Que si se consulta por una configuración de determinado tipo que no existe para un determinado experimento, la misma sea nula |
| getConfigurationsForExperiment | Que la cantidad de configuraciones insertadas para un experimento sea la misma que la cantidad obtenidas para la consulta del mismo |
| insert | La inserción de configuraciones de ventanas |
| insertAdvertise | La inserción de configuraciones de transmisión |
| nonExistentAdvertise | Que si no se realiza ninguna inserción de configuraciones de transmisión, al consultar se devuelva un valor nulo |

SensorRepositoryTest

Tabla 20: Descripción de pruebas para SensorRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|------------------|--|
| getAllSamplesFor | Que la cantidad de elementos insertados sea igual a la cantidad de elementos consultados para todas las muestras insertadas para una corrida |
| getLiveCount | La notificación de los cambios en la cuenta de cantidad de muestras insertadas para una corrida |
| insert | La inserción de muestras |

SourceTypeRepositoryTest

Tabla 21: Descripción de pruebas para SourceTypeRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-----------|---|
| getIdBy | Que la obtención de un tipo de fuente no sea nula cuando se consulta por un identificador existente |
| getTypeBy | Que la obtención de un tipo de fuente no sea nula cuando se consulta por un tipo válido |
| insert | La inserción de tipos de fuente |

WifiRepositoryTest

Tabla 22: Descripción de pruebas para WifiRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|------------------|--|
| getAllSamplesFor | Que la cantidad de elementos insertados sea igual a la cantidad de elementos consultados para todas las muestras insertadas para una corrida |
| getLiveCount | La notificación de los cambios en la cuenta de cantidad de muestras insertadas para una corrida |
| insertWifi | La inserción de muestras |

WindowRepositoryTest

Tabla 23: Descripción de pruebas para WindowRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|--------|-----------------------------|
| insert | La inserción de una ventana |

ExperimentRepositoryTest

Tabla 24: Descripción de pruebas para ExperimentRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-----------------------------|---|
| delete | La eliminación de un experimento |
| getAllExperimentDoneEmpty | La notificación de cambios en la lista de experimentos |
| getAllExperimentDoneNotNull | Que la lista de experimentos no sea nula |
| getAllTagList | Que la cantidad de elementos en la lista de etiquetas sea igual a la cantidad de etiquetas insertadas |
| getByIdNotNull | Que la consulta de un experimento existente no sea nula |
| getByIdNull | Que la consulta de un experimento inexistente sea nula |
| getLastEquals | Que el último experimento insertado sea igual al obtenido de consultar por el último insertado |
| getLastNotNull | Que la consulta del último experimento insertado no sea nula cuando se haya realizado una inserción previamente |
| getLastNull | Que la consulta del último experimento insertado sea nula si no se realizaron inserciones previamente |
| getRelatedTagList | Que la cantidad de etiquetas para un experimento sea la misma que la cantidad de etiquetas a las que se asoció el mismo |
| getTagIdNotNull | Que la consulta de una etiqueta existente no sea nula |
| getTagIdNull | Que la consulta de una etiqueta inexistente sea nula |
| insert | La inserción de experimentos |
| insertTag | La inserción de etiquetas |

ShareToolsTest

Tabla 25: Descripción de pruebas para ShareToolsTest. Fuente: Elaboración propia

| Prueba | Verifica |
|--------------------|--|
| getCodedExperiment | Que el la representación de un experimento para una cadena de caracteres no sea nula |
| getQrExperiment | Que el código QR para una representación de un experimento válido no sea nulo |

RunRepositoryTest

Tabla 26: Descripción de pruebas para RunRepositoryTest. Fuente: Elaboración propia

| Prueba | Verifica |
|--------------------------------|---|
| delete | La eliminación de una corrida |
| getIdNotNull | Que la consulta de una corrida existente no sea nula |
| getIdNull | Que la consulta de una corrida inexistente sea nula |
| getLastRunForExperimentNonZero | Que el número de corridas no sea cero para un experimento que ya tiene corridas asociadas |
| getLastRunForExperimentZero | Que el número de corridas no cero para un experimento que no tiene corridas asociadas |
| getLiveRun | La notificación de cambios en una corrida |
| getMaxDuration | Que la consulta de la máxima duración del experimento sea mayor o igual a la de cualquier configuración que tenga asociada el mismo |
| getRunForExperiment | La notificación de cambios en la lista de corridas para un determinado experimento |
| insert | La inserción de corridas |
| scheduledOrRunningEmpty | Que la lista de corridas este vacia si no se ha programado ninguna |
| scheduledOrRunningNotEmpty | Que la lista de corridas no este vacia si se han programado corridas previamente |
| updateState | La actualización del campo de estado de una corrida |

BluetoothScanSchedulerTest

Tabla 27: Descripción de pruebas para BluetoothScanSchedulerTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------------|--|
| scheduledDataCollection | Instanciación y notificación de observadores |

SensorsScanSchedulerTest

Tabla 28: Descripción de pruebas para SensorsScanSchedulerTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------------|--|
| scheduledDataCollection | Instanciación y notificación de observadores |

WifiScanSchedulerTest

Tabla 29: Descripción de pruebas para WifiScanSchedulerTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------------|--|
| scheduledDataCollection | Instanciación y notificación de observadores |

BluetoothLeScanSchedulerTest

Tabla 30: Descripción de pruebas para BluetoothLeScanSchedulerTest. Fuente: Elaboración propia

| Prueba | Verifica |
|-------------------------|--|
| scheduledDataCollection | Instanciación y notificación de observadores |

Anexo B: Manual para el usuario

Inicio de aplicación y configuración del dispositivo

Para iniciar la aplicación diríjase al menú de aplicaciones y busque el ícono que tenga el nombre *DigiTriad Lab* como indica la Figura 51, toque el ícono y espere a visualizar la pantalla principal.

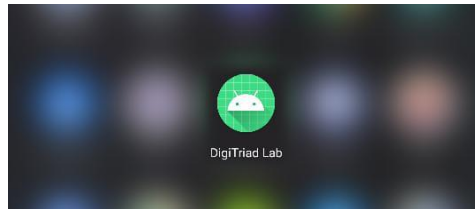







Figura 51: Ícono de la aplicación en el menú. Fuente: Elaboración propia

Como se aprecia en la Figura 52, esta pantalla muestra una lista de los experimentos que tiene configurados el dispositivo, junto a un ícono de estado para cada uno de estos:

-  El experimento tiene al menos una corrida pendiente.
-  El experimento tiene una corrida ejecutándose.
-  El experimento no tiene ninguna corrida programada.
-  Todas las corridas programadas para un experimento han finalizado.

Para visualizar o modificar datos del dispositivo toque el botón con el icono  en la esquina superior derecha del dispositivo.

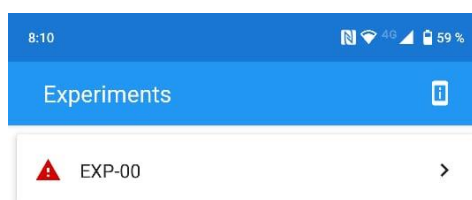


Figura 52: Recorte superior de pantalla principal Fuente: Elaboración propia

Al hacer esto se abrirá una nueva pantalla (Figura 53) donde puede indicar el nombre en código del dispositivo y donde también dispone de dos botones, uno para generar un nuevo UUID y otro para guardar los cambios .

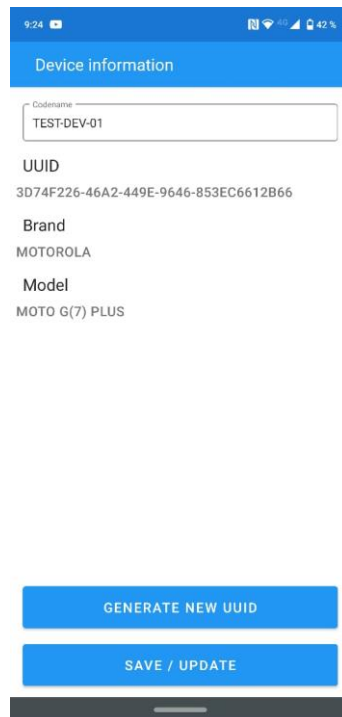


Figura 53: Pantalla de detalle del dispositivo Fuente: Elaboración propia

Configuración de nuevos experimentos

Para configurar nuevos dispositivos debe tocar el botón con el ícono **+** en la parte inferior derecha de la pantalla, como indica la Figura 54.



Figura 54: Recorte inferior de pantalla principal Fuente: Elaboración propia

Al hacer esto la aplicación lo llevará a la pantalla de la Figura 55 donde podrá completar cada uno de los campos de acuerdo a las necesidades del experimento. Antes de abandonar la pantalla debe tocar el botón de guardar, de lo contrario los datos cargados no tendrán efecto.

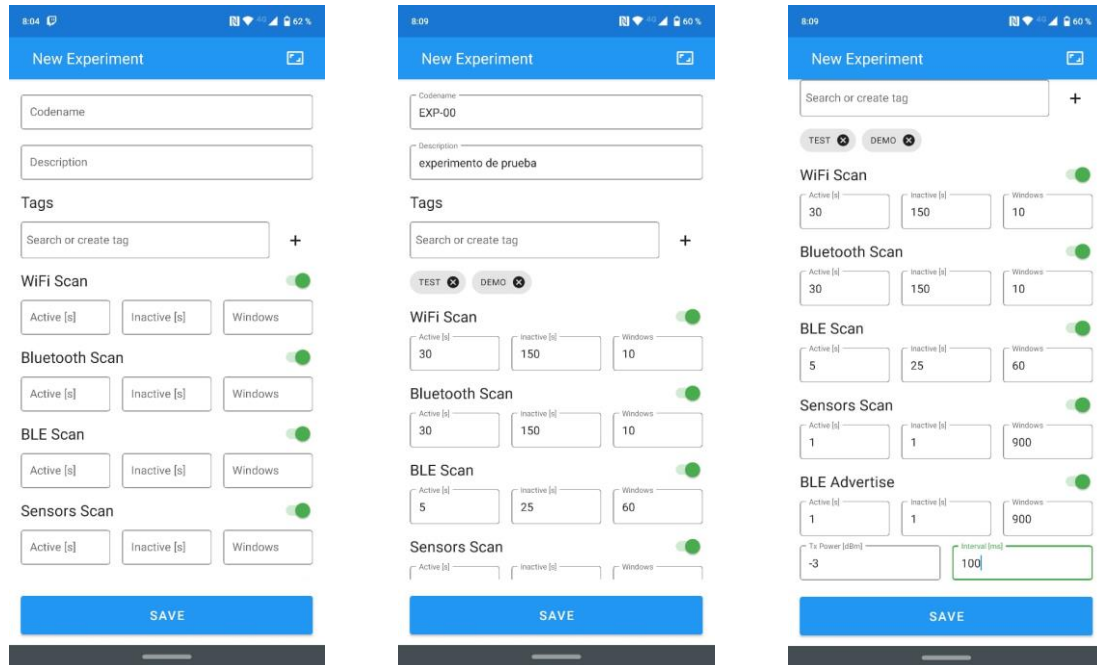



Figura 55: Secuencia de imágenes para la creación de un experimento Fuente: Elaboración propia

La aplicación también ofrece la posibilidad de cargar los campos de un experimento mediante el escaneo de un código QR, para esto debe tocar el botón con el ícono , al hacerlo se abrirá la cámara como en la Figura 56.a.

Una vez escaneado el código la aplicación lo devolverá a la pantalla de configuración del experimento con los campos precargados tal como señala la Figura 56.b.

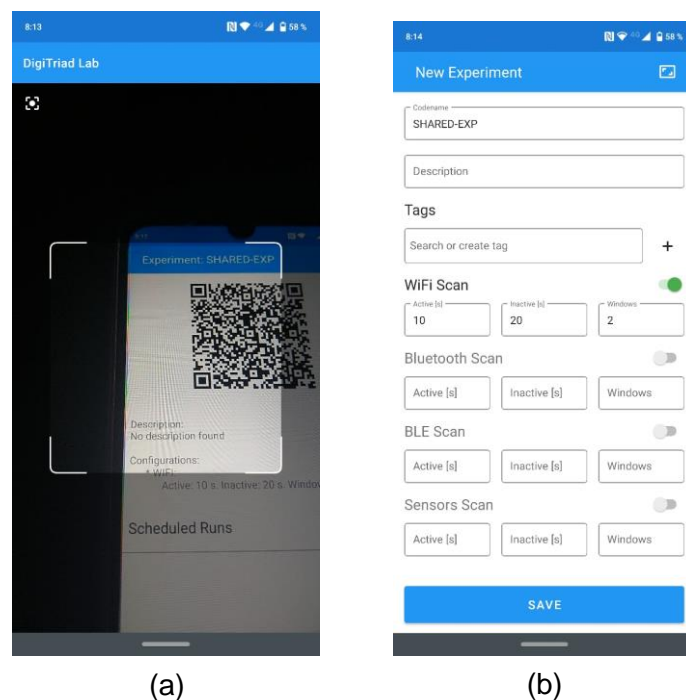


Figura 56: Secuencia de imágenes para el escaneo de un experimento Fuente: Elaboración propia

Como se aprecia, también se pueden desactivar algunas configuraciones *apagando* los interruptores visibles a la derecha de la pantalla. Esta acción es necesaria **si no se van a cargar esos campos**, de lo contrario la aplicación **no le dejará guardar el experimento**.

Detalle de experimento y programación de nuevas corridas

Para acceder al detalle de un experimento determinado basta con tocar el elemento de la lista visible en la pantalla principal. Esta acción guiará al usuario a una pantalla como la de la Figura 57.a, donde podrá observar el nombre del experimento, su código QR, sus etiquetas asociadas, su descripción, sus configuraciones y también sus corridas programadas.

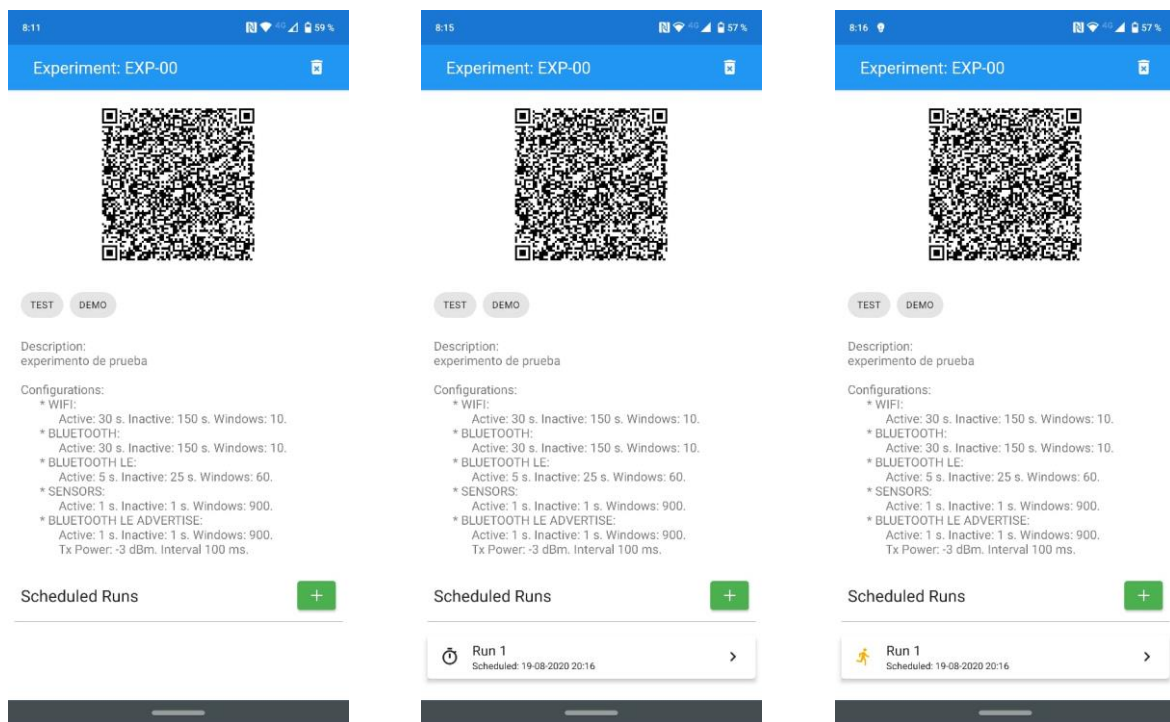








Figura 57: Secuencia de imágenes para el detalle de un experimento Fuente: Elaboración propia

Nótese que la lista de corridas muestra cada una con la fecha y hora para la que se programó junto a un ícono de estado de ejecución de la misma:

-  La corrida está pendiente de ejecución.
-  La corrida se está ejecutando.
-  La corrida fue cancelada.
-  La corrida ha finalizado.

Desde esta pantalla se puede eliminar el experimento tocando el botón con el ícono , al hacerlo la aplicación solicitará confirmación ya que esta acción es irreversible.

También desde esta pantalla se pueden programar nuevas corridas tocando el botón con el ícono , acción que lo dirigirá a una pantalla como la de la Figura 58 donde puede especificar la fecha y hora de inicio de la corrida.




Figura 58: Pantalla para la programación de una nueva corrida. Fuente: Elaboración propia

Al intentar guardar la corrida, la aplicación validará que no se produzca ninguna superposición de ejecuciones ya que solo se puede ejecutar una corrida a la vez.

Detalle de corridas

Para acceder al detalle de una corrida, solo es necesario tocar el elemento deseado en la lista de corridas visible en el detalle de un experimento, llevándolo a una pantalla como la de la Figura 59 donde podrá visualizar la cantidad de muestras recolectadas de cada tipo, como así también disparar manualmente la exportación de datos con el botón disponible para tal fin.

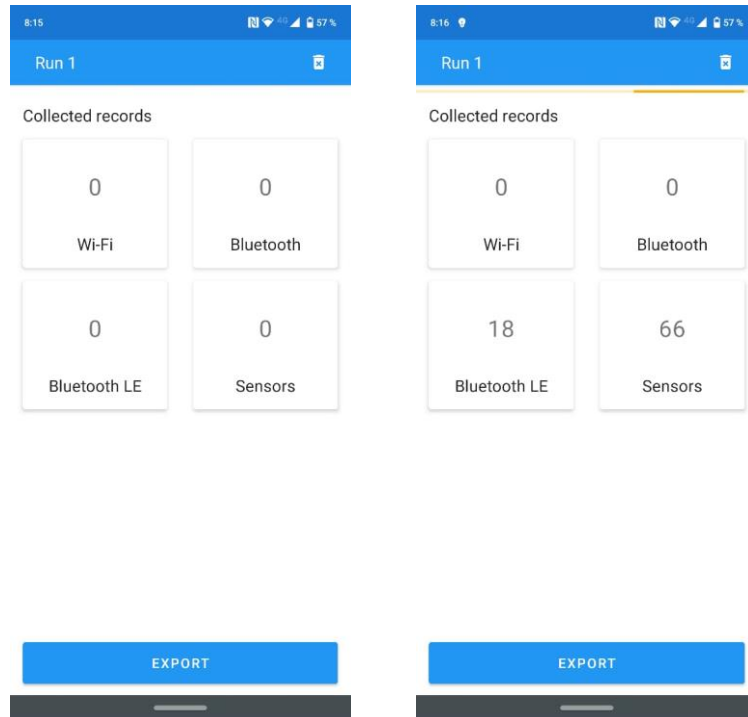



Figura 59: Pantalla de detalle de una corrida Fuente: Elaboración propia

Nótese que cuando una corrida está en ejecución se visualiza una barra de progreso indefinida en la parte superior.

Desde esta pantalla también es posible eliminar una corrida tocando el botón con el ícono  visible en la parte superior derecha. Esta acción sólo tendrá efecto cuando la corrida haya finalizado.

Exportación de datos

Cada vez que se ejecuta un experimento los datos recolectados se escriben en una base de datos local en la memoria del teléfono. Al finalizar la ejecución, automáticamente se exportan todas las muestras recolectadas durante esa corrida a la memoria del dispositivo.

Los archivos generados por la aplicación se encuentran en el directorio:

```

→ raíz del dispositivo
  ↳ Android
    ↳ data
      ↳ com.rfdetke.digitriadlaboratory
        ↳ files
          ↳ exports
  
```

Dentro de la carpeta **exports** se genera una nueva carpeta por cada experimento, y estas contienen los archivos en formato CSV para las muestras y JSON para la descripción del experimento.

Anexo C: Algoritmos para graficar datos de prueba

En esta sección se presentan los tres algoritmos utilizados para procesar y generar los gráficos de la prueba de ejecución de un experimento.

Bluetooth Low Energy

```
import pandas as pd
import json

fig_size = (8, 4.5)

with open('datos\E6\X-exp-demo_E6.json') as f:
    e6_X = json.load(f)[0]
e6_10 = pd.read_csv('datos\E6\X-exp-demo_R-1_bluetooth_le_E6.csv').sort_values(by=['record_timestamp', 'run_id', 'window_id'])
e6_04 = pd.read_csv('datos\E6\X-exp-demo_R-2_bluetooth_le_E6.csv').sort_values(by=['record_timestamp', 'run_id', 'window_id'])

e6_10 = e6_10[e6_10['uuid'].notna()]
e6_10['distance'] = 1.0
e6_04 = e6_04[e6_04['uuid'].notna()]
e6_04['distance'] = 0.4

e6_distance = pd.concat([e6_10, e6_04], axis=0)
fig = e6_distance.plot.scatter(x = 'rssi', y = 'distance',
figsize=fig_size).get_figure()
fig.savefig('./images/e6_scatter_ble.png')

e6_hist = pd.DataFrame()
e6_hist['rssi_04'] = e6_04.rssi
fig = e6_hist.plot.hist(bins=30, grid=False, figsize=fig_size, legend=False,
title='RSSI captured by E6 at 1 meter').get_figure()
fig.savefig('./images/e6_04_hist_ble.png')

e6_hist = pd.DataFrame()
e6_hist['rssi_10'] = e6_10.rssi
fig = e6_hist.plot.hist(bins=30, grid=False, figsize=fig_size, color=['orange'],
legend=False, title='RSSI captured by E6 at 40 centimeters').get_figure()
fig.savefig('./images/e6_10_hist_ble.png')

with open('datos\G7\X-exp-demo_G7.json') as f:
    g7_X = json.load(f)[0]
g7_10 = pd.read_csv('datos\G7\X-exp-demo_R-1_bluetooth_le_G7.csv').sort_values(by=['record_timestamp', 'run_id', 'window_id'])
g7_04 = pd.read_csv('datos\G7\X-exp-demo_R-2_bluetooth_le_G7.csv').sort_values(by=['record_timestamp', 'run_id', 'window_id'])

g7_10 = g7_10[g7_10['uuid'].notna()]
g7_10['distance'] = 1.0
g7_04 = g7_04[g7_04['uuid'].notna()]
g7_04['distance'] = 0.4
```

```
g7_distance = pd.concat([g7_10, g7_04], axis=0)
fig = g7_distance.plot.scatter(x = 'rssi', y = 'distance',
                               figsize=fig_size).get_figure()
fig.savefig('./images/g7_scatter_ble.png')

g7_hist = pd.DataFrame()
g7_hist['rssi_04'] = g7_04.rssi
fig = g7_hist.plot.hist(bins=30, grid=False, figsize=fig_size, legend=False,
                        title='RSSI captured by G7 at 1 meter').get_figure()
fig.savefig('./images/g7_04_hist_ble.png')

g7_hist = pd.DataFrame()
g7_hist['rssi_10'] = g7_10.rssi
fig = g7_hist.plot.hist(bins=30, grid=False, figsize=fig_size, color=['orange'],
                        legend=False, title='RSSI captured by G7 at 40 centimeters').get_figure()
fig.savefig('./images/g7_10_hist_ble.png')

g7_04_samples = g7_04.copy()
fig = g7_04_samples.plot.line(x='record_timestamp', y='rssi', figsize=fig_size,
                              title='RSSI captured by G7 at 40 centimeters').get_figure()
fig.savefig('./images/g7_04_line_ble.png')

g7_10_samples = g7_10.copy()
fig = g7_10_samples.plot.line(x='record_timestamp', y='rssi', figsize=fig_size,
                              color=['orange'], title='RSSI captured by G7 at 1 meter').get_figure()
fig.savefig('./images/g7_10_line_ble.png')

e6_04_samples = e6_04.copy()
fig = e6_04_samples.plot.line(x='record_timestamp', y='rssi', figsize=fig_size,
                              title='RSSI captured by E6 at 40 centimeters').get_figure()
fig.savefig('./images/e6_04_line_ble.png')

e6_10_samples = e6_10.copy()
fig = e6_10_samples.plot.line(x='record_timestamp', y='rssi', figsize=fig_size,
                              color=['orange'], title='RSSI captured by E6 at 1 meter').get_figure()
fig.savefig('./images/e6_10_line_ble.png')
```

Wi-Fi

```
import matplotlib.pyplot as plt
import pandas as pd

e6 = pd.read_csv('datos\E6\X-exp-demo_R-
1_wifi_E6.csv').sort_values(by=['record_timestamp', 'run_id', 'window_id'])
g7 = pd.read_csv('datos\G7\X-exp-demo_R-
1_wifi_G7.csv').sort_values(by=['record_timestamp', 'run_id', 'window_id'])

unique_addresses = list(set(e6.address.unique()) |
                           set(g7.address.unique()))
unique_addresses_df = pd.DataFrame(index=unique_addresses, columns=['address'])
unique_addresses_df = unique_addresses_df.fillna(0)

e6_address_count = pd.DataFrame(e6.address.value_counts())
e6_address_count = e6_address_count.join(unique_addresses_df, how='right',
rsuffix='_r').fillna(0).drop(['address_r'], axis=1)

g7_address_count = pd.DataFrame(g7.address.value_counts())
g7_address_count = g7_address_count.join(unique_addresses_df, how='right',
rsuffix='_r').fillna(0).drop(['address_r'], axis=1)

width = 0.35
plt.bar(unique_addresses, e6_address_count.address)
plt.bar(unique_addresses, g7_address_count.address,
bottom=e6_address_count.address)
plt.ylabel('Occurrences')
plt.title('Wi-Fi Detections By Phone')
plt.legend(labels=['E6', 'G7'])
plt.xticks(unique_addresses, rotation='vertical')
plt.savefig('./images/bar_10_wifi_occurences.png', bbox_inches = 'tight')
```

Sensores

```
import matplotlib.pyplot as plt
import pandas as pd

e6 = pd.read_csv('datos\E6\\X-exp-demo_R-
2_sensors_E6.csv').sort_values(by=['timestamp', 'run_id', 'window_id'])
g7 = pd.read_csv('datos\G7\\X-exp-demo_R-
2_sensors_G7.csv').sort_values(by=['timestamp', 'run_id', 'window_id'])

common_sensor_types = list(set(e6.sensor_type.unique()) |
                             set(g7.sensor_type.unique()))

for sensor_type in common_sensor_types:
    e6_sensor = e6[e6['sensor_type']==sensor_type].dropna()
    g7_sensor = g7[g7['sensor_type']==sensor_type].dropna()

    for component in g7_sensor.value_id.unique():
        e6_filtered = e6_sensor[e6_sensor['value_id']==component]
        g7_filtered = g7_sensor[g7_sensor['value_id']==component]

        ax = plt.axes()
        ax.plot(e6_filtered.timestamp, e6_filtered.value, label='E6')
        ax.plot(g7_filtered.timestamp, g7_filtered.value, label='G7')

        plt.title('Sensor: %s Component: %d' % (sensor_type.lower(), component))
        plt.xlabel('time')
        plt.xticks([])
        plt.legend()

        plt.savefig('./images/04_sensors/%s-%d.png'%(sensor_type.lower(),
component), bbox_inches = 'tight')
        plt.close()
```

Referencias

- [1] “Cisco Annual Internet Report (2018–2023) White Paper”, Marzo 2020. [Online] Available: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>
- [2] “The history of Wi-Fi”, Mayo 2015. [Online] Available <https://purple.ai/blogs/history-wifi/>
- [3] “Wi-Fi Alliance Website” [Online] Available: <https://www.wi-fi.org/>
- [4] “IEEE 802.11 Working Group Website” [Online] Available: <https://www.ieee802.org/11/>
- [5] “Power Saving Features Highlights” [Online] Available: <https://www.wi-fi.org/file/power-saving-features-highlights>
- [6] Farrahi K, Emonet R, Cebrian M. “Epidemic Contact Tracing via Communication Traces” PLoS ONE 9(5) [Online]. Available: <https://doi.org/10.1371/journal.pone.0095133>
- [7] “The Bluetooth™ Wireless Technology White Paper”. Atmel. [Online] Available: <http://educypedia.karadimov.info/library/DOC1991.PDF>
- [8] “Radio Versions | Bluetooth® Technology Website,” *Bluetooth® Technology Website*. [Online]. Available: bluetooth.com/learn-about-bluetooth/bluetooth-technology/radio-versions
- [9] “Bluetooth Low Energy Overview” [Online] Available: <https://developer.android.com/guide/topics/connectivity/bluetooth-le>
- [10] M. S. Gast “802.11 Wireless Networks: The Definitive Guide” O'Reilly Media, 2005. Chapter 4 [Online] Available: <https://www.oreilly.com/library/view/80211-wireless-networks/0596100523/ch04.html>
- [11] A. Basalamah. “BLESS: Opportunistic Crowdsensing Framework for Internet of Things using Bluetooth Low Energy” The Journal of Engineering, 2016. [Online] Available: https://www.researchgate.net/publication/301569324_BLESS_Opportunistic_Crowdsensing_Framework_for_Internet_of_Things_using_Bluetooth_Low_Energy
- [12] “Sensors Overview - Android Developers” [Online] Available: https://developer.android.com/guide/topics/sensors/sensors_overview
- [13] “Application Fundamentals - Android Developers” [Online] Available: <https://developer.android.com/guide/components/fundamentals>
- [14] “Permissions Overview - Android Developers” [Online] Available: <https://developer.android.com/guide/topics/permissions/overview>
- [15] “Introduction to Activities - Android Developers” [Online] Available: <https://developer.android.com/guide/components/activities/intro-activities>
- [16] “Services Overview - Android Developers” [Online] Available: <https://developer.android.com/guide/components/services>

- [17] “Notifications Overview - Android Developers” [Online] Available:
<https://developer.android.com/guide/topics/ui/notifiers/notifications>
- [18] “Intent and Intent Filters - Android Developers” [Online] Available:
<https://developer.android.com/guide/components/intents-filters>
- [19] “Broadcasts Overview - Android Developers” [Online] Available:
<https://developer.android.com/guide/components/broadcasts>
- [20] “BroadcastReceiver - Android Developers” [Online] Available:
<https://developer.android.com/reference/android/content/BroadcastReceiver>
- [21] M. Richards, N. Ford “Fundamentals of Software Architecture: An engineering approach”
O'Reilly Media, Febrero 2020. pp-16-20.
- [22] “Android MVVM Architectural Pattern” JournalDev. [Online] Available:
<https://www.journaldev.com/20292/android-mvvm-design-pattern>
- [23] E. Gamma, R. Helm, R. Johnson, J. Vlissides “Design Patterns Elements of Reusable Object-
Oriented Software” Addison-Wesley Longman Publishing Co., 1995.
- [24] O. Taylor “J2EE Data Access Objects: A brief Article for Developers”. The Middleware
Company.
- [25] “Guide to app architecture” [Online]. Available:
<https://developer.android.com/jetpack/guide>
- [26] K. Kosobudzki, “Repository Design Pattern,” Medium, Junio 2018. [Online]. Available:
medium.com/@krzychukosobudzki/repository-design-pattern-bc490b256006
- [27] “Save data in a local database using Room” [Online]. Available:
developer.android.com/training/data-storage/room
- [28] O. Latcu, “Android Repository Pattern using RX & Room,” [Online]. Available:
medium.com/corebuild-software/android-repository-pattern-using-rx-room-bac6c65d7385
- [29] “QR Generator - Github Repository” [Online]. Available:
<https://github.com/androidmads/QRGenerator>
- [30] “Code Scanner - Github Repository” [Online]. Available: <https://github.com/yuriy-budiyev/code-scanner>
- [31] “Android Studio” [Online] Available:
<https://developer.android.com/studio>
- [32] “Material Design - Android” [Online] Available:
<https://material.io/develop/android>
- [33] “Gitlab VCS” [Online] Available:
<https://about.gitlab.com/>
- [34] “Visual Paradigm Online Suite” [Online] Available:
<https://online.visual-paradigm.com/es/>

- [35] “Wi-Fi scanning overview - Android Developers” [Online] Available:
<https://developer.android.com/guide/topics/connectivity/wifi-scan>
- [36] “Bluetooth overview - Android Developers” [Online] Available:
<https://developer.android.com/guide/topics/connectivity/bluetooth>
- [37] “WifiManager API Reference - Android Developers” [Online]
Available: <https://developer.android.com/reference/android/net/wifi/WifiManager>
- [38] “BluetoothDevice API Reference - Android Developers” [Online] Available:
<https://developer.android.com/reference/android/bluetooth/BluetoothDevice>
- [39] “BluetoothAdapter.ScanCallback - Android Developers” [Online] Available:
<https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.LeScanCallback>
- [40] “Scheduling repeating alarms” [Online]. Available:
<https://developer.android.com/training/scheduling/alarms>
- [41] “Java - Oracle” [Online] Available:
<https://www.java.com/es/>
- [42] “GSON Library” [Online] Available:
<https://github.com/google/gson>
- [43] “Common Format and MIME Type for Comma-Separated Values (CSV) Files” [Online].
Available: tools.ietf.org/html/rfc4180
- [44] “Bluetooth Low Energy Advertising on Android” [Online]. Available:
source.android.com/devices/bluetooth/ble_advertising
- [45] “The Ultimate Guide to Android Bluetooth Low Energy” [Online]. Available:
<https://punchthrough.com/android-ble-guide/>
- [46] “JavaScript Object Notation Format” [Online] Available:
<https://www.json.org/json-en.html>
- [47] “JUnit testing framework” [Online] Available:
<https://junit.org/junit4/>
- [48] “JaCoCo Java Code Coverage Tool” [Online] Available:
<https://www.eclemma.org/jacoco/>
- [49] Jason Bay, Joel Kek, Alvin Tan, Chai Sheng Hau, Lai Yongquan, Janice Tan, Tang Anh Quy
“BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders”.
[Online] Available:
https://bluetrace.io/static/bluetrace_whitepaper-938063656596c104632def383eb33b3c.pdf
- [50] “Decentralized Privacy-Preserving Proximity Tracing”. DP-3T Team. [Online] Available:
<https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf>