Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación para la integración de un dispositivo de bioimpedancia en un entorno sanitario estandarizado

Autor: Gonzalo Caballero Lora
Tutor: Jorge Calvillo Arbizu

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación para la integración de un dispositivo de bioimpedancia en un entorno sanitario estandarizado

Autor:
Gonzalo Caballero Lora

Tutor:
Jorge Calvilo Arbizu
Profesor Permanente Laboral

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Aplicación para la integración de un dispositivo de bioimpedancia en un entorno sanitario estandarizado

Autor: Gonzalo Caballero Lora

Tutor: Jorge Calvillo Arbizu

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Escribo estas líneas con la emoción de haber llegado al final de una etapa significativa en mi vida. Este Trabajo Fin de Grado no solo marca el cierre de mi etapa universitaria, sino también el fin de un camino que comenzó mucho antes, lleno de momentos inolvidables y personas que han sido fundamentales en mi recorrido.

En primer lugar, quiero expresar mi más profundo agradecimiento a mi familia, quienes siempre han confiado en mí, y hacer mención especial a mis padres, quienes me han estado apoyando en todo momento, brindandóme su ayuda y atención.

A mis amigos de siempre, que aunque cada uno de nosotros escogiéramos caminos distintos, siempre hemos estado ahí los unos para los otros para cuando necesitábamos despejar nuestra mente y pasar un buen rato entre amigos. Gracias por estar tanto en los buenos y malos momentos.

A la familia que encontré en Sevilla: Carlos, Moreno, Marta, Álvaro, Alejandro, Lucía, Antonio y Rafa. A mi otra familia que encontré en mi querida R.U. Pinomontano. Gracias a ti Juanjo, el mejor compañero de habitación que se puede tener, por tu forma de ser tan peculiar que ha alegrado muchos de mis días universitarios. A mi gran descubrimiento, Teresa, gracias por haber formado parte de mi vida en Sevilla y continuarla en nuestro pueblo. Y gracias a todos aquellos que no he mencionado pero que tienen un espacio reservado en mi corazón.

Y como no, gracias a ti Sara, por ser la persona que se ha convertido en mi pilar todos estos años, y la que más apoyo y confianza a mostrado hacia mí. Por esos paseos por Sevilla, que nos servían para alejarnos un poco de la universidad. Por haber estado en los buenos momentos, pero sobretodo en los malos, y a pesar de ello simpre has estado ahí. Gracias de corazón.

Por último, pero no menos importante, quiero agradecer a todos los profesores que me han guiado y enseñado a lo largo de estos años, desde mis maestros de infantil hasta los profesores de universidad. En especial, agradecer a Jorge Calvillo, mi tutor en este proyecto, por su paciencia, atención, apoyo constante y confianza en mí.

Gonzalo Caballero Lora

Sevilla, 2024

Resumen

Actualmente, en el sector sanitario, la interoperabilidad y el intercambio de información entre sistemas es un desafío significativo. La información de un paciente suele estar distribuida en múltiples sistemas que no siempre se comunican entre sí de manera eficiente, lo que dificulta la integración y uso adecuado de los datos. Dentro de las fuentes de información de pacientes, los dispositivos de monitorización y pruebas tienen el potencial de recoger datos de salud de los pacientes de manera continua y en formatos heterogéneos. Esta información, para su correcta utilización en la atención de los pacientes, debería estar integrada en los sistemas de historia clínica. Entre las técnicas de diagnóstico y dispositivos de monitorización, el uso de la bioimpedancia, una técnica que permite medir la composición corporal y otras variables clínicas importantes, plantea un reto adicional al intentar incorporar estos datos en sistemas de información sanitaria ya existentes.

Para facilitar la integración de datos entre sistemas separados, varias organizaciones han propuesto diversos estándares para el intercambio de información sanitaria a lo largo de los últimos años. Entre ellos destaca FHIR, un estándar desarrollado por HL7 (*Health Level Seven International*) que busca mejorar la interoperabilidad entre sistemas de información sanitaria, permitiendo un intercambio de datos más efectivo y seguro.

El objetivo de este trabajo es desarrollar una aplicación que sirva como intermediario entre los dispositivos de bioimpedancia y las organizaciones sanitarias facilitando la integración de los datos de bioimpedancia mediante el estándar de interoperabilidad FHIR (*Fast Healthcare Interoperability Resources*) para su posterior envío a dichas organizaciones, además de realizar un almacenamiento para la persistencia de datos.

A medida que pasan los días, la velocidad y la rapidez en todos los ámbitos de la vida aumentan, mientras que la paciencia disminuye y el tiempo se vuelve más escaso. Por lo tanto, un aspecto crucial en la gestión de datos es el ahorro de tiempo. Esta aplicación facilita la toma de decisiones clínicas al centralizar y estandarizar la información relevante, permitiendo acceder a los profesionales de la salud a datos precisos y actualizados, lo que les permite tomar decisiones informadas de manera más rápida y con mayor confianza, mejorando la calidad de la atención al paciente. La automatización en la lectura, almacenamiento y mapeo de los datos reduce el esfuerzo manual y minimiza los errores, lo que a su vez optimiza el flujo de trabajo en los entornos médicos. Esto no solo acelera los procesos, sino que también mejora la precisión de los datos manejados.

En este trabajo, se hace uso del estándar FHIR junto con su API (HAPI FHIR) para realizar el mapeo de los datos recibidos del dispositivo de bioimpedancia. De los cuatro recursos más importantes a usar, el recurso *Observation* es el recurso que destaca ya que es el que almacena toda la información referida a los datos de bioimpedancia de los pacientes, junto con los recursos *Patient* y *Device*. Una vez mapeados los datos, son enviados a un servidor FHIR público de pruebas, el cual se utiliza para simular las organizaciones sanitarias que recibirán lo datos enviados por parte de la aplicación desarrollada. Adicionalmente, los datos se persisten en una base de datos local, para la cual se ha utilizado SQLite.

Concluyendo, esta aplicación sienta las bases para futuras mejoras y expansiones, como la inclusión de servicios en tiempo real o la creación de un servidor FHIR propio, lo que lo convierte en un servicio flexible, seguro y preparado para los desafíos futuros.

Abstract

Currently, in the healthcare sector, interoperability and information exchange between systems is a significant challenge. Patient information is often distributed across multiple systems that do not always communicate with each other efficiently, making it difficult to integrate and use data appropriately. Within the sources of patient information, monitoring and testing devices have the potential to collect patient health data continuously and in heterogeneous formats. This information, for proper use in patient care, should be integrated into medical record systems. Among diagnostic techniques and monitoring devices, the use of bioimpedance, a technique for measuring body composition and other important clinical variables, poses an additional challenge when trying to incorporate these data into existing health information systems.

To make it easier to integrate data between separate systems, a number of standards for health information exchange have been proposed by various organisations in recent years. Among them is FHIR, a standard developed by HL7 (Health Level Seven International) that aims to improve interoperability between health information systems, enabling more effective and secure data exchange.

The aim of this work is to develop an application that serves as an intermediary between bioimpedance devices and healthcare organisations by enabling the integration of bioimpedance data using the FHIR (Fast Healthcare Interoperability Resources) interoperability standard for subsequent submission to these organisations, as well as storage for data persistence.

As the days go by, velocity and pace in all areas of life increase, while patience decreases and time becomes scarcer. Therefore, a crucial aspect of data management is time saving. This application facilitates clinical decision-making by centralising and standardising relevant information, giving healthcare professionals access to accurate and up-to-date data, enabling them to make informed decisions faster and with greater confidence, improving the quality of patient care. Automation in reading, storing and mapping data reduces manual effort and minimises errors, which in turn optimises workflow in medical environments. This not only speeds up processes, but also improves the accuracy of the data handled.

In this project, the FHIR standard is used as well as its API (HAPI FHIR) to map the data received from the bioimpedance device. Of the four most important resources to be used, the Observation resource is the one that stands out because it is the one that stores all the information related to the patients' bioimpedance data, as well as the Patient and Device resources. Once data is mapped, it is sent to a public FHIR test server, which is used to simulate the healthcare organisations that will receive the data sent by the developed application. Additionally, the data is persisted in a local database, for which SQLite has been used.

In conclusion, this application lays the foundations for future improvements and expansions, such as the inclusion of real-time services or the creation of its own FHIR server, making it a flexible, secure service ready for future challenges.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xv**

Índice de Tablas **xvii**

Índice de Figuras **xix**

Notación **xxii**

1 Introducción **1**
1.1 Motivación y objetivos *1*
1.2 Metodología empleada *3*
1.3 Plan de trabajo *4*

2 Estado del arte **7**
2.1 Introducción a la bioimpedancia *7*
2.1.1 Fundamentos de la bioimpedancia 7
2.1.2 Modelado de la bioimpedancia 8
2.1.3 Parámetros asociados a una medida de bioimpedancia 9
2.2 Interoperabilidad en el ámbito sanitario *10*
2.2.1 El estándar FHIR 11
2.2.2 El recurso FHIR 12
2.3 Materiales y herramientas software *19*
2.3.1 HAPI FHIR (Librería externa) 19
2.3.2 HAPI FHIR Public Test Server 19
2.3.3 Eclipse IDE 20
2.3.4 *Gradle* 20
2.3.5 *SQLite* 21

3 Resultados **23**
3.1 Descripción general del sistema *23*
3.2 Base de datos SQLite *25*
3.2.1 Modelado de la Base de Datos 25
3.2.2 Implementación de la Base de Datos 29
3.3 Recursos FHIR *32*
3.3.1 Recurso Patient 32
3.3.2 Recurso *Device* 32
3.3.3 Recurso Observation 32
3.3.4 Recurso *Bundle* 34
3.4 Implementación de la aplicación *34*
3.4.1 Estructura del proyecto 34
3.4.2 Implementación del código 35
3.4.3 Resumen del proceso de la aplicación 42
3.5 Pruebas *44*

3.5.1 Comprobación de la gestión de datos 44
3.5.2 Ejecución 44
3.5.3 Comprobación del correcto almacenamiento en la base de datos 45
3.5.4 Comprobación del correcto almacenamiento en el servidor FHIR 46

4 Conclusiones y líneas futuras 50
4.1 Conclusiones 50
4.2 Líneas futuras 51

Referencias 52

ÍNDICE DE TABLAS

Tabla 1–1. Plan de trabajo 5
Tabla 3–1 Método *"main"* de Aplicación.java 36
Tabla 3–2 Método "leer_fichero" de Lectura_Fichero.java 37
Tabla 3–3 Método *"removeTrailingQuotes"* de Lectura_Fichero.java 39
Tabla 3–4 Método "configurarRecursos"de Gestion_Fhir.java 40
Tabla 3–5 Método "configurarObservation"de Gestion_Fhir.java 41

ÍNDICE DE FIGURAS

Figura 1-1. Funcionamiento general del sistema 2
Figura 1-2. Funcionamiento de una metodología incremental 3
Figura 2-1. Impedancia (Z) 7
Figura 2-2. Modelo de circuito de un tejido biológico 8
Figura 2-3. Expresión más detallada de la impedancia (Z) 8
Figura 2-4. Gráfica Cole-Cole 9
Figura 2-5. Expresión de la impedancia en 1941 (Z) 9
Figura 2-6. Expresión final de la impedancia (Z) 9
Figura 2-7. Contenido del recurso básico *"Resource"* 12
Figura 2-8. Contenido del recurso "Meta" 12
Figura 2-9. Lista de recursos 13
Figura 2-10. Tipos de datos en los recursos FHIR 14
Figura 2-11. Estructura del recurso *Patient* 15
Figura 2-12. Estructura del recurso *Observation* 16
Figura 2-13. Estructura del recurso *Device* 17
Figura 2-14. Estructura del recurso *Bundle* 18
Figura 2-15. Logo HAPI FHIR 19
Figura 2-16. Logo Eclipse IDE 20
Figura 2-17. Eclipse IDE 20
Figura 2-18. Archivo *build.gradle* 21
Figura 2-19. Logo *SQLite* 21
Figura 3-1. Descripción general del sistema 24
Figura 3-2. Diagrama de secuencia 25
Figura 3-3. Modelo Entidad-Relación 26
Figura 3-4. Tabla "Paciente" 26
Figura 3-5. Tabla "Medición" 27
Figura 3-6. Tabla "Tripleta" 27
Figura 3-7. Tabla "Par_Bioimp" 28
Figura 3-8. Tabla "Par_Comp_Cor" 29
Figura 3-9. Código referente a la conexión con la base de datos 30
Figura 3-10. Código referente a la consulta a la base de datos 30
Figura 3-11. Código referente a la ejecución de sentencias SQL 31
Figura 3-12. Código referente a la inserción de datos en la BD 31
Figura 3-13. Parámetros de composición corporal en el recurso *Observation* 33

Figura 3-14. Uso del recurso *Bundle* 34
Figura 3-15. Paquetes de la aplicación 35
Figura 3-16. Fichero *"build.gradle"* 35
Figura 3-17. Fichero "Aplicación.java" 36
Figura 3-18. Métodos constructores de la clase "Lectura_Fichero.java" 37
Figura 3-19. Método "leer_fichero" de la clase "Lectura_Fichero.java" 38
Figura 3-20. Método *"removeTrailingQuotes"* de la clase "Lectura_Fichero.java" 39
Figura 3-21. Código referente a la conexión con el servidor FHIR 39
Figura 3-22. Creación o recuperación del recurso *Patient* 40
Figura 3-23. Creación del recurso *Observation* 41
Figura 3-24. Diagrama de flujo UML representando el proceso de la aplicación 43
Figura 3-25. Datos de pacientes en el fichero CSV 44
Figura 3-26. Base de datos vacía 44
Figura 3-27. Ejemplo de ejecución 45
Figura 3-28. Base de datos completa 45
Figura 3-29. Primera ejecución de la aplicación 46
Figura 3-30. Instancia del recurso *Patient* 46
Figura 3-31. Instancia del recurso *Device* 47
Figura 3-32. Instancia del recurso *Observation* 47
Figura 3-33. Respuesta del servidor FHIR del recurso *Patient* 48
Figura 3-34. Respuesta del servidor FHIR del recurso *Device* 48
Figura 3-35. Respuesta del servidor FHIR del recurso *Observation* 49
Figura 3-36. Segunda ejecución de la aplicación 49

Notación

API	Application Programming Interface
BBDD	Bases de Datos
CDA	Clinical Document Architecture
CSV	Comma Separated Values
FHIR	Fast Healthcare Interoperability Resources
HL7	Health Level Seven
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LOINC	Logical Observation Identifiers Names and Codes
REST	Representational State Transfer
SQL	Structured Query Language
UML	Unified Modeling Language
XML	Extensible Markup Language

1 INTRODUCCIÓN

"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber."

- Albert Einstein -

En este capítulo se ofrece una visión general del trabajo realizado, abordando desde la motivación que originó el proyecto hasta la metodología adoptada y el tiempo empleado en su desarrollo. También se muestran los objetivos a cumplir tras el conocimiento de la motivación y el plan de trabajo propuesto para ello.

1.1 Motivación y objetivos

Los avances en la tecnología están transformando todos los sectores de la sociedad, permitiendo nuevos servicios y capacidades que antes eran inimaginables. En concreto, en el dominio sanitario, los dispositivos de monitorización van a permitir la atención continuada de los pacientes, el registro de información de salud a lo largo del tiempo y con gran detalle, además de poder actuar ante eventos de riesgo en tiempo real.

Pero una problemática que nos encontramos actualmente es que los dispositivos van a ser capaces de registrar grandes cantidades de información (en tiempo real a lo largo del tiempo y de muchos pacientes) que tiene que ser integrada con la historia clínica de los pacientes para poder ser utilizada en la práctica clínica. Existe por tanto un salto (por ahora no resuelto) entre los dispositivos y los sistemas de información de la organización sanitaria, que frecuentemente implementan estándares para el intercambio de datos de salud.

La motivación de este trabajo surge de un caso particular de esta problemática: la necesidad de abordar la falta de una herramienta intermediaria capaz de recoger, almacenar y mapear a un estándar sanitario los datos generados por dispositivos de bioimpedancia, facilitando así su integración en el ámbito clínico. Actualmente, los datos de bioimpedancia son valiosos para la evaluación de la composición corporal, el estado nutricional y la salud general de los pacientes. Sin embargo, existe una brecha debido a la ausencia de una aplicación que pueda tomar estos datos, persistirlos de manera segura y estructurada, y hacerlos accesibles para los sistemas clínicos que los necesitan para el seguimiento y control de los pacientes.

El Grupo de Ingeniería Biomédica de la Universidad de Sevilla [1] ha desarrollado un dispositivo de bioimpedancia multifrecuencia que envía los datos a través del protocolo *Bluetooth* siguiendo un formato de datos propietario.

El objetivo principal de este trabajo es desarrollar una aplicación que funcione como intermediario entre dicho dispositivo de bioimpedancia y los sistemas de información sanitaria estandarizados. Como ya se ha comentado anteriormente, la aplicación debe ser capaz de recoger los datos generados por estos dispositivos, almacenarlos en una base de datos estructurada y mapearlos a un servidor FHIR (*Fast Healthcare Interoperability Resources*), permitiendo así su integración con otros registros médicos y facilitando su uso en la práctica clínica diaria.

Si bien contamos con el dispositivo de bioimpedancia desarrollado, para el diseño y desarrollo de este trabajo se ha simulado su comportamiento a través de la lectura de un fichero CSV que almacena los datos de bioimpedancia.

Así, el objetivo principal se desglosa en los siguientes subobjetivos:

- **Diseño y desarrollo de un sistema de lectura de datos:** capaz de leer y procesar datos de bioimpedancia desde un fichero CSV generado por el dispositivo de bioimpedancia.
- **Diseño e implementación de una base de datos:** para almacenar de manera estructurada los datos de bioimpedancia, asegurando su integridad y accesibilidad.
- **Estudio del estándar FHIR:** para comprender sus especificaciones y cómo puede ser aplicado en la integración de datos de bioimpedancia con sistemas clínicos.
- **Integración con un servidor FHIR:** a través del mapeo de los datos de bioimpedancia a recursos FHIR, permitiendo su integración y compatibilidad con otros sistemas de salud que utilicen el estándar FHIR.
- **Creación y gestión de recursos clínicos:** Implementación de funcionalidades para la creación de recursos clínicos en el servidor FHIR, asegurando que los datos de bioimpedancia se registren adecuadamente y sean accesibles.
- **Validación y pruebas:** Realización de pruebas exhaustivas para asegurar que la aplicación funciona correctamente y cumple con los requisitos establecidos.

En cuanto a FHIR [2], es un estándar desarrollado por HL7 para facilitar el intercambio de información sanitaria entre diferentes sistemas. FHIR utiliza una estructura basada en recursos, donde cada recurso representa un concepto específico de la información sanitaria, como pacientes (*Patient*), observaciones (*Observation*) y diagnósticos (*Diagnostic Report*). En este trabajo, utilizaremos FHIR para estructurar y mapear los parámetros de bioimpedancia a recursos clínicos que puedan ser fácilmente integrados y utilizados en los sistemas de información sanitaria.

En la Figura 1-1 se muestra un diagrama que representa el funcionamiento general del sistema cuyos elementos que lo componen son la aplicación móvil, que va a desarrollar el papel de intermediario entre el Dispositivo de bioimpedancia y el Servidor FHIR, los cuáles intercambiarán información entre ellos, y la BD en la que se cargarán y recuperarán los datos de bioimpedancia a través de la Aplicación móvil.

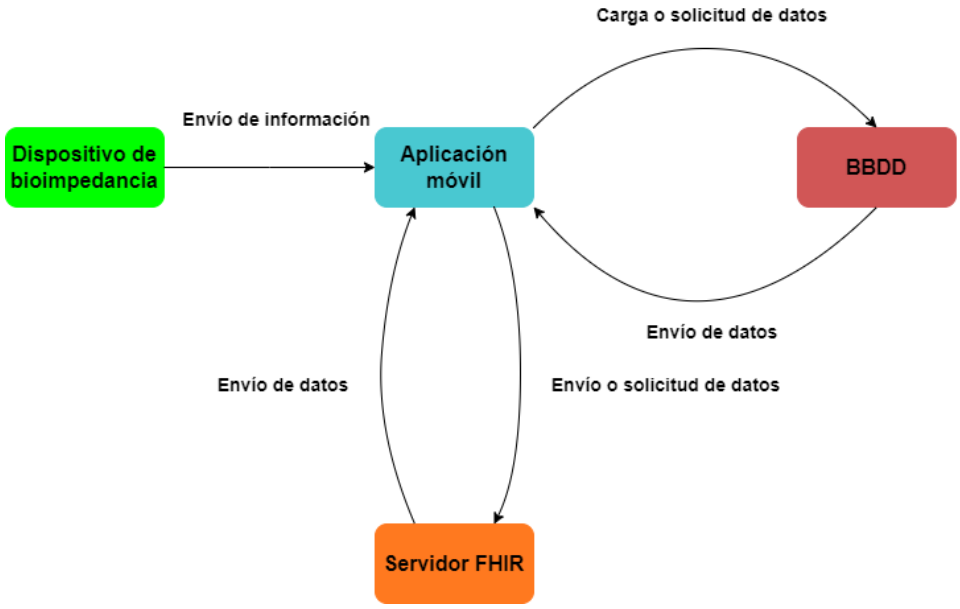

Figura 1-1. Funcionamiento general del sistema

1.2 Metodología empleada

La metología empleada para el desarrollo del trabajo está basada en el modelo incremental, que ha permitido abordar el proyecto en fases sucesivas, iterativas y acumulativas. Esta elección se debe a la necesidad de ir construyendo la aplicación de manera gradual, asegurando en cada fase la correcta implementación y funcionamiento, y permitiendo realizar ajustes y mejoras continuas a medida que se avanza hacia el producto final.

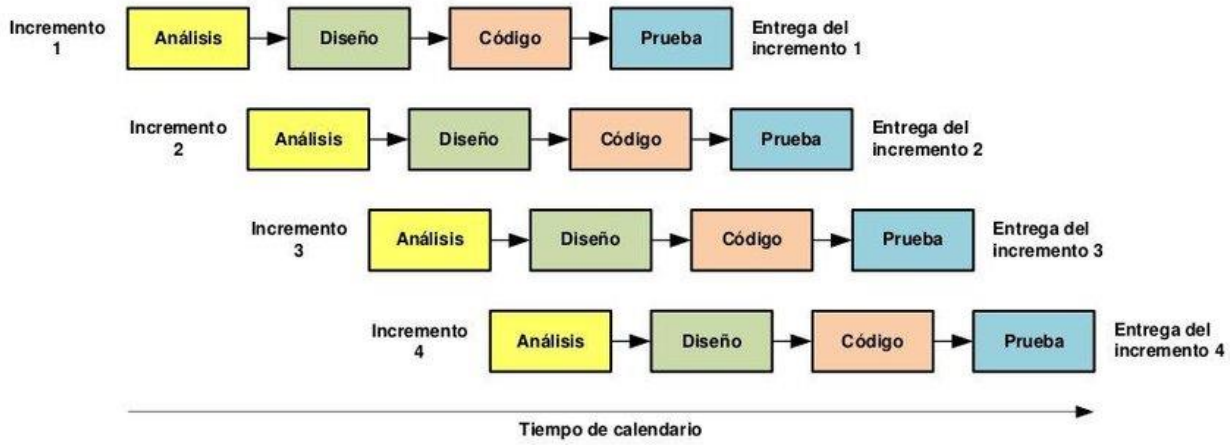

Figura 1-2. Funcionamiento de una metodología incremental

En la Figura 1-2 se pueden apreciar las fases genéricas que se suelen seguir usando en esta metodología. El primer incremento representa el desarrollo del núcleo del proyecto. Al finalizar esta etapa, se produce una retroalimentación que permite aprovechar los conocimientos y la experiencia adquiridos para progresar de forma más eficiente en la siguiente iteración. Este ciclo de retroalimentación es fundamental, ya que nos permite aprender y adaptarnos a los problemas encontrados en cada fase, facilitando así el avance en cada nueva iteración. Finalmente, el proceso iterativo culmina cuando el cliente está completamente satisfecho con la solución, garantizando un producto final con todas sus expectativas y necesidades.

1.3 Plan de trabajo

En este apartado, se detalla un desglose de las actividades planificadas para alcanzar los objetivos establecidos y según las fases indicadas en el anterior apartado:

1. **Análisis y Diseño:**
   i. **Análisis de requisitos:** Determinación de la información a tratar y resultados a obtener.
   ii. **Diseño de la arquitectura:** Basado en los requisitos identificados, se diseña la arquitectura del sistema identificando las necesidades de los distintos usuarios, las funcionalidades esperadas y las restricciones del proyecto.
2. **Desarrollo:**
   i. **Implementación del lector de ficheros CSV:** Se desarrolla un módulo capaz de leer y procesar archivos CSV que contienen los datos de bioimpedancia. Este lector extrae los datos necesarios y los prepara para su almacenamiento y posterior procesamiento.
   ii. **Desarrollo de la base de datos:** Se crea una base de datos SQL a través de herramientas software para almacenar de manera persistente los datos extraídos de los archivos CSV. Este componente asegura que los datos estén organizados.
   iii. **Integración con servidor FHIR:** En esta fase, se desarrolla la integración con un servidor FHIR público proporcionado por el propio estándar. Esto permite que los datos almacenados en la base de datos sean mapeados y enviados a este servidor.
3. **Pruebas:**
   i. **Pruebas unitarias y de integración:** Se llevan a cabo pruebas exhaustivas para asegurar la funcionalidad del sistema, como pruebas de clases creadas, resultados obtenidos y corrección del código.
   ii. **Pruebas de usuario:** Se realizan pruebas para asegurar que el sistema cumpla con las expectativas y necesidades.

A continuación, se adjunta en la Tabla 1-1 la comparativa entre la estimación de horas necesarias para la realización de cada una de las tareas y las horas que realmente se necesitaron.

Tabla 1–1. Plan de trabajo

Tareas	Horas estimadas	Horas reales
Determinación de información a tratar y resultados a obtener	2	2
Estudio y comprensión del estándar FHIR, así como los recursos que vamos a usar	5	6
Estudio de la librería HAPI-FHIR	7	8
Estudio y modelado de la base de datos	5	5
Búsqueda de librerías y dependencias para nuestro proyecto *Gradle*	4	4
Diseño de la estructura del servicio a implementar	4	6
Diseño e implementación de los objetos de información y recursos FHIR	10	14
Implementación y desarrollo de la base de datos	8	9
Implementación del lector de ficheros CSV	3	4
Implementación del módulo FHIR para el mapeo de los datos de bioimpedancia y el envío de los mismos a un servidor FHIR	11	11
Pruebas y evaluación de los resultados obtenidos	13	14
Corrección de errores ante problemas surgidos	12	14
Documentación del proyecto	50	70
Tiempo total	**134**	**167**

2 ESTADO DEL ARTE

"Aquella teoría que no encuentre aplicación práctica en la vida, es una acrobacia del pensamiento."

- Swami Vivekananda -

A continuación se va a describir la parte teórica del proyecto, abordando una introducción a la Bioimpedancia, la definición del estándar FHIR junto con su implementación, los recursos que han sido necesarios para el desarrollo del proyecto y la interoperabilidad en el ámbito sanitario. Además, trataremos las herramientas software y hardware que se han utilizado para el desarrollo del servicio a implementar.

2.1 Introducción a la bioimpedancia

La bioimpedancia es una tecnología que ha ganado relevancia en los últimos años como método de sensorización no invasivo. Permite obtener información sobre los procesos internos del cuerpo mediante la introducción de una corriente eléctrica alterna de baja intensidad en el cuerpo humano o en tejidos específicos. Esta técnica se basa en el comportamiento eléctrico de los tejidos biológicos, que pueden actuar como conductores, dieléctricos o aislantes según su composición [3].

Una de las aplicaciones más destacadas de la bioimpedancia en el ámbito de la salud es la estimación de la composición corporal. Mediante este método, es posible evaluar la nutrición y la hidratación, proporcionando información valiosa para el seguimiento clínico y la planificación de tratamientos [4].

2.1.1 Fundamentos de la bioimpedancia

La bioimpedancia es una técnica que se utiliza para evaluar la composición corporal en términos de porcentaje de grasa, músculo, huesos, órganos y agua. Se basa en la capacidad del cuerpo humano para conducir corriente eléctrica. A continuación, se detallan algunos de los aspectos clave:

- La impedancia (Z), también conocida como bioimpedancia en contextos biológicos, se define como la relación entre el voltaje (V) y la corriente elétrica (I) generada (Z = V/I) y representa la resistencia de un conductor a la circulación de una corriente eléctrica [5].
- La bioimpedancia es un número complejo que consta de dos componentes (ambas medidas dadas en Ohmios) [6][7]:
  - Resistencia eléctrica (R): Es la parte real, y se define como la oposición del medio al flujo de corriente eléctrica, siendo inversamente proporcional a la conductancia.
  - Reactancia (X_c): Es la parte imaginaria, y se define como la oposición del medio a un cambio en la corriente eléctrica. En el caso de medios biológicos, se puede considerar como inversamente proporcional a la capacitancia.

$$Z = R + jXc$$

Figura 2-1. Impedancia (Z)

La técnica de bioimpedancia consiste entonces en inyectar corrientes eléctricas a diversas frecuencias por los tejidos y medir las señales resultantes que se habrán visto afectadas por la composición de esos tejidos.

2.1.2 Modelado de la bioimpedancia

El modelado de la bioimpedancia es fundamental para comprender el comportamiento de los tejidos biológicos. Se han propuesto diversos modelos matemáticos y electrónicos para aproximarse a esta situación. Uno de los modelos más simples para este complejo comportamiento representa el flujo instantáneo en dos ramas paralelas: la parte extracelular y la parte intracelular [8].

La parte extracelular se considera puramente resistiva, mientras que la parte intracelular tiene la parte capacitiva de la membrana celular, así pues, esta última presenta una dependencia en la impedancia con la frecuencia.

El circuito más común con el que se representan los tejidos biológicos está formado por una resistencia para definir la trayectoria extracelular de la corriente (R_e) colocada en paralelo con el segundo brazo, que representa la corriente intracelular, en el que se colocan, en serie, una capacidad y una resistencia (R_i) como podemos observar en la Figura 2-2 [5].

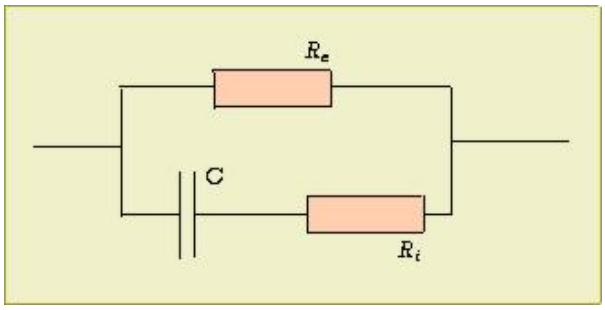

Figura 2-2. Modelo de circuito de un tejido biológico

De esta forma, la impedancia del circuito en la Figura 2-2 se corresponde con la siguiente expresión:

$$Z = R_\infty + \frac{R_0 - R_\infty}{1 + j\omega\tau}$$

Figura 2-3. Expresión más detallada de la impedancia (Z)

Donde R_∞ es el paralelo de las resistencias extracelulares e intracelulares (R_e y R_i) y R_0 es equivalente a la resistencia extracelular (R_e). En una gráfica en la que se representa la impedancia del modelo del circuito para cada una de las frecuencias desde cero a infinito, se obtiene una impedancia de curva circular cuya forma es el resultado de características eléctricas y estructurales del tejido. El modelo nos lleva a que la posición de la impedancia en la gráfica se mueve en el sentido contrario a las agujas del reloj en un semicírculo de R_0 a R_∞ [5]. Es lo que se conoce como la gráfica Cole-Cole la Figura 2-4.

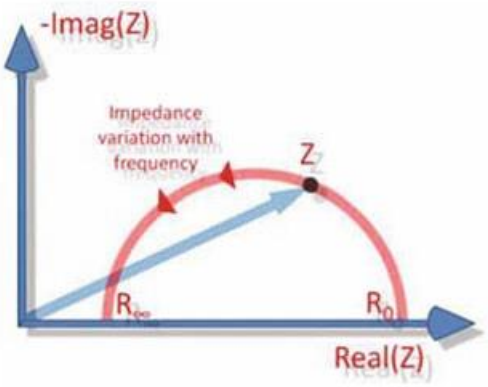

Figura 2-4. Gráfica Cole-Cole

En 1928, Cole descubrió que las mediciones de impedancia de materiales biológicos tenían una forma circular, pero su centro estaba ubicado debajo del eje de resistencia [8]. En 1941, Cole y Cole mostraron que, cuando hay una distribución de las constantes de tiempo, la impedancia compleja se puede presentar con la siguiente expresión de la Figura 2-5.

$$Z = R_\infty + \frac{R_0 - R_\infty}{1 + (j\omega\tau)^{(1-\alpha)}}$$

Figura 2-5. Expresión de la impedancia en 1941 (Z)

Tras varios años de investigaciones y pruebas con el modelo Cole-Cole incluyeron en dicha expresión un parámetro característico de la distribución de la relajación de frecuencias (α), y un retardo de tiempo invariante con la frecuencia (T_d). Finalmente, la expresión de la Figura 2-6 es la que se utiliza [5][8]:

$$Z = \left(R_\infty + \frac{R_0 - R_\infty}{1 + (j\omega\tau)^{(1-\alpha)}}\right) e^{-j\omega T_d}$$

Figura 2-6. Expresión final de la impedancia (Z)

2.1.3 Parámetros asociados a una medida de bioimpedancia

Los parámetros medidos por bioimpedancia se detallan a continuación:

- Parámetros antropométricos:
  - Peso (Kg)
  - Estatura (cm)
  - Fecha de nacimiento (Día, mes y año)

- Parámetros de bioimpedancia:
  - Frecuencia (kHz)
  - Módulo (Ω)
  - Fase (º)

- Modelo de bioimpedancia:
  - R_0 (Ω)
  - $R\infty$ (Ω)
  - R_e (Ω)
  - R_i (Ω)
  - C (nF)
  - α
  - T_D (ns)

Con los parámetros antropométricos, los parámetros de bioimpedancia y el modelo de bioimpedancia podemos obtener los parámetros de composición corporal.

- Parámetros de composición corporal:
  - Masa celular corporal (BCM, del inglés *Body Cell Mass*) (kg)
  - % BCM (%)
  - Volumen de agua extracelular (ECW, del inglés *Extra-Cellular Water*) (l)
  - Volumen de agua intracelular (ICW, del inglés *Intra-Cellular Water*) (l)
  - Agua corporal total (TBW, del inglés *Total Body Water*) (l)
  - Masa libre de grasa (FFM, del inglés *Fat-Free Mass*) (kg)
  - % FFM (%)
  - Índice de masa libre de grasa (FFMI, del inglés *Fat Free Mass Index*) (kg/m^2)
  - Masa grasa (FM, del inglés *Fat Mass*) (kg)
  - % FM (%)
  - Índice de masa grasa (FMI, del inglés *Fat Mass Index*) (kg/m^2)

2.2 Interoperabilidad en el ámbito sanitario

La interoperabilidad en el ámbito sanitario es esencial para garantizar que los datos de salud se puedan compartir y utilizar de manera eficaz entre diferentes sistemas y organizaciones. Este aspecto es crucial para mejorar la calidad de la atención al paciente, facilitar la investigación y optimizar la eficiencia del sistema de salud.

Podemos definir la "interoperabilidad en el ámbito sanitario" como la capacidad de los sistemas de información y los procedimientos que soportan, de compartir datos y permitir el intercambio de información y conocimiento entre ellos [9]. Esta interoperabilidad es crucial hoy en día, ya que cada organización puede tener sus sistemas implementados de diversas maneras, lo que complica la comunicación entre ellos. Por lo tanto, se necesita un marco de referencia que unifique esta información, facilitando la comunicación. Aquí es donde entran los estándares. Estos estándares proporcionan un denominador común para los componentes utilizados en el ámbito sanitario y para la forma en que se transmiten, asegurando que sean útiles para terceros.

En nuestro caso, nos vamos a centrar en el estándar FHIR.

2.2.1 El estándar FHIR

El estándar FHIR combina las mejores características de los estándares HL7 v2 [10], HL7 v3 [11] y CDA [12]. HL7 es el ejemplo perfecto de la fusión de la tecnología y la sanidad. Es un promotor que hace más sencillo la interoperabilidad entre todas las partes del mundo. Es una organización de desarrollo de estándares para el ámbito de la salud, fundada en 1987 que opera a nivel internacional y su misión es proveer estándares globales para los dominios clínicos, asistencial, administrativo y logístico para conseguir una comunicación entre los diferentes sistemas de información en el área de la salud. HL7 fue quien dio forma y definió FHIR.

FHIR es una especificación de plataforma que describe formatos y elementos de datos, también conocidos como recursos. Estos (los recursos) son la unidad básica de interoperabilidad y la más pequeña que se puede intercambiar de manera significativa. Todos los recursos, salvo excepciones, contienen un identificador a través del cual puede ser registrado, localizado y recuperado. Además, los recursos disponen de extensiones que permite a los implementadores agregar fácilmente nuevas propiedades.

Por otro lado, FHIR se caracteriza como una especificación "*RESTful*" porque utiliza el protocolo REST para la comunicación. Esto implica que se emplean mensajes HTTP request/response para el envío y recepción de información, la cual se transmite en estructuras JSON o XML. Además, la API RESTful [13] define un conjunto de operaciones conocidas como interacciones para gestionar los diferentes recursos. Una interacción particularmente destacada es *Search*, que permite buscar el recurso deseado.

En cuanto a la seguridad de FHIR [14], es importante señalar que FHIR no es un protocolo de seguridad ni define funciones específicas relacionadas con la seguridad. Sin embargo, sí establece una serie de protocolos de intercambio y modelos de contenido que deben ser utilizados junto con varios protocolos de seguridad para garantizar un uso seguro. Los protocolos de intercambio y los modelos de contenido que se utilizan para garantizar la seguridad incluyen:

- *HTTPS (HTTP Secure)*
- *OAuth 2.0*
- *SMART on FHIR*

} Protocolos de intercambio

- *AuditEvent*
- *Consent*
- *Provenance*
- *Security Labels*

} Modelos de contenido

2.2.2 El recurso FHIR

Un recurso FHIR es la unidad fundamental de intercambio en el estándar FHIR. Representa una pieza de información clínica o administrativa, estructurada de manera que pueda ser fácilmente compartida y entendida por diferentes sistemas de salud. Cada recurso contiene una estructura de datos definida, con atributos específicos que describen una entidad del mundo real, como un paciente, una observación, una receta, etc.

Según se define en la página oficial de FHIR, una instancia de recurso es una entidad que:

- tiene una identidad conocida (una URL) por la que se puede dirigir
- se identifica como uno de los tipos de recurso definidos en la presente especificación
- contiene un conjunto de elementos de datos estructurados según se describe en la definición del tipo de recurso
- tiene una versión identificada que cambia si el contenido del recurso cambia

La Figura 2-7 muestra los campos comunes a los recursos FHIR. Estos son:

- Un identificador con el que podremos recuperar el recurso a posteriori. Este podrá ser una *"Uniform Resource Locator"* (URL) que cambiará a medida que cambie la localización del recurso o podrá formar parte del mismo recurso de tal forma que no cambiará.
- Metadatos para describir al propio recurso cuyos campos podemos observar en la Figura 2-8.
- Una referencia a una guía de implementación que describa cómo se está usando el recurso.
- Puede tener un elemento de lenguaje que especifica el idioma que tendrán los datos del recurso. Si este se usa, también deberá especificarse en el elemento narrativo.

Name	Flags	Card.	Type	Description & Constraints
Resource	«A» N		Base	Base Resource Elements defined in Ancestors:
id	Σ	0..1	id	Logical id of this artifact
meta	Σ	0..1	Meta	Metadata about the resource
implicitRules	?! Σ	0..1	uri	A set of rules under which this content was created
language		0..1	code	Language of the resource content

Figura 2-7. Contenido del recurso básico *"Resource"*

Name	Flags	Card.	Type	Description & Constraints
Meta	Σ N		Element	Metadata about a resource Elements defined in Ancestors: id, extension
versionId	Σ	0..1	id	Version specific identifier
lastUpdated	Σ	0..1	instant	When the resource version last changed
source	Σ	0..1	uri	Identifies where the resource comes from
profile	Σ	0..*	canonical(StructureDefinition)	Profiles this resource claims to conform to
security	Σ	0..*	Coding	Security Labels applied to this resource Binding: All Security Labels (Extensible)
tag	Σ	0..*	Coding	Tags applied to this resource Binding: Common Tags (Example)

Figura 2-8. Contenido del recurso "Meta"

En la Figura 2-9 podemos observar la gran variedad de recursos existentes, que se definen y van desarrollando con diferentes grados de mandurez en el que el mínimo es "0" y el máximo es "N" (*"normative"*) [15]. Cuando poseen un grado "N" quiere decir que el recurso en cuestión no va a sufrir más modificaciones.

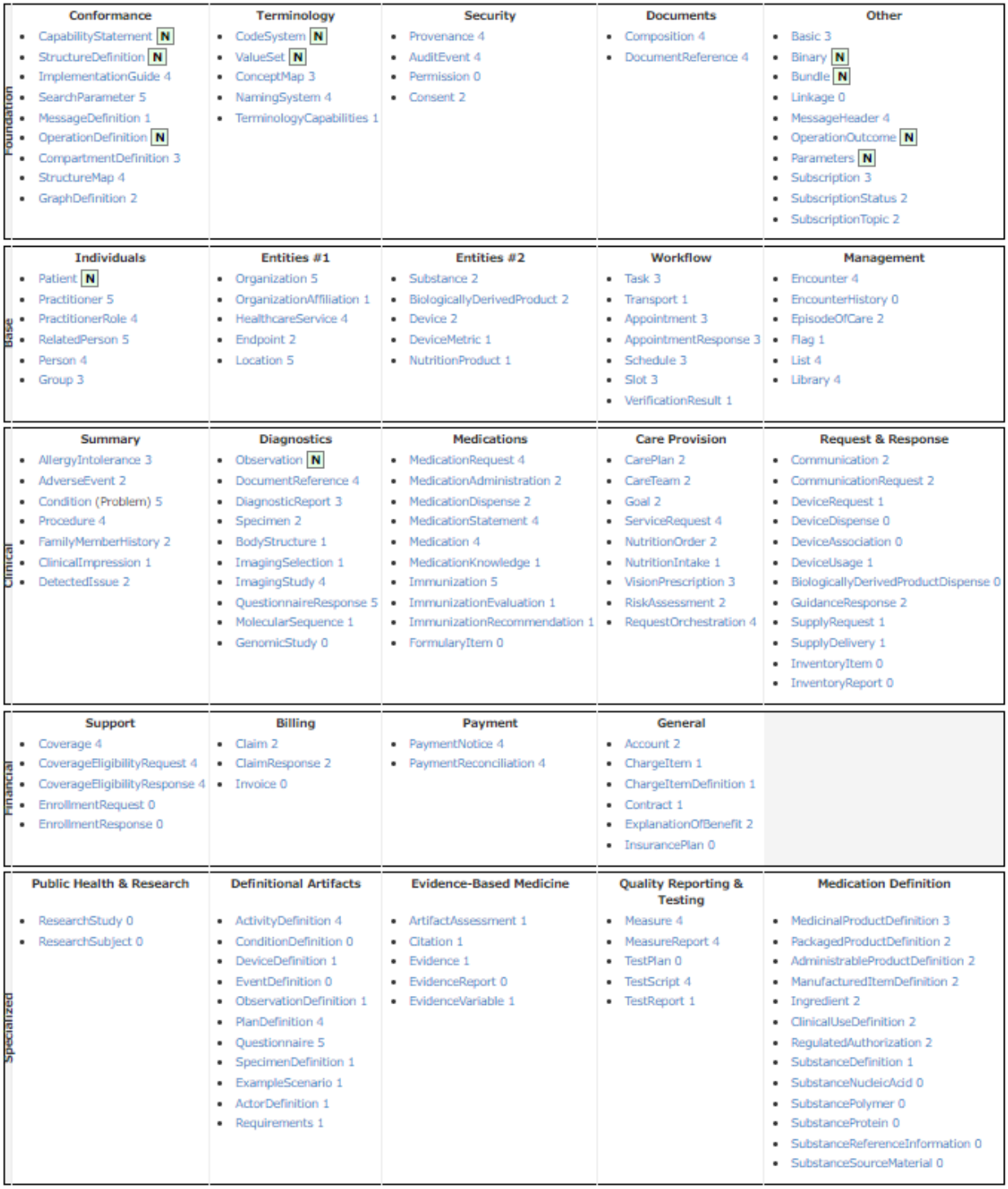

Figura 2-9. Lista de recursos

Además de los elementos anteriores, los recursos poseen otros elementos que almacenan información. Cada tipo de recurso tendrá una serie de campos que aportarán la información por la que usamos un recurso determinado. Estos campos se podrán completar con un tipo de dato concreto. La especificación define 4 categorías en las que clasifica estos tipos [16]:

- Simples o primitivos (*string o integer*)
- Complejos (*CodeableConcept* o *ContactPoint*)
- Metadatos (*Contributor* o *UsageContext*)
- De uso especial (*Reference* o *Extension*)

En la Figura 2-10 podemos observar los tipos de cada categoría:

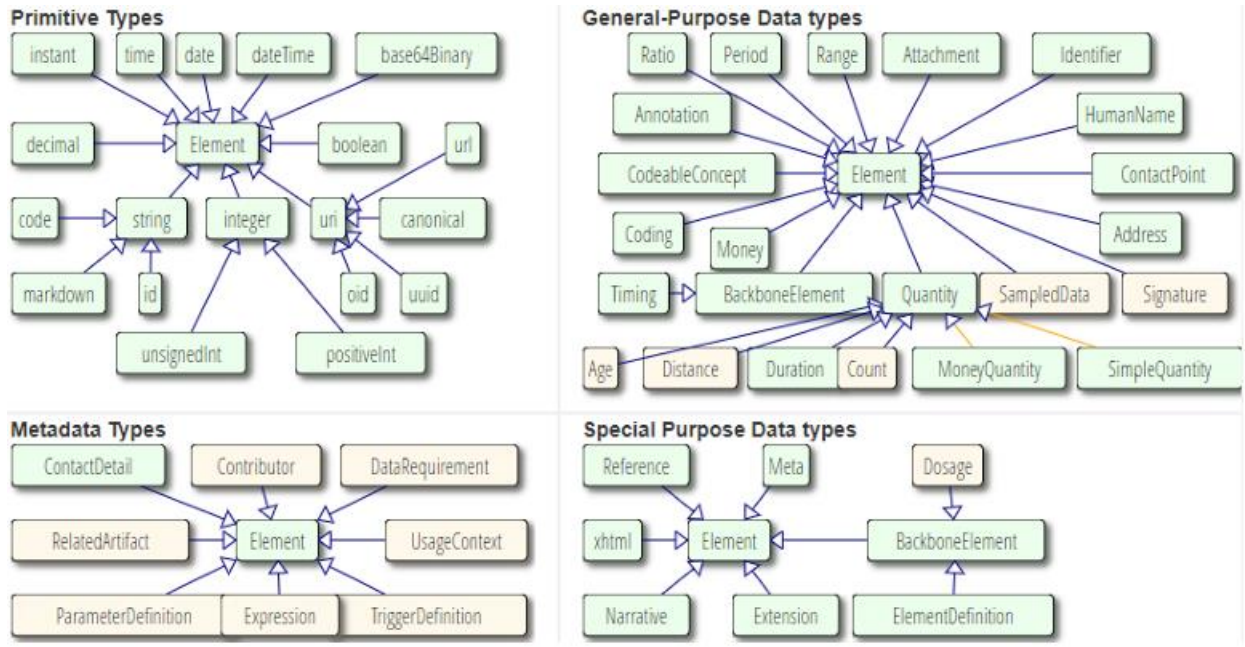

Figura 2-10. Tipos de datos en los recursos FHIR

2.2.2.1 Recursos usados

Para el desarrollo de la aplicación de bioimpedancia, se han utilizado varios recursos FHIR clave para gestionar y compartir la información de bioimpedancia: *Patient, Observation, Device y Bundle.*

- **Recurso *Patient***

El recurso *Patient* [17] es uno de los más fundamentales en FHIR y se utiliza para representar información sobre un individuo que recibe atención médica. Incluye detalles como el nombre, la fecha de nacimiento, el género, los identificadores únicos, y otros datos demográficos relevantes.

En la Figura 2-11 tenemos la estructura del recurso con todos los campos de información que el estándar aporta.

Name	Flags	Card.	Type	Description & Constraints
Patient	N		DomainResource	Information about an individual or animal receiving health care services Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	An identifier for this patient
active	?! Σ	0..1	boolean	Whether this patient's record is in active use
name	Σ	0..*	HumanName	A name associated with the patient
telecom	Σ	0..*	ContactPoint	A contact detail for the individual
gender	Σ	0..1	code	male | female | other | unknown Binding: AdministrativeGender (Required)
birthDate	Σ	0..1	date	The date of birth for the individual
deceased[x]	?! Σ	0..1		Indicates if the individual is deceased or not
deceasedBoolean			boolean	
deceasedDateTime			dateTime	
address	Σ	0..*	Address	An address for the individual
maritalStatus		0..1	CodeableConcept	Marital (civil) status of a patient Binding: Marital Status Codes (Extensible)
multipleBirth[x]		0..1		Whether patient is part of a multiple birth
multipleBirthBoolean			boolean	
multipleBirthInteger			integer	
photo		0..*	Attachment	Image of the patient
contact	C	0..*	BackboneElement	A contact party (e.g. guardian, partner, friend) for the patient *+ Rule: SHALL at least contain a contact's details or a reference to an organization*
relationship		0..*	CodeableConcept	The kind of relationship Binding: Patient Contact Relationship (Extensible)
name	C	0..1	HumanName	A name associated with the contact person
telecom	C	0..*	ContactPoint	A contact detail for the person
address	C	0..1	Address	Address for the contact person
gender		0..1	code	male | female | other | unknown Binding: AdministrativeGender (Required)
organization	C	0..1	Reference(Organization)	Organization that is associated with the contact
period		0..1	Period	The period during which this contact person or organization is valid to be contacted relating to this patient
communication		0..*	BackboneElement	A language which may be used to communicate with the patient about his or her health
language		1..1	CodeableConcept	The language which can be used to communicate with the patient about his or her health Binding: All Languages (Required) **Additional Bindings** | Purpose Common Languages | Starter Set
preferred		0..1	boolean	Language preference indicator
generalPractitioner		0..*	Reference(Organization | Practitioner | PractitionerRole)	Patient's nominated primary care provider
managingOrganization	Σ	0..1	Reference(Organization)	Organization that is the custodian of the patient record
link	?! Σ	0..*	BackboneElement	Link to a Patient or RelatedPerson resource that concerns the same actual individual
other	Σ	1..1	Reference(Patient | RelatedPerson)	The other patient or related person resource that the link refers to
type	Σ	1..1	code	replaced-by | replaces | refer | seealso Binding: Link Type (Required)

Figura 2-11. Estructura del recurso *Patient*

En nuestro caso se usará el recurso *Patient* para identificar cada paciente mediante el DNI del ciudadano e identificarlo, y para registrar datos demográficos como el género y la fecha de nacimiento del paciente, ya que son cruciales para contextualizar las observaciones clínicas y personalizar la atención médica.

- **Recurso *Observation***

El recurso *Observation* [18] se usa para registrar y almacenar datos de observaciones y mediciones clínicas realizadas sobre un paciente. Puede incluir cualquier tipo de información clínica, como signos vitales, resultados de laboratorio, y otros datos cuantitativos y cualitativos.

La mayoría de los campos de este recurso son simples aserciones de pares de nombre/valor con algunos metadatos, salvo algunas observaciones que agrupan otras observaciones.

En la Figura 2-12 tenemos parte de la estructura del recurso *Observation* con todos los campos de información que el estándar aporta.

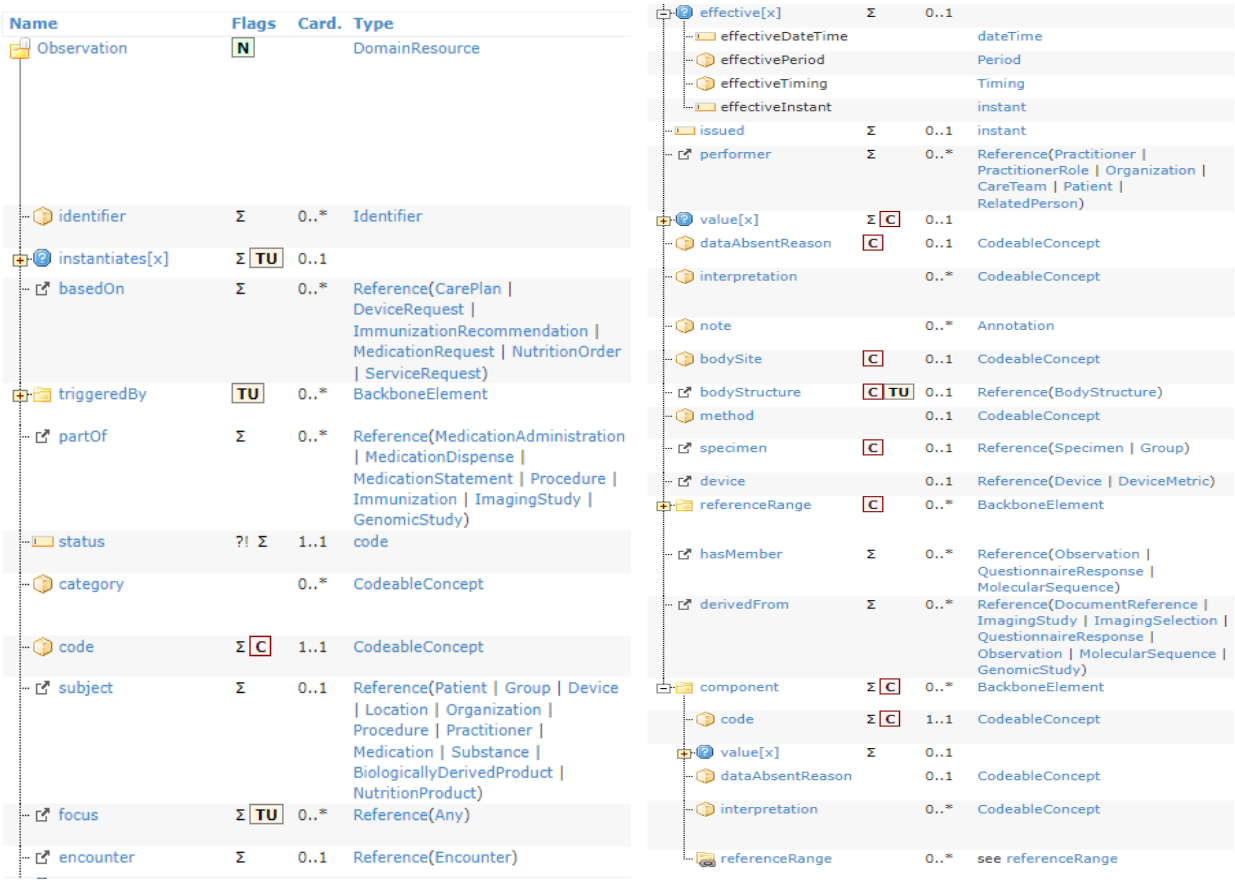

Figura 2-12. Estructura del recurso *Observation*

Este recurso se utilizará para el registro de mediciones de bioimpedancia, incluyendo parámetros como la masa celular corporal (BCM), el porcentaje de masa corporal (BCM%), el agua extracelular (ECW), el agua intracelular (ICW), y otros datos derivados de los dispositivos de bioimpedancia. Además, se usará el recurso *Observation* para el almacenamiento de valores específicos, como la altura del paciente, capturando no solo el valor númerico sino también el contexto, como la unidad de medida (cm) y el sistema de codificación utilizado (LOINC [19] *o* SNOMED [20]).

LOINC y SNOMED CT son dos sistemas de terminología médica ampliamente utilizados a nivel mundial para estandarizar y codificar información clínica. LOINC es un sistema de codificación desarrollado para identificar de manera única las observaciones y los resultados de laboratorio, y SNOMED es un sistema de terminología integral que cubre todos los aspectos de la atención médica. Ambos sistemas son esenciales para la interoperabilidad en la atención sanitaria, ya que permiten que los datos clínicos sean compartidos, comprendidos y utilizados de manera eficiente a nivel global.

Como podemos observar en la Figura 2-12, los datos de información clínica mencionados anteriormente no existen literalmente en la estructura del recurso, por lo que para almacenar este tipo de información haremos uso del campo *"Value"* de dicha estructura, el cual nos permite guardar distintos valores médicos según el formato.

- **Recurso *Device***

Este recurso describe un dispositivo médico utilizado en la atención al paciente, y puede incluir detalles como el tipo de dispositivo, el fabricante, el número de serie, y otra información relevante.

En la Figura 2-13 tenemos la estructura del recurso *Device* [21] con todos los campos de información que el estándar aporta.

En nuestro proyecto, usaremos el recurso *Device* para gestionar el registro de dispositivos de bioimpedancia, identificando los dispositivos utilizados para capturar las mediciones de los pacientes, asegurando así la trazabilidad y el contexto de las observaciones. Además, aparte de la identificación de los dispositivos, se configurará el estado del mismo. Para ello se usarán los campos *"identifier"* y *"status"* que podemos observar en la estructura del recurso de la Figura 2-13.

Name	Flags	Card.	Type	Description & Constraints
Device	TU		DomainResource	Item used in healthcare *+ Rule: only one Device.name.display SHALL be true when there is more than one Device.name* Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier		0..*	Identifier	Instance identifier
displayName		0..1	string	The name used to display by default when the device is referenced
definition		0..1	CodeableReference(DeviceDefinition)	The reference to the definition for the device
udiCarrier	Σ	0..*	BackboneElement	Unique Device Identifier (UDI) Barcode string
deviceIdentifier	Σ	1..1	string	Mandatory fixed portion of UDI
issuer	Σ	1..1	uri	UDI Issuing Organization
jurisdiction		0..1	uri	Regional UDI authority
carrierAIDC	Σ	0..1	base64Binary	UDI Machine Readable Barcode String
carrierHRF	Σ	0..1	string	UDI Human Readable Barcode String
entryType		0..1	code	barcode | rfid | manual | card | self-reported | electronic-transmission | unknown Binding: UDI Entry Type (Required)
status	?! Σ	0..1	code	active | inactive | entered-in-error Binding: FHIR Device Status (Required)
availabilityStatus		0..1	CodeableConcept	lost | damaged | destroyed | available Binding: FHIR Device Availability Status (Extensible)
biologicalSourceEvent		0..1	Identifier	An identifier that supports traceability to the event during which material in this product from one or more biological entities was obtained or pooled
manufacturer		0..1	string	Name of device manufacturer
manufactureDate		0..1	dateTime	Date when the device was made
expirationDate		0..1	dateTime	Date and time of expiry of this device (if applicable)
lotNumber		0..1	string	Lot number of manufacture
serialNumber		0..1	string	Serial number assigned by the manufacturer
name	C	0..*	BackboneElement	The name or names of the device as known to the manufacturer and/or patient
value	Σ	1..1	string	The term that names the device
type	Σ	1..1	code	registered-name | user-friendly-name | patient-reported-name Binding: Device Name Type (Required)
display	?! Σ C	0..1	boolean	The preferred device name
modelNumber		0..1	string	The manufacturer's model number for the device
partNumber		0..1	string	The part number or catalog number of the device
category		0..*	CodeableConcept	Indicates a high-level grouping of the device Binding: FHIR Device Category (Example)
type		0..*	CodeableConcept	The kind or type of device Binding: Device Type (Example)
version		0..*	BackboneElement	The actual design of the device or software version running on the device
type		0..1	CodeableConcept	The type of the device version, e.g. manufacturer, approved, internal Binding: Device Version Type (Example)
component		0..1	Identifier	The hardware or software module of the device to which the version applies
installDate		0..1	dateTime	The date the version was installed on the device
value		1..1	string	The version text
conformsTo		0..*	BackboneElement	Identifies the standards, specifications, or formal guidances for the capabilities supported by the device
category		0..1	CodeableConcept	Describes the common type of the standard, specification, or formal guidance. communication | performance | measurement Binding: Device Specification Category (Example)
specification		1..1	CodeableConcept	Identifies the standard, specification, or formal guidance that the device adheres to Binding: Device Specification Type (Example)
version		0..1	string	Specific form or variant of the standard
property		0..*	BackboneElement	Inherent, essentially fixed, characteristics of the device. e.g., time properties, size, material, etc.
type		1..1	CodeableConcept	Code that specifies the property being represented Binding: Device Property Type (Example)
value[x]		1..1		Value of the property
valueQuantity			Quantity	
valueCodeableConcept			CodeableConcept	
valueString			string	
valueBoolean			boolean	
valueInteger			integer	
valueRange			Range	
valueAttachment			Attachment	
mode		0..1	CodeableConcept	The designated condition for performing a task Binding: FHIR Device Operation Mode (Example)
cycle		0..1	Count	The series of occurrences that repeats during the operation of the device
duration		0..1	Duration	A measurement of time during the device's operation (e.g., days, hours, mins, etc.)
owner		0..1	Reference(Organization)	Organization responsible for device
contact		0..*	ContactPoint	Details for human/organization for support
location		0..1	Reference(Location)	Where the device is found
url		0..1	uri	Network address to contact device
endpoint		0..*	Reference(Endpoint)	Technical endpoints providing access to electronic services provided by the device
gateway		0..*	CodeableReference(Device)	Linked device acting as a communication/data collector, translator or controller
note		0..*	Annotation	Device notes and comments
safety	Σ	0..*	CodeableConcept	Safety Characteristics of Device Binding: Device Safety (Example)
parent		0..1	Reference(Device)	The higher level or encompassing device that this device is a logical part of

Figura 2-13. Estructura del recurso *Device*

El recurso *Bundle* [22] es una colección de recursos relacionados que se agrupan para ser procesados de manera conjunta. Los *Bundle* son útiles para realizar transacciones, intercambiar conjuntos de datos, o representar una colección de recursos en una sola unidad. Esto en FHIR se denomina "agrupación" de recursos.

Bundle es un recurso FHIR que se encuentra dentro del grupo "Otros" en la zona de "Fundación", cuyo grado de madurez es el máximo, es decir, el grado *"Normative"*. Dado que un paquete en sí mismo es un recurso, este recurso *Bundle* tiene los mismos metadatos comunes que todos los recursos, incluyendo afirmaciones de perfiles, etiquetas y etiquetas de seguridad.

En la Figura 2-14 podemos ver la estructura de este recurso con todos los campos correspondientes que FHIR nos aporta.

Name	Flags	Card.	Type	Description & Constraints
Bundle	N		Resource	Contains a collection of resources + Rule: total only when a search or history + Rule: entry.search only when a search + Rule: FullUrl must be unique in a bundle, or else entries with the same fullUrl must have different meta.versionId (except in history bundles) + Rule: A document must have an identifier with a system and a value + Rule: A document must have a date + Rule: A document must have a Composition as the first resource + Rule: A message must have a MessageHeader as the first resource + Rule: A subscription-notification must have a SubscriptionStatus as the first resource + Rule: entry.request.method PATCH not allowed for history + Rule: Bundle resources where type is not transaction, transaction-response, batch, or batch-response or when the request is a POST SHALL have Bundle.entry.fullUrl populated + Rule: Issue.severity for all issues within the OperationOutcome must be either 'information' or 'warning'. + Rule: Use and meaning of issues for documents has not been validated because the content will not be rendered in the document. + Rule: Self link is required for searchsets. + Rule: For collections of type document, message, searchset or collection, all entries must contain resources, and not have request or response elements + Rule: For collections of type history, all entries must contain request or response elements, and resources if the method is POST, PUT or PATCH + Rule: For collections of type transaction or batch, all entries must contain request elements, and resources if the method is POST, PUT or PATCH + Rule: For collections of type transaction-response or batch-response, all entries must contain response elements Elements defined in Ancestors: id, meta, implicitRules, language
identifier	Σ C	0..1	Identifier	Persistent identifier for the bundle
type	Σ C	1..1	code	document | message | transaction | transaction-response | batch | batch-response | history | searchset | collection | subscription-notification Binding: Bundle Type (Required)
timestamp	Σ C	0..1	instant	When the bundle was assembled
total	Σ C	0..1	unsignedInt	If search, the total number of matches
link	Σ C	0..*	BackboneElement	Links related to this Bundle
relation	Σ C	1..1	code	See http://www.iana.org/assignments/link-relations/link-relations.xhtml#link-relations-1 Binding: Link Relation Types (Required)
url	Σ C	1..1	uri	Reference details for the link
entry	Σ C	0..*	BackboneElement	Entry in the bundle - will have a resource or information + Rule: must be a resource unless there's a request or response + Rule: fullUrl cannot be a version specific reference This repeating element order: For bundles of type 'document' and 'message', the first resource is special (must be Composition or MessageHeader respectively). For all bundles, the meaning of the order of entries depends on the bundle type
link	Σ	0..*	see link	Links related to this entry
fullUrl	Σ C	0..1	uri	URI for resource (e.g. the absolute URL server address, URI for UUID/OID, etc.)
resource	Σ C	0..1	Resource	A resource in the bundle
search	Σ C	0..1	BackboneElement	Search related information
mode	Σ	0..1	code	match | include - why this is in the result set Binding: Search Entry Mode (Required)
score	Σ	0..1	decimal	Search ranking (between 0 and 1)
request	Σ C	0..1	BackboneElement	Additional execution information (transaction/batch/history)
method	Σ C	1..1	code	GET | HEAD | POST | PUT | DELETE | PATCH Binding: H T T P Verb (Required)
url	Σ	1..1	uri	URL for HTTP equivalent of this entry
ifNoneMatch	Σ	0..1	string	For managing cache validation
ifModifiedSince	Σ	0..1	instant	For managing cache currency
ifMatch	Σ	0..1	string	For managing update contention
ifNoneExist	Σ	0..1	string	For conditional creates
response	Σ C	0..1	BackboneElement	Results of execution (transaction/batch/history)
status	Σ	1..1	string	Status response code (text optional)
location	Σ	0..1	uri	The location (if the operation returns a location)
etag	Σ	0..1	string	The Etag for the resource (if relevant)
lastModified	Σ	0..1	instant	Server's date time modified
outcome	Σ	0..1	Resource	OperationOutcome with hints and warnings (for batch/transaction)
signature	Σ TU	0..1	Signature	Digital Signature
issues	Σ C TU	0..1	Resource	Issues with the Bundle

Figura 2-14. Estructura del recurso *Bundle*

En nuestro caso el recurso *Bundle* se utilizará para comprobar la existencia de pacientes y dispositivos específicos en la base de datos de FHIR. Esto lo realizaremos mediante una consulta que buscará pacientes por su identificador único (DNI) y dispositivos por su identificador de dispositivo correspondiente. Finalmente el *Bundle* resultante contendrá los recursos *Patient* y *Device* que coincidan con los criterios de búsqueda. Con esto nos aseguraremos de que si existe un paciente con un DNI concreto en los servidores FHIR o un dispositivo clínico con un identificador específico se use la referencia ya existente de esos recursos *Patient* o *Device*, en caso contrario habrá que crear dichos recursos y solicitar la referencia de estos al servidor FHIR.

2.3 Materiales y herramientas software

Para el desarrollo de este proyecto, se han utilizado diversas herramientas software que facilitan la implementación y el manejo de los recursos FHIR, así como la gestión de los datos de bioimpedancia.

2.3.1 HAPI FHIR (Librería externa)

Una biblioteca Java de código abierto para el estándar FHIR que facilita la creación, manipulación y transmisión de recursos FHIR. HAPI FHIR [23] proporciona una API RESTful que simplifica la integración con servidores FHIR.

En nuestro proyecto, HAPI FHIR se ha utilizado para configurar los recursos *Patient* y *Observation*, y posteriormente ser enviados mediante la API RESTful al servidor que HAPI FHIR ofrece [24] para realizar pruebas de este estilo.

Figura 2-15. Logo HAPI FHIR

2.3.2 HAPI FHIR Public Test Server

La API de FHIR también ofrece un servidor público [24] para la realización de pruebas así como una base de datos FHIR pública. En este caso, se ha utilizado este servidor de FHIR público para simular las organizaciones sanitarias a las que se envían los datos de bioimpedancia de los pacientes.

No se ha usado un servidor FHIR propio debido a la limitación de tiempo para este proyecto. Este servidor FHIR propio sería el resultado de un servidor que implementaría el estándar FHIR y los métodos necesarios para la escucha y respuesta a las peticiones por parte de los clientes, como ya lo hace el servidor de FHIR público.

2.3.3 Eclipse IDE

Se trata de un entorno de desarrollo integrado que soporta Java y otros lenguajes de programación. Eclipse se utiliza para escribir y depurar el código del proyecto [25].

Es un entorno de desarrollo que te permite organizar los proyectos en distintos paquetes según su funcionalidad y así poder seguir un orden de códigos implementados. En nuestro caso, la aplicación se ha dividido en dos bloques, uno dirigido a la parte de la aplicación en sí, la gestión de los recursos FHIR y la lectura del fichero CSV, y el segundo bloque se ha dedicado a la gestión de la base de datos externa que se ha implementado para este trabajo.

Figura 2-16. Logo Eclipse IDE

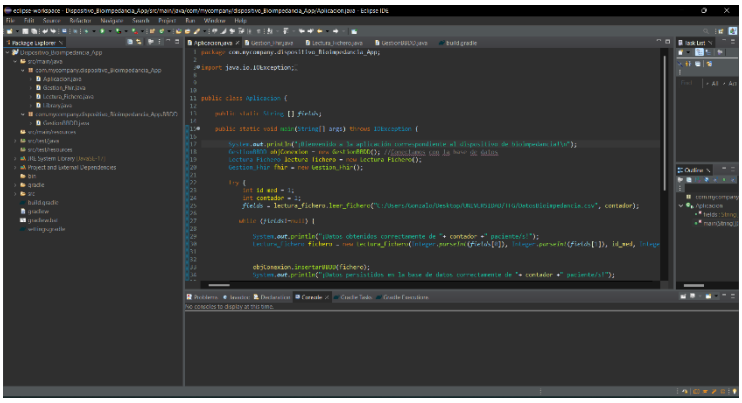

Figura 2-17. Eclipse IDE

2.3.4 *Gradle*

Gradle [26] es una herramienta de automatización de compilaciones que permite gestionar de manera eficiente la construcción, prueba y despliegue de aplicaciones. A diferencia de otras herramientas como *Maven* [27], *Gradle* utiliza un lenguaje de script basado en *Groovy* o *Kotlin*.

Una de las principales ventajas de *Gradle* es su capacidad para manejar proyectos complejos, lo que facilita la administración de dependencias, la organización de tareas y la creación de procesos automatizados. *Gradle* permite definir las tareas y dependencias del proyecto en un archivo denomidado *"build.gradle"*.

En el desarrollo de este proyecto, *Gradle* ha sido utilizado para gestionar todas las dependencias y la configuración del entorno de desarrollo. A través del archivo *"build.gradle"* se han añadido las dependencias necesarias para trabajar con la especificación FHIR utilizando la biblioteca HAPI FHIR, además de otras dependencias relacionadas con la base de datos y otras herramientas.

En la Figura 2-18 se puede observar lo detallado con anterioridad.

```
build.gradle
repositories {
    // Use Maven Central for resolving dependencies.
    mavenCentral()
    maven {
        url "https://oss.sonatype.org/content/repositories/snapshots"
    }
}

dependencies {
    // Use JUnit Jupiter for testing.
    testImplementation 'org.junit.jupiter:junit-jupiter:5.9.1'

    // This dependency is exported to consumers, that is to say found on their compile classpath.
    api 'org.apache.commons:commons-math3:3.6.1'

    // This dependency is used internally, and not exposed to consumers on their own compile classpath.
    implementation 'com.google.guava:guava:31.1-jre'

    implementation 'ca.uhn.hapi.fhir:hapi-fhir-base:7.0.2'
    implementation 'ca.uhn.hapi.fhir:hapi-fhir-structures-r5:7.0.2'
    implementation 'ca.uhn.hapi.fhir:hapi-fhir-client:7.0.2'

    implementation 'org.slf4j:slf4j-api:2.0.13'
    implementation 'org.slf4j:slf4j-simple:2.0.13'

    implementation files('C:/Users/Gonzalo/Desktop/UNIVERSIDAD/TFG/Base_de_Datos/sqliteadmin/sqlite-jdbc-3.8.11.2.jar')

}

// Apply a specific Java toolchain to ease working on different environments.
```

Figura 2-18. Archivo *build.gradle*

2.3.5 *SQLite*

Es un sistema de gestión de bases de datos relacional ligero y autónomo que no requiere un servidor separado para funcionar [28]. *SQLite* almacena toda la base de datos en un solo archivo en el disco, lo que lo hace fácil de implementar y mover entre diferentes entornos.

En este proyecto, se ha utilizado *SQLite* como el sistema de base de datos para almacenar y gestionar los datos de bioimpedancia recogidos por el dispositivo, y información relevante de los pacientes y dispositivos.

La base de datos se manipula a través de consultas SQL estándar, lo que facilita la gestión de los datos almacenados y permite realizar operaciones de consulta, inserción, actualización y eliminación de forma sencilla y eficiente.

Figura 2-19. Logo *SQLite*

3 RESULTADOS

"Los científicos investigan lo que ya es; los ingenieros crean lo que nunca ha sido."

- Albert Einstein -

En este capítulo se presenta una visión completa del desarrollo y los logros alcanzados a lo largo del proyecto. Aquí se detallarán todos los pasos realizados desde la concepción del sistema hasta la implementación y pruebas finales. No solo se mostrarán los resultados obtenidos, sino también el proceso llevado a cabo para alcanzarlos, incluyendo una descripción detallada del sistema, la arquitectura empleada, lor problemas encontrados, y las soluciones implementadas.

3.1 Descripción general del sistema

El sistema desarrollado en este trabajo tiene como objetivo centralizar y gestionar los datos obtenidos de dispositivos de bioimpedancia. Estos dispositivos, aunque simulados en este caso mediante la lectura de datos clínicos de los pacientes desde un fichero CSV, generan información crítica que necesita ser persistida, procesada, y finalmente integrada en un sistema clínico de información sanitaria para su uso en la toma de decisiones médicas.

El sistema está compuesto por varios módulos que trabajan de manera coordinada para cumplir con las funcionalidades requeridas:

- **Módulo de Ingesta de Datos:** Este módulo se encarga de la lectura de los datos de los pacientes desde un fichero CSV.
- **Módulo de Persistencia de Datos:** Una vez procesados, los datos son almacenados en una base de datos *SQLite*.
- **Módulo de Interoperabilidad FHIR:** Este es uno de los núcleos del sistema, encargado de mapear los datos obtenidos y almacenados al formato FHIR, permitiendo así que sean compatibles y transferibles entre diferentes sistemas de información clínica.
- **Módulo de Comunicación y API:** Este módulo gestiona las comunicaciones con el servidor FHIR, utilizando operaciones RESTful para enviar y recibir datos en formato JSON o XML.

La solución propuesta no se limita al uso de un fichero CSV como fuente de datos. Esta implementación simula la función del dispositivo de bioimpedancia, pero el diseño y arquitectura del sistema permiten fácilmente la integración con dispositivos reales en un entorno de producción. Esta flexibilidad asegura que la solución sea adaptable a futuros avances tecnológicos o cambios en el entorno operativo.

En la Figura 3-1 podemos observar la identificación de estos módulos sobre la Figura 1-1 del capítulo 1, apartado en el que explicamos el funcionamiento general del sistema apoyándonos sobre un esquema representativo de la arquitectura del mismo.

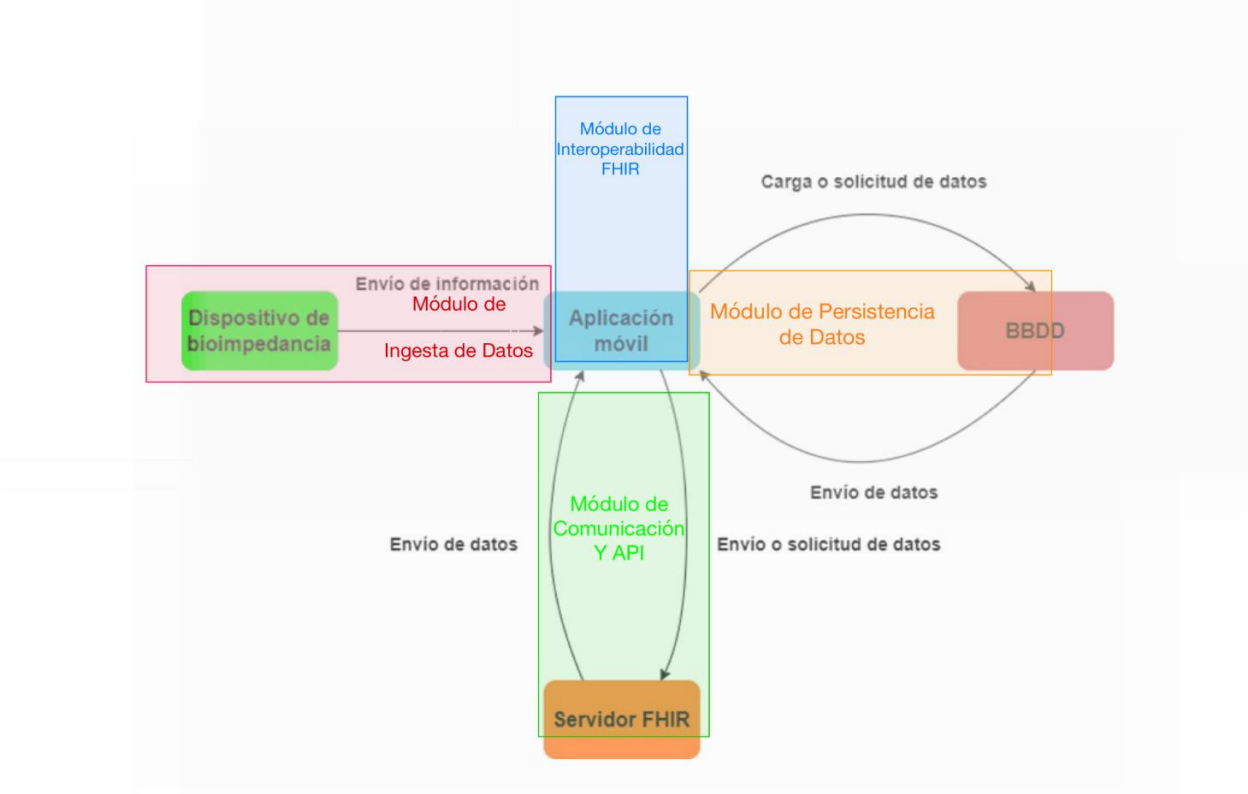

Figura 3-1. Descripción general del sistema

En la Figura 3-2 se puede observar claramente el orden en que se realizan las tareas con la participación del dispositivo de bioimpedancia, nuestra aplicación, la base de datos *SQLite* y el servidor FHIR.

En esta Figura nos centramos más en la interacción de la aplicación con el servidor FHIR, es decir, el módulo de comunicación y API descrito anteriormente. En primer lugar, tenemos que conectarnos con el servidor FHIR a través de una conexión REST. A continuación, se procedería a realizar las consultas de la existencia de los recursos *Patient* y *Device* al servidor FHIR. Y por último se configura el recurso *Observation* y se envían todos los recursos configurados al servidor.

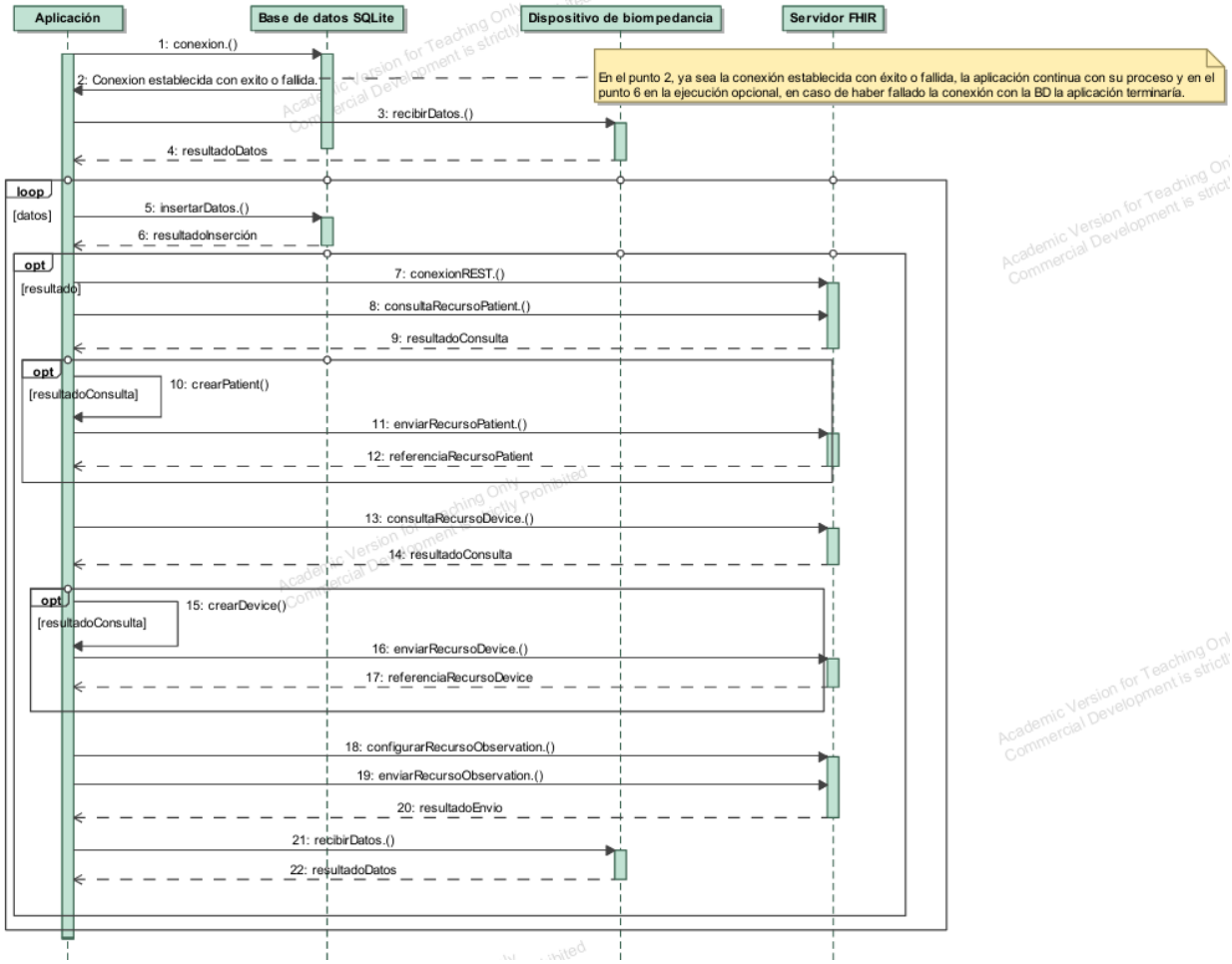

Figura 3-2. Diagrama de secuencia

3.2 Base de datos *SQLite*

En esta sección se detalla el diseño y la implementación de la base de datos *SQLite* [28] que se ha utilizado en el proyecto. La base de datos se encarga de almacenar de manera estructurada los datos clínicos obtenidos del dispositivo de bioimpedancia, que como ya hemos mencionado anteriormente son simulados mediante la lectura del fichero CSV. A continuación, se describen las tablas principales que componen la base de datos, y además se analizará el código implementado en Eclipse IDE [25] que gestiona la interacción con la base de datos.

3.2.1 Modelado de la Base de Datos

La base de datos se ha diseñado siguiendo un modelo entidad-relación, con el objetivo de organizar de manera eficiente la información relacionada con los pacientes, los dispositivos de bioimpedancia, y las observaciones clínicas.

A continuación, en la Figura 3-3 se presenta un esquema de la base de datos que sigue el modelo entidad-relación realizado con anterioridad a la implementación de la misma con el objetivo de definir qué parámetros por cada tabla eran los que iban a ser únicos y no nulos. Este esquema muestra las relaciones entre las diferentes tablas, y asegura la integridad de los datos y facilita las operaciones de consulta y análisis.

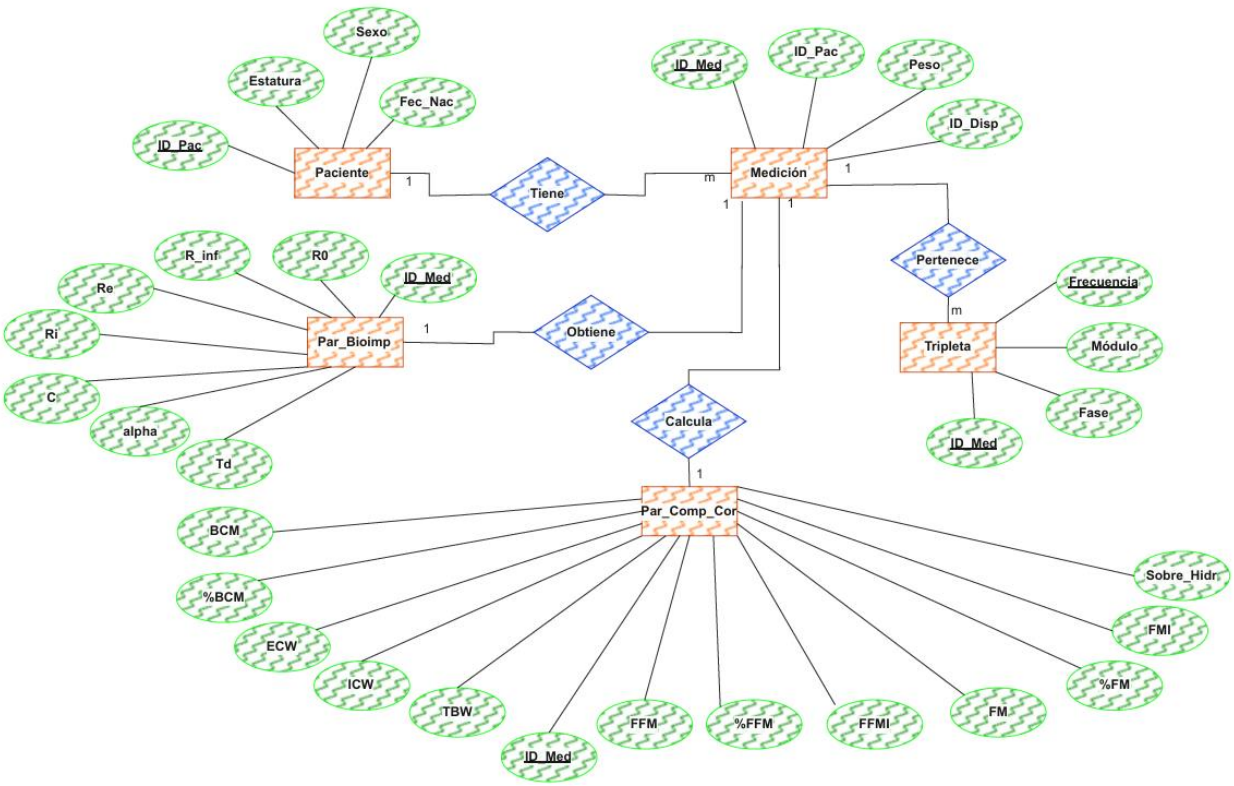

Figura 3-3. Modelo Entidad-Relación

A continuación se explicará en detalle cada una de las tablas con sus parámetros correspondientes. Las tablas que forman parte de la base de datos son:

- **Tabla "Paciente"**

Almacena información básica de los pacientes, incluyendo su identificador único, fecha de nacimiento, y otros datos relevantes.

- "ID_Pac": Identificador único del paciente, que en este caso se ha utilizado el DNI del paciente.
- "Fec_Nac": Fecha de nacimiento del paciente.
- "Sexo": Género del paciente.
- "Estatura": Estatura del paciente, medida en centímetros.

```
CREATE TABLE [Pacientes] (
[ID_Pac] INTEGER  PRIMARY KEY NULL,
[Sexo] VARCHAR(1)  NULL,
[Fec_Nac] DATE  NULL,
[Estatura] FLOAT  NULL
)
```

ID_Pac	Sexo	Fec_Nac	Estatura
18562210	M	01/04/1990	163
18563310	F	24/07/1980	175
22567315	M	12/06/1975	185

Figura 3-4. Tabla "Paciente"

➢ **Tabla "Medición"**

Almacena información referente a las mediciones realizadas sobre los pacientes, como la fecha de realización de la medición, el identificador del paciente y del dispositivo, y el peso del paciente.

- "ID_Pac": Identificador único del paciente, que en este caso es el DNI del paciente.
- "Peso": Pesaje del paciente a estudio, medido en kilogramos.
- "ID_Disp": Identificador único del dispositivo de bioimpedancia con el que se realiza la observación.
- "ID_Med": Identificador único de la observación realizada al paciente.

```
CREATE TABLE [Medicion] (
[ID_Med] INTEGER  PRIMARY KEY AUTOINCREMENT NOT NULL,
[ID_Pac] INTEGER  NULL,
[ID_Disp] INTEGER  NULL,
[Peso] FLOAT  NULL
)
```

ID_Med	ID_Pac	ID_Disp	Peso
1	18562210	12546	65
2	18563310	12557	70
3	22567315	13457	82

Figura 3-5. Tabla "Medición"

Un paciente puede tener varias mediciones (Figura 3-3), por lo que en la tabla "Medición" un paciente podría tener varias entradas con distintos identificadores de medición pero con el mismo identificador de paciente.

➢ **Tabla "Tripleta"**

Tabla que se utiliza para almacenar las distintas frecuencias con sus respectivos módulo y fase en las que se realiza las mediciones de la observación.

- "ID_Med": Identificador único de la observación realizada al paciente.
- "Frecuencia": Frecuencia de la impedancia medida en kHz.
- "Módulo": Módulo de la impedancia medido en ohmios.
- "Fase": Fase de la impedancia medida en grados.

```
CREATE TABLE [Tripleta] (
[ID_Med] INTEGER  NOT NULL,
[Fase] INTEGER  NULL,
[Modulo] FLOAT  NULL,
[Frecuencia] INTEGER  NOT NULL,
PRIMARY KEY ([ID_Med],[Frecuencia])
)
```

ID_Med	Fase	Modulo	Frecuencia
1	45	5,8	5000
1	39	6,2	8000
1	19	10,8	12000
1	23	7,9	15000

Figura 3-6. Tabla "Tripleta"

Disponemos de 22 frecuencias distintas para obtener el modelo de bioimpedancia que calcula los datos de bioimpedancia que se necesitan. Por tanto, por cada entrada en la tabla "Medición" tenemos 22 entradas en la tabla "Tripleta", una por cada frecuencia (Figura 3-3).

➢ **Tabla "Par_Bioimp"**

Tabla que almacena los parámetros de bioimpedancia obtenidos a partir de la frecuencia, módulo y fase para formar el modelo de bioimpedancia.

- "ID_Med": Identificador único de la observación realizada al paciente.
- "R0": Resistencia a frecuencia cero, medida en ohmios.
- "R_inf". Resistencia a frecuencia infinita, medida en ohmios.
- "Re": Resistencia extracelular, medida en ohmios.
- "Ri": Resistencia intracelular, medida en ohmios.
- "C": Capacidad asociada a las membranas celulares, medida en nanofaradios.
- "alpha": Parámetro característico de la distribución de las frecuencias de relajación. Es un parámetro adimensional.
- "Td": Retraso temporal invariante con la frecuencia, medido en nanosegundos.

```
CREATE TABLE [Par_Bioimp] (
[ID_Med] INTEGER  NOT NULL PRIMARY KEY AUTOINCREMENT,
[R0] INTEGER  NULL,
[R_inf] FLOAT  NULL,
[Re] INTEGER  NULL,
[Ri] FLOAT  NULL,
[C] FLOAT  NULL,
[alpha] FLOAT  NULL,
[Td] INTEGER  NULL
)
```

ID_Med	R0	R_inf	Re	Ri	C	alpha	Td
1	1500	900	1500	2250	1,1	0,1	5
2	1300	700	1300	2050	2,1	0,7	3
3	1250	820	1250	2125	3,2	0,4	7

Figura 3-7. Tabla "Par_Bioimp"

La tabla "Par_Bioimp" tiene una relación 1:1 con la tabla "Medición" (Figura 3-3), es decir, por cada medición realizada a un paciente se tendría una entrada en la tabla "Par_Bioimp".

➢ **Tabla "Par_Comp_Cor"**

Almacena los distintos parámetros de composición corporal obtenidos gracias a los parámetros de bioimpedancia como resultado de las mediciones de la observación a los pacientes.

- "ID_Med": Identificador único de la observación realizada al paciente.
- "BCM": Masa celular corporal.
- "%BCM": Porcentaje de masa celular corporal.
- "ECW": Volumen de agua extracelular.
- "ICW": Volumen de agua intracelular.
- "TBW": Agua corporal total.
- "FFM": Masa libre de grasa.
- "%FFM": Porcentaje de masa libre de grasa.
- "FFMI": Índice de masa libre de grasa.

- "FM": Masa grasa.
- "%FM": Porcentaje de masa grasa.
- "FMI": Índice de masa grasa.
- "Sobre_Hidr": Sobrehidratación.

```
CREATE TABLE [Par_Comp_Cor] (
[ID_Med] INTEGER  PRIMARY KEY AUTOINCREMENT NOT NULL,
[BCM] INTEGER  NULL,
[Porcentaje_BCM] INTEGER  NULL,
[ECW] FLOAT  NULL,
[ICW] FLOAT  NULL,
[TBW] FLOAT  NULL,
[FFM] INTEGER  NULL,
[Porcentaje_FFM] INTEGER  NULL,
[FFMI] FLOAT  NULL,
[FM] INTEGER  NULL,
[Porcentaje_FM] INTEGER  NULL,
[FMI] FLOAT  NULL,
[Sobre_Hidra] FLOAT  NULL
)
```

ID_Med	BCM	Porcentaje_BCM	ECW	ICW	TBW	FFM	Porcentaje_FFM	FFMI	FM	Porcentaje_FM	FMI	Sobre_Hidra
1	39	26	29,25	53,18182	82,431816	52	80	19,571682	13	20	4,8929205	0,5
2	49	23	27,35	43,191917	80,45182	55	70	29,571682	10	19	5,79292	0,7
3	52	18	21,25	57,121212	78,67182	49	67	24,681982	15	16	3,8939204	0,4

Figura 3-8. Tabla "Par_Comp_Cor"

La relación de la tabla "Par_Comp_Cor" con la tabla "Medición" es la misma que tiene la tabla "Par_Bioimp", una relación 1:1 (Figura 3-3).

3.2.2 Implementación de la Base de Datos

El manejo de la base de datos en este proyecto se ha implementado a través de código Java utilizando el IDE de Eclipse. La interacción con la base de datos *SQLite* se realiza mediante consultas SQL, que son gestionadas por clases y métodos dedicados dentro del proyecto. A continuación, se describe cómo se ha estructurado el código y los principales métodos utilizados para la gestión de la base de datos.

- **Conexión a la Base de Datos:**
  - El primer paso para interactuar con la base de datos es establecer una conexión. Para ello, se utiliza la clase *"SQLiteConnection"*, que configura la conexión a la base de datos *SQLite* a través de la ruta del archivo que contiene la base de datos. Esta función se ha implementado en el método constructor de la clase "GestionBBDD":

```java
public class GestionBBDD {
    String strConexionBD="jdbc:sqlite:C:/Users/Gonzalo/Desktop/UNIVERSIDAD/TFG/Base_de_Datos/Aplicacion.s3db";
    Connection conn= null;

    public GestionBBDD() {
        try {
            // Cargar la clase del driver JDBC
            Class.forName("org.sqlite.JDBC");

            // Intentar establecer la conexión
            conn = DriverManager.getConnection(strConexionBD);

            // Verificar si la conexión fue exitosa
            if (conn != null) {
                System.out.println("Conexión a la base de datos establecida con éxito.");
            } else {
                System.out.println("Conexión a la base de datos fallida.");
            }
        } catch (SQLException e) {
            // Manejo de la excepción SQLException
            System.out.println("Error al intentar conectar a la base de datos.");
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // Manejo de la excepción ClassNotFoundException
            System.out.println("Error: No se encontró el driver JDBC.");
            e.printStackTrace();
        }
    }
```

Figura 3-9. Código referente a la conexión con la base de datos

- **Consulta y Recuperación de Datos:**
  - Se han implementado métodos para realizar consultas a la base de datos, como la búsqueda de un paciente o dispositivo específico. Estos métodos devuelven los datos solicitados y se utilizan para verificar si los registros ya existen antes de insertar nuevos datos. Para ello se ha utilizado el método "consultarRegistros":

```java
public ResultSet consultarRegistros(String strSentenciaSQL){

    try {
        PreparedStatement pstm = conn.prepareStatement(strSentenciaSQL);
        ResultSet respuesta= pstm.executeQuery();
        return respuesta;

    } catch (Exception e) {
        System.out.println(e);
        return null;
    }

}
```

Figura 3-10. Código referente a la consulta a la base de datos

- **Ejecución de sentencias SQL:**
  - Uno de los componentes clave del sistema es un método genérico que permita ejecutar sentencias SQL, que podría ser utilizado tanto para la consulta de registros como para la inserción de datos. Este método centraliza la ejecución de operaciones SQL, facilitando la reutilización del código y mejorando la mantenibilidad del sistema. A continuación, se muestra el método "ejecutarSentenciaSQL" creado para dicha función:

```java
public int ejecutarSentenciaSQL(String strSentenciaSQL){

    try {
        PreparedStatement pstm = conn.prepareStatement(strSentenciaSQL);
        pstm.execute();
        return 1;

    } catch (SQLException e) {
        System.out.println(e);
        return 0;
    }
}
```

Figura 3-11. Código referente a la ejecución de sentencias SQL

- **Inserción de datos:**
  - Los datos obtenidos del fichero CSV se insertan en las correspondientes tablas mediante sentencias *"INSERT"*. Estas operaciones están encapsuladas en métodos específicos dentro de la clase que gestiona cada entidad. Para ejecutar estas sentencias *"INSERT"* se ha utilizado el método "ejecutarSentenciaSQL" mencionado anteriormente. Para dicha función se ha creado el método "insertarBBDD":

```java
public int insertarBBDD(Lectura_Fichero fichero){
    int resultado;
    try {

        resultado = ejecutarSentenciaSQL("INSERT INTO Pacientes (ID_Pac, Sexo, Fec_Nac, Estatura) VALUES ("+fichero.getId_pac()+","+fichero.getSexo()+","+ "\'" + fichero.getFec_nac() + "\'" + ",
        resultado = ejecutarSentenciaSQL("INSERT INTO Medicion (ID_Med, ID_Pac, ID_Disp, Peso) VALUES ("+fichero.getId_med()+","+fichero.getId_pac()+","+fichero.getId_disp()+","+fichero.getPeso(
        resultado = ejecutarSentenciaSQL("INSERT INTO Par_Bioimp (ID_Med, R0, R_inf, Re, Ri, C, alpha, Td) VALUES ("+fichero.getId_med()+","+fichero.getR0()+","+fichero.getR_inf()+","+fichero.ge
        resultado = ejecutarSentenciaSQL("INSERT INTO Par_Comp_Cor (ID_Med, BCM, Porcentaje_BCM, ECW, ICW, TBW, FFM, Porcentaje_FFM, FFMI, FM, Porcentaje_FM, FMI, Sobre_Hidra) VALUES ("+fichero.
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase1()+","+fichero.getModulo1()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase2()+","+fichero.getModulo2()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase3()+","+fichero.getModulo3()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase4()+","+fichero.getModulo4()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase5()+","+fichero.getModulo5()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase6()+","+fichero.getModulo6()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase7()+","+fichero.getModulo7()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase8()+","+fichero.getModulo8()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase9()+","+fichero.getModulo9()+","+fichero.getFre
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase10()+","+fichero.getModulo10()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase11()+","+fichero.getModulo11()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase12()+","+fichero.getModulo12()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase13()+","+fichero.getModulo13()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase14()+","+fichero.getModulo14()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase15()+","+fichero.getModulo15()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase16()+","+fichero.getModulo16()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase17()+","+fichero.getModulo17()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase18()+","+fichero.getModulo18()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase19()+","+fichero.getModulo19()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase20()+","+fichero.getModulo20()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase21()+","+fichero.getModulo21()+","+fichero.getF
        resultado = ejecutarSentenciaSQL("INSERT INTO Tripleta (ID_Med, Fase, Modulo, Frecuencia) VALUES ("+fichero.getId_med()+","+fichero.getFase22()+","+fichero.getModulo22()+","+fichero.getF

        // Verificar si la inserción de datos fue exitosa
        if (resultado == 1) {
            System.out.println("La inserción de datos en la base de datos fue realizada con éxito.");
        } else {
            System.out.println("Inserción de datos en la base de datos fallida.");
        }

        return resultado;
```

Figura 3-12. Código referente a la inserción de datos en la BD

3.3 Recursos FHIR

A continuación se van a describir los campos de los recursos FHIR usados en detalle. Con ello, se comprobará cúales han sido usados para el desarrollo del proyecto.

3.3.1 Recurso Patient

El recurso *Patient* [17] representa a una persona que recibe atención médica en un contexto clínico. En este proyecto, se utiliza para almacenar la información básica de los pacientes cuyos datos de bioimpedancia están siendo gestionados. Los campos utilizados en este recurso son:

- *"Identifier"*: Este campo se usa para asignar un identificador único al paciente, basado en el ID extraído del fichero CSV que como ya se mencionó anteriormente es su DNI.
- *"Gender"*: Se configura el género del paciente utilizando el campo *"Gender"*, que acepta valores como *"MALE"*, *"FEMALE"*, y *"UNKNOWN"*. Esta información se extrae del fichero CSV, que simula la recepción de datos desde el dispositivo de bioimpedancia.
- *"BirthDate"*: Este campo almacena la fecha de nacimiento del paciente, también obtenida del fichero CSV.

El uso de estos campos es crucial para identificar al paciente y gestionar su información en el sistema FHIR [2], facilitando su localización y manipulación en operaciones posteriores.

3.3.2 Recurso *Device*

El recurso *Device* [21] se utiliza para representar el dispositivo de bioimpedancia que recopila los datos del paciente. Este recurso permite gestionar los dispositivos involucrados en las mediciones. Los campos utilizados en este recurso son:

- *"Identifier"*: Similar al recurso *Patient,* este campo se utiliza para asignar un identificador único al dispositivo, basado en el ID del dispositivo que se extrae del fichero CSV. Esto asegura que cada dispositivo sea único en la base de datos.
- *"Status"*: El estado del dispositivo se configura utilizando el campo *"Status"*, que se establece como *"ACTIVE"*. Este campo indica que el dispositivo está operativo y en uso para realizar mediciones.

La correcta configuración de este recurso garantiza que los datos de bioimpedancia se asocien adecuadamente con el dispositivo que realizó la medición, lo cual es esencial para el seguimiento y trazabilidad en el contexto clínico.

3.3.3 Recurso Observation

El recurso *Observation* [18] se utiliza para almacenar las mediciones de bioimpedancia obtenidas de los pacientes, en concreto solo se almacenan en este recurso los parámetros de composición corporal. En el contexto del proyecto, este recurso es fundamental para registrar y manejar los datos biométricos. Los campos utilizados son:

- *"Status"*: El estado de la observación se configura como *"FINAL"*, indicando que la medición ha sido completada y no es provisional.
- *"Code"*: Este campo se utliza para especificar el tipo de medición realizada, como *"Body height"* para la estatura, ya que ni en el recurso *Patient* ni en el recurso *Observation* existe un campo específico para indicar la estatura del paciente. Los códigos se extraen de sistemas estándar como LOINC [19] o SNOMED [20].
- *"Value"*: Aquí se almacena el valor de la medición anterior (por ejemplo, la estatura en centímetros). Se utiliza el campo *"Quantity*" para almacenar el valor numérico y su unidad de medida.
- *"Effective"*: Este campo almacena la fecha y hora en que se realizó la observación, utilizando el tipo de datos *"DateTimeType"*.

- *"Subject"*: Se establece una referencia al recurso *Patient* que corresponde al paciente asociado con la observación.
- *"Device"*: Se configura una referencia al recurso *Device* que representa el dispositivo utilizado para realizar la medición.

Adicionalmente, se utiliza el campo *"Component"* para almacenar mediciones detalladas de bioimpedancia. En concreto se incluyen los parámetros de composición corporal como la Masa Corporal Magra (FFM), Agua Corporal Total (TBW), y el resto de parámetros corporales, todos ellos configurados dentro del recurso *Observation*. Para ello se ha hecho uso de los campos como ya hemos mencionado *"Component"*, y además los campos *"Code"*, *"Value"* y *"Quantity"*. Podemos observar en la Figura 3-13 como se ha llevado a cabo este proceso:

```
// %FFM
Observation.ObservationComponentComponent percentFfmComponent = new Observation.ObservationComponentComponent();
percentFfmComponent.getCode().addCoding().setSystem("http://example.com").setCode("%FFM").setDisplay("Percentage of Fat-Free Mass");
percentFfmComponent.setValue(new Quantity().setValue(fichero.getPctj_ffm()).setUnit("%"));
observation.addComponent(percentFfmComponent);

// FFMI (Fat-Free Mass Index)
Observation.ObservationComponentComponent ffmiComponent = new Observation.ObservationComponentComponent();
ffmiComponent.getCode().addCoding().setSystem("http://example.com").setCode("FFMI").setDisplay("Fat-Free Mass Index");
ffmiComponent.setValue(new Quantity().setValue(fichero.getFfmi()).setUnit("kg/m²"));
observation.addComponent(ffmiComponent);

// FM (Fat Mass)
Observation.ObservationComponentComponent fmComponent = new Observation.ObservationComponentComponent();
fmComponent.getCode().addCoding().setSystem("https://www.snomed.org/").setCode("5941000237109").setDisplay("Fat Mass");
fmComponent.setValue(new Quantity().setValue(fichero.getFm()).setUnit("kg"));
observation.addComponent(fmComponent);

// %FM
Observation.ObservationComponentComponent percentFmComponent = new Observation.ObservationComponentComponent();
percentFmComponent.getCode().addCoding().setSystem("http://example.com").setCode("%FM").setDisplay("Percentage of Fat Mass");
percentFmComponent.setValue(new Quantity().setValue(fichero.getPctj_fm()).setUnit("%"));
observation.addComponent(percentFmComponent);

// FMI (Fat Mass Index)
Observation.ObservationComponentComponent fmiComponent = new Observation.ObservationComponentComponent();
fmiComponent.getCode().addCoding().setSystem("http://example.com").setCode("FMI").setDisplay("Fat Mass Index");
fmiComponent.setValue(new Quantity().setValue(fichero.getFmi()).setUnit("kg/m²"));
observation.addComponent(fmiComponent);

// Sobre Hidratación
Observation.ObservationComponentComponent shComponent = new Observation.ObservationComponentComponent();
shComponent.getCode().addCoding().setSystem("https://www.snomed.org/").setCode("61688009").setDisplay("Overhydration");
shComponent.setValue(new Quantity().setValue(fichero.getSobre_hidra()).setUnit("L"));
observation.addComponent(shComponent);

// BCM (Body Cell Mass)
Observation.ObservationComponentComponent bcmComponent = new Observation.ObservationComponentComponent();
bcmComponent.getCode().addCoding().setSystem("http://example.com").setCode("BCM").setDisplay("Body Cell Mass");
bcmComponent.setValue(new Quantity().setValue(fichero.getBcm()).setUnit("kg"));
observation.addComponent(bcmComponent);

// %BCM
Observation.ObservationComponentComponent percentBcmComponent = new Observation.ObservationComponentComponent();
percentBcmComponent.getCode().addCoding().setSystem("http://example.com").setCode("%BCM").setDisplay("Percentage of Body Cell Mass");
percentBcmComponent.setValue(new Quantity().setValue(fichero.getPctj_bcm()).setUnit("%"));
observation.addComponent(percentBcmComponent);

// ECW (Extracellular Water)
Observation.ObservationComponentComponent ecwComponent = new Observation.ObservationComponentComponent();
ecwComponent.getCode().addCoding().setSystem("https://www.snomed.org/").setCode("55450009").setDisplay("Extracellular Water");
ecwComponent.setValue(new Quantity().setValue(fichero.getEcw()).setUnit("L"));
observation.addComponent(ecwComponent);

// ICW (Intracellular Water)
Observation.ObservationComponentComponent icwComponent = new Observation.ObservationComponentComponent();
icwComponent.getCode().addCoding().setSystem("https://www.snomed.org/").setCode("20125000").setDisplay("Intracellular Water");
icwComponent.setValue(new Quantity().setValue(fichero.getIcw()).setUnit("L"));
observation.addComponent(icwComponent);

// TBW (Total Body Water)
Observation.ObservationComponentComponent tbwComponent = new Observation.ObservationComponentComponent();
tbwComponent.getCode().addCoding().setSystem("https://www.snomed.org/").setCode("251837008").setDisplay("Total Body Water");
tbwComponent.setValue(new Quantity().setValue(fichero.getTbw()).setUnit("L"));
observation.addComponent(tbwComponent);

// FFM (Fat-Free Mass)
Observation.ObservationComponentComponent ffmComponent = new Observation.ObservationComponentComponent();
ffmComponent.getCode().addCoding().setSystem("https://www.snomed.org/").setCode("248363008").setDisplay("Fat-Free Mass");
ffmComponent.setValue(new Quantity().setValue(fichero.getFfm()).setUnit("kg"));
observation.addComponent(ffmComponent);
```

Figura 3-13. Parámetros de composición corporal en el recurso *Observation*

Como se puede apreciar en la Figura 3-13, varios parámetros de composición corporal tienen configurado los campos *"System"* y *"Code"* con valores reales. En este caso se ha usado **SNOMED** como sistema de terminología médica, ya que **LOINC** no dispone de algunos códigos relacionados a parámetros de composición corporal.

Aquellos parámetros de composición corporal que sí disponen de código en **SNOMED CT** [30]: *Extracellular Water* (ECW) con el código 55450009*, Intracellular Water* (ICW) con el código 20125000, *Total Body Water* (TBW) con el código 251837008, *Fat-Free Mass* (FFM) con el código 248363008, *Fat Mass* (FM) con el código 5941000237109, y *Overhydration* (Sobrehidratación) con el código 61688009. Algunos de estos parámetros de composición corporal también disponen de código en LOINC pero por unicidad se ha decicido utilizar SNOMED, ya que era el sistema que disponía de más parámetros de composición corporal.

3.3.4 Recurso *Bundle*

El recurso *Bundle* [22] se utiliza principalmente para manejar conjuntos de recursos relacionados, y en nuestro proyecto ha sido utilizado para la búsqueda de pacientes y dispositivos ya existentes o no en la base de datos de FHIR. Se realizan consultas sobre los recursos *Patient* y *Device*, asegurando que no se creen duplicados. Los campos utilizados en este recurso son:

- *"Entry"*: Contiene una lista de recursos encontrados durante la búsqueda. Este campo es útil para identificar si un paciente o dispositivo ya existe en la base de datos.
- *"Type"*: Especifica el tipo de operación realizada, como una búsqueda o creación de recursos.

En la Figura 3-14 podemos observar el uso de este recurso para este proyecto.

```
Bundle searchResultsDevices = client.search()
        .forResource(Device.class)
        .where(Device.IDENTIFIER.exactly().code(Integer.toString(fichero.getId_disp())))
        .returnBundle(Bundle.class)
        .execute();

Bundle searchResultsPatients = client.search()
       .forResource(Patient.class)
       .where(Patient.IDENTIFIER.exactly().code(Integer.toString(fichero.getId_pac())))
       .returnBundle(Bundle.class)
       .execute();
```

Figura 3-14. Uso del recurso *Bundle*

3.4 Implementación de la aplicación

En este apartado se detallará la implementación de la aplicación. Se comenzará describiendo la estructura de esta aplicación junto a los paquetes en los que se ha dividido para después entrar en el código usado para lograr su correcto funcionamiento.

3.4.1 Estructura del proyecto

El servicio está dividido en dos paquetes. El primer paquete incluye los códigos referentes a la implementación de la aplicación junto con el código que realiza la lectura de los ficheros CSV, y el código que se encarga de toda la gestión de recursos y parámetros de FHIR. El segundo paquete se utiliza para implementar el código referente a la base de datos ya que es un sistema externo que no pertenece a FHIR, es por eso la razón de la división de la aplicación en dos paquetes.

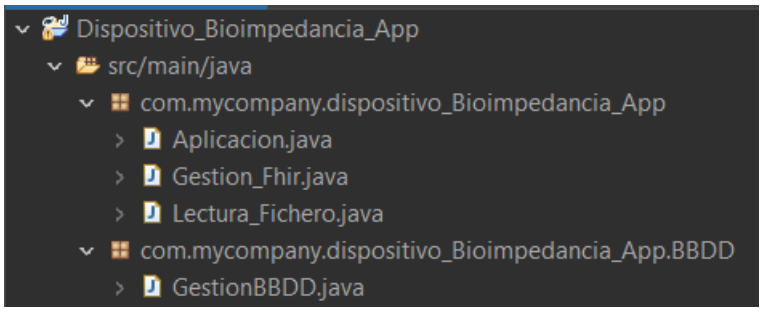

Figura 3-15. Paquetes de la aplicación

3.4.1.1 Paquete "com.mycompany.dispositivo_Bioimpedancia_App"

El paquete está formado por las siguientes clases Java:

- *"Aplicación.java"*: Este archivo contiene el punto de entrada principal de la aplicación. Aquí se inicializa el proceso de lectura del archivo CSV, la gestión de los datos obtenidos y la interacción con el sistema FHIR.
- *"Gestion_Fhir.java"*: Esta clase es responsable de la comunicación con el servidor FHIR para realizar operaciones como la creación, lectura y actualización de recursos (explicado en el apartado 3.3).
- *"Lectura_Fichero.java"*: Esta clase se encarga de la lectura y procesamiento del archivo CSV o cualquier otro formato que se utilice para la entrada de datos en la aplicación.

3.4.1.2 Paquete "com.mycompany.dispositivo_Bioimpedancia_App.BBDD"

El paquete está formado por la clase *"GestionBBDD.java"*:

- Como se explicó en el apartado 3.2, esta clase se encarga de la conexión y las operaciones de base de datos, como la inserción, actualización y consulta de registros.

3.4.2 Implementación del código

En este subapartado se describe el resto de código de la aplicación que aunque se ha mencionado no ha sido detallado anteriormente. A continuación, en la Figura 3-16 se puede observar el fichero *"build.gradle"*, fichero necesario para cargar las dependencias necesarias para poder trabajar con la especificación FHIR.

```
plugins {
    // Apply the java-library plugin for API and implementation separation.
    id 'java-library'
}

repositories {
    // Use Maven Central for resolving dependencies.
    mavenCentral()
    maven {
        url "https://oss.sonatype.org/content/repositories/snapshots"
    }
}

dependencies {
    // Use JUnit Jupiter for testing.
    testImplementation 'org.junit.jupiter:junit-jupiter:5.9.1'

    // This dependency is exported to consumers, that is to say found on their compile classpath.
    api 'org.apache.commons:commons-math3:3.6.1'

    // This dependency is used internally, and not exposed to consumers on their own compile classpath.
    implementation 'com.google.guava:guava:31.1-jre'

    implementation 'ca.uhn.hapi.fhir:hapi-fhir-base:7.0.2'
    implementation 'ca.uhn.hapi.fhir:hapi-fhir-structures-r5:7.0.2'
    implementation 'ca.uhn.hapi.fhir:hapi-fhir-client:7.0.2'

    implementation 'org.slf4j:slf4j-api:2.0.13'
    implementation 'org.slf4j:slf4j-simple:2.0.13'

    implementation files('C:/Users/Gonzalo/Desktop/UNIVERSIDAD/TFG/Base_de_Datos/sqliteadmin/sqlite-jdbc-3.8.11.2.jar')

}

// Apply a specific Java toolchain to ease working on different environments.
java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}

tasks.named('test') {
```

Figura 3-16. Fichero *"build.gradle"*

3.4.2.1 Aplicacion.java

La clase *"Aplicación.java"* actúa como el punto de entrada principal de la aplicación. En ella se lleva a cabo la inicialización del sistema y se orquesta la interacción entre las diferentes partes de la aplicación.

En la Tabla 3-1 se describe el método *"main"* usado para gestionar todo lo que es la aplicación.

Tabla 3–1 Método *"main"* de Aplicación.java

Método	*main*
Descripción	Método que se invoca cuando se ejecuta la aplicación. Aquí se inicializan las instancias de las clases "GestionBBDD", "Lectura_Fichero" y "Gestion_Fhir" para conectarnos con la base de datos, leer los datos del fichero CSV y gestionar las operaciones con el servidor FHIR.
Parámetros de entrada	Args (*String*[]) : Los posibles parámetros que se le pueden pasar por el terminal.
Respuesta	Genera la respuesta del proyecto

Este código se encarga de leer los datos del archivo utilizando el método "leer_fichero" de la clase "Lectura_Fichero.java", y posteriormente, invocamos al constructor con parámetros de la clase "Lectura_Fichero.java" pasándole todos los datos leídos del fichero CSV. Una vez tenemos los datos guardados en sus respectivas variables procedemos a guardar dichos datos en la base de datos mediante el método "insertarBBDD" de la clase "GestionBBDD.java" y configuramos los recursos FHIR a través del método "configurarRecursos" de la clase "Gestion_Fhir.java". En la Figura 3-17 podemos observar el código de la clase "Aplicación.java":

```
public class Aplicacion {

    public static String [] fields;

    public static void main(String[] args) throws IOException {

        System.out.println("¡Bienvenido a la aplicación correspondiente al dispositivo de bioimpedancia!\n");
        GestionBBDD objConexion = new GestionBBDD(); //Conectamos con la base de datos
        Lectura_Fichero lectura_fichero = new Lectura_Fichero();
        Gestion_Fhir fhir = new Gestion_Fhir();

        try {
            int id_med = 1;
            int contador = 1;
            int resultado = 1;
            fields = lectura_fichero.leer_fichero("C:/Users/Gonzalo/Desktop/UNIVERSIDAD/TFG/DatosBioimpedancia.csv", contador);

           while (fields!=null && resultado!=0) {

                System.out.println("¡Datos obtenidos correctamente de "+ contador +" paciente/s!");
                Lectura_Fichero fichero = new Lectura_Fichero(Integer.parseInt(fields[0]), Integer.parseInt(fields[1]), id_med, Integer.parseInt(fields[72]), Integer.parseInt(fields[74]), Integer.par


                resultado = objConexion.insertarBBDD(fichero);
                if (resultado == 1) {
                    System.out.println("¡Datos persistidos en la base de datos correctamente de "+ contador +" paciente/s!");
                }
                else {
                    System.out.println("¡Error en la persistencia de datos en la base de datos!");
                    resultado = 0;
                }

                if (resultado!=0) {

                    fhir.configurarRecursos(fichero);
                    System.out.println("¡Recursos Patient, Device y Observation creados y configurados!");
                    System.out.println("¡Configurado los parámetros del servidor FHIR y realizado el envío al mismo de "+ contador +" paciente/s!\n");
                    System.out.println("-----------------------------------------------------------------------------------------------------------------------------\n");

                    id_med = id_med+1;
                    contador = contador + 1;
                fields = lectura_fichero.leer_fichero("C:/Users/Gonzalo/Desktop/UNIVERSIDAD/TFG/DatosBioimpedancia.csv", contador);
            }
        }

    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Figura 3-17. Fichero "Aplicación.java"

Como podemos ver en la Figura 3-17 la clase también incluye un manejo de errores básico con la estructura *try-catch* para capturar y registrar excepciones que puedan ocurrir durante la ejecución, asegurando así la robustez de la aplicación.

3.4.2.2 Lectura_Fichero.java

La clase *"Lectura_Fichero.java"* se encarga de la lectura y procesamiento de los datos que se encuentran en un archivo de entrada. Se tienen dos métodos constructores: uno de ellos sin parámetros y otro con todos los datos existentes en el fichero CSV como parámetros. Este segundo método constructor se utiliza en la clase "Aplicacion.java" para una vez leídos asignar a las variables sus valores correspondientes de los datos leídos. Posteriormente se tienen los métodos *getter* y *setter* de cada una de las variables.

Se cuenta con un total de 92 variables, que cada una de ellas corresponde a un dato leído del fichero CSV. Por esa razón en la Figura 3-18 no se puede mostrar todo el código referente a los métodos contructores, *getters* y *setters* implementados en esta clase:

```
public class Lectura_Fichero {

    public static final String SEPARATOR=",";
    public static final String QUOTE="\"";

    private int id_pac, id_disp, id_med, r0, re, td, bcm, pctj_bcm, ffm, pctj_ffm, fm, pctj_fm;
    private int frecuencia1, frecuencia2, frecuencia3, frecuencia4, frecuencia5, frecuencia6, frecuencia7, frecuencia8, frecuencia9, frecuencia10, frecuencia11, frecuencia12, frecuencia13, frecuencia
    private int fase1, fase2, fase3, fase4, fase5, fase6, fase7, fase8, fase9, fase10, fase11, fase12, fase13, fase14, fase15, fase16, fase17, fase18, fase19, fase20, fase21, fase22;
    private float modulo1, modulo2, modulo3, modulo4, modulo5, modulo6, modulo7, modulo8, modulo9, modulo10, modulo11, modulo12, modulo13, modulo14, modulo15, modulo16, modulo17, modulo18, modulo19,
    private Date fec_nac;
    private float peso, estatura, r_inf, ri, c, alpha, icw, tbw, ffmi, fmi, sobre_hidra, ecw;
    private String sexo;

    private String [] fields;
    private String line;
    private BufferedReader br = null;

    public Lectura_Fichero(){

    }

    public Lectura_Fichero(int id_pac, int id_disp, int id_med, int r0, int re, int td, int bcm, int pctj_bcm, int ffm,
            int pctj_ffm, int fm, int pctj_fm, int frecuencia1, int frecuencia2, int frecuencia3, int frecuencia4,
            int frecuencia5, int frecuencia6, int frecuencia7, int frecuencia8, int frecuencia9, int frecuencia10,
            int frecuencia11, int frecuencia12, int frecuencia13, int frecuencia14, int frecuencia15, int frecuencia16,
            int frecuencia17, int frecuencia18, int frecuencia19, int frecuencia20, int frecuencia21, int frecuencia22,
            int fase1, int fase2, int fase3, int fase4, int fase5, int fase6, int fase7, int fase8, int fase9,
            int fase10, int fase11, int fase12, int fase13, int fase14, int fase15, int fase16, int fase17, int fase18,
            int fase19, int fase20, int fase21, int fase22, float modulo1, float modulo2, float modulo3, float modulo4,
            float modulo5, float modulo6, float modulo7, float modulo8, float modulo9, float modulo10, float modulo11,
            float modulo12, float modulo13, float modulo14, float modulo15, float modulo16, float modulo17,
            float modulo18, float modulo19, float modulo20, float modulo21, float modulo22, Date fec_nac, float peso,
            float estatura, float r_inf, float ri, float c, float alpha, float icw, float tbw, float ffmi, float fmi,
            float sobre_hidra, float ecw, String sexo) {
        super();
        this.id_pac = id_pac;
        this.id_disp = id_disp;
        this.id_med = id_med;
        this.r0 = r0;
        this.re = re;
        this.td = td;
        this.bcm = bcm;
```

Figura 3-18. Métodos constructores de la clase "Lectura_Fichero.java"

A continuación, se muestran los métodos restantes de esta clase. En la Tabla 3-2 explicamos el método "leer_fichero".

Tabla 3–2 Método "leer_fichero" de Lectura_Fichero.java

Método	leer_fichero
Descripción	Método que se invoca cuando se ejecuta la aplicación. Aquí se inicializan las instancias de las clases "GestionBBDD", "Lectura_Fichero" y "Gestion_Fhir" para conectarnos con la base de datos, leer los datos del fichero CSV y gestionar las operaciones con el servidor FHIR.
Parámetros de entrada	fichero (*String*) : Ruta del almacenamiento del fichero CSV contador (*int*) : Contador del número de líneas leídas del fichero
Respuesta	*fields* (*String* []) : Lista con todos los valores leídos del fichero *null*: En caso de un error o fin de fichero

Este método utiliza un *"BufferedReader"* para leer el archivo línea por línea que es inicializado sólo la primera vez que se invoca el método. El contador permite leer la primera vez que se invoca al método el encabezado que dispone el fichero CSV y posteriormente lee las demás líneas del fichero sin tener que volver a leer el encabezado. Tras esto, se comienza a rellenar la lista *"fields"* con ayuda del método "*split*(SEPARADOR)" que separa los campos de la lista según el separador que le pongamos como referencia, y después con el método *"removeTrailingQuotes"* eliminamos de los campos de la lista las comillas dobles. Finalmente se retorna la lista completa con los datos leídos del fichero.

En la Figura 3-19 se puede observar el código perteneciente a este método.

```
public String[] leer_fichero(String fichero, int contador) throws IOException {
    try {
        // Inicializamos el BufferedReader solo la primera vez
        if (br == null) {
            br = new BufferedReader(new FileReader(fichero));
        }

        // Leemos la primera línea (encabezado) solo una vez
        if (contador == 1) {
            line = br.readLine(); // Encabezado
        }

        // Leemos la siguiente línea de datos
        line = br.readLine();

        // Si no hay más líneas, cerramos el BufferedReader y retornamos null
        if (line == null) {
            br.close();
            return null;
        }

        // Procesamos la línea leída
        fields = line.split(SEPARATOR);
        fields = removeTrailingQuotes(fields); // Eliminamos las comillas dobles
        return fields;

    } catch (Exception e) {
        System.out.println(e);
        return null;
    }
}
```

Figura 3-19. Método "leer_fichero" de la clase "Lectura_Fichero.java"

Como se puede comprobar en la Figura 3-19 la clase también incluye un manejo de errores básico con la estructura *try-catch* para capturar y registrar excepciones que puedan ocurrir durante la ejecución.

Y por último, se describe el método *"removeTrailingQuotes"* en la Tabla 3-3.

Tabla 3–3 Método *"removeTrailingQuotes"* de Lectura_Fichero.java

Método	*removeTrailingQuotes*
Descripción	Método que se invoca a través del método "leer_fichero". Este método se encarga de eliminar las comillas dobles de cada uno de los campos leídos de la lista de datos procesados en "leer_fichero".
Parámetros de entrada	*fields* (*String* []) : Lista con todos los valores leídos del fichero (con comillas dobles)
Respuesta	*fields* (*String* []) : Lista con todos los valores leídos del fichero (sin comillas dobles)

El método *"removeTrailingQuotes"* tiene como propósito eliminar las comillas (tanto simples como dobles) que pueden estar al principio o al final de las cadenas en un *array* de *strings*, y para ello se utiliza el método *"replaceAll"*. Con la expresión "^" indicamos el inicio de la cadena, por lo que "^" + "*QUOTE* (cuyo valor es \")" busca eliminar una comilla si se encuentra al principio de la cadena. Por otro lado, la expresión "$" indica el final de la cadena, por lo que "*QUOTE* + "$"" busca eliminar las comillas si se encuentran al final de la cadena.

En la Figura 3-20 se puede observar el código perteneciente a este método.

```
private static String[] removeTrailingQuotes(String[] fields) {
    String result[] = new String[fields.length];

    for (int i=0;i<result.length;i++){
       result[i] = fields[i].replaceAll("^"+QUOTE, "").replaceAll(QUOTE+"$", "");
    }
    return result;
}
```

Figura 3-20. Método *"removeTrailingQuotes"* de la clase "Lectura_Fichero.java"

3.4.2.3 Gestion_Fhir.java

Esta clase desempeña un papel crucial en la integración y manipulación de los datos de bioimpedancia con el estándar FHIR. Utiliza la API HAPI FHIR para comunicarse a través de REST con un servidor FHIR público, que también nos proporciona HAPI FHIR, y gestionar los recursos correspondientes a los pacientes, dispositivos y observaciones médicas.

En primer lugar, la clase *"Gestion_Fhir.java"* comienza con la inicialización de un contexto FHIR utilizando la versión R5 del estándar y la configuración de un cliente genérico que se conecta al servidor FHIR público de la API (http://hapi.fhir.org/baseR5) (Figura 3-21). Esto permite interactuar con el servidor para crear, buscar y manipular recursos FHIR.

```
public class Gestion_Fhir {


    FhirContext ctx = FhirContext.forR5();
    String serverBase = "http://hapi.fhir.org/baseR5";
    IGenericClient client = ctx.newRestfulGenericClient(serverBase);
```

Figura 3-21. Código referente a la conexión con el servidor FHIR

Continuando con el resto de código, se procede a explicar el método “configurarRecursos()” (Tabla 3-4).

Tabla 3–4 Método “configurarRecursos”de Gestion_Fhir.java

Método	configurarRecursos
Descripción	Método responsable de la creación o recuperación de los recursos FHIR asociados a un paciente y un dispositivo, así como de la creación de una observación médica basada en los datos de bioimpedancia.
Parámetros de entrada	fichero (Lectura_Fichero) : Instancia de la clase “Lectura_Fichero.java” que contiene todos los datos de bioimpedancia leídos y almacenados.
Respuesta	*void*

El método “configurarRecursos” comienza con la búsqueda de un recurso *Patient* en el servidor FHIR utilizando el identificador del paciente leído del fichero CSV. Si no existe, se crea un nuevo recurso *Patient* con el identificador, género y fecha de nacimietno proporcionados por el fichero, se envía al servidor FHIR y obtenemos su referencia. Si el paciente ya existe, se recupera su referencia directamente (Figura 3-22).

```
 Bundle searchResultsPatients = client.search()
       .forResource(Patient.class)
       .where(Patient.IDENTIFIER.exactly().code(Integer.toString(fichero.getId_pac())))
       .returnBundle(Bundle.class)
       .execute();

 List<Patient> patients = BundleUtil.toListOfResourcesOfType(ctx, searchResultsPatients, Patient.class);
 String patientReference = null;

if (patients.isEmpty()) {
    Patient patient = new Patient();

    // Configura el identificador del paciente
    Identifier identifier = new Identifier();
    identifier.setValue(Integer.toString(fichero.getId_pac())); // Identificador único
    patient.addIdentifier(identifier);


    // Configura el género (sexo)
    if ("M".equals(fichero.getSexo())) {
        patient.setGender(Enumerations.AdministrativeGender.MALE);
    } else if ("F".equals(fichero.getSexo())) {
        patient.setGender(Enumerations.AdministrativeGender.FEMALE);
    } else {
        patient.setGender(Enumerations.AdministrativeGender.UNKNOWN);
    }

    // Configura la fecha de nacimiento
    patient.setBirthDate(fichero.getFec_nac());

    //Enviamos el recurso Patient y si todo va bien obtenemos su referencia
    MethodOutcome outcome = client.create().resource(patient).prettyPrint().encodedJson().execute();

    // This will return Boolean.TRUE if the server responded with an HTTP 201 created, otherwise it will return null.
    Boolean created = outcome.getCreated();

     if (created) {
      System.out.println("El servidor ha creado el recurso Patient");
      patientReference = outcome.getResource().getIdElement().getValue();
      }
} else {
     System.out.println("El recurso Patient ya existe!");
     patientReference = patients.get(0).getIdElement().getValue();
}
```

Figura 3-22. Creación o recuperación del recurso *Patient*

Posteriormente, se realiza el mismo proceso para el recurso *Device*. Debido a que el proceso que se sigue con el recurso *Device* es el mismo que se usa para el recurso *Patient* vamos a omitir explicar el proceso en detalle de la creación o recuperación del recurso *Device* y vamos a continuar con la explicación del resto de código perteneciente al método "configurarRecursos".

Una vez tenemos las referencias de los recursos *Patient* y *Device*, se crea el recurso *Observation*. Este recurso registra las mediciones de bioimpedancia, incluyendo la estatura y otros parámetros específicos como BCW, ECW, ICW, entre otros. Se configura el estado del recurso *Observation*, la fecha y hora en la que se realiza la observación médica, y una vez los parámetros de composición corporal están configurados (a través del método "configurarObservation()") se procede a vincular las referencias de los recursos *Patient* y *Device* al recurso *Observation*. Por último se envía el recurso al servidor FHIR y se comprueba si el envío se ejecutó de forma correcta (Figura 3-23).

```
        Observation observation = new Observation();

         // Configura el estado del recurso observacion
        observation.setStatus(org.hl7.fhir.r5.model.Enumerations.ObservationStatus.FINAL);

        // Configura el código para la medición de estatura
        observation.getCode().addCoding().setSystem("http://loinc.org").setCode("8302-2").setDisplay("Body height");
        // Configura el valor de la estatura (por ejemplo, en centímetros)
        observation.setValue(new Quantity().setValue(fichero.getEstatura()).setUnit("cm"));

          // Configura la fecha y hora de la observación
         observation.setEffective(new DateTimeType(new Date()));

        //Configuramos los parámetros de composicion corporal en un metodo aparte
        configurarObservation(observation, fichero);

        // Configura la referencia al paciente y dispositivo
        observation.setSubject(new Reference(patientReference)); // Referencia al paciente
        observation.setDevice(new Reference(deviceReference)); // Referencia al dispositivo

        //Enviamos el recurso Observation
        MethodOutcome outcome = client.create().resource(observation).prettyPrint().encodedJson().execute();

        // This will return Boolean.TRUE if the server responded with an HTTP 201 created, otherwise it will return null.
        Boolean created = outcome.getCreated();

        if (created) {
         System.out.println("El servidor ha creado el recurso Observation");
         }

    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Figura 3-23. Creación del recurso *Observation*

Por último, en la Tabla 3-5 se detalla el método "configurarObservation()" de la clase *"Gestion_Fhir.java"*.

Tabla 3–5 Método "configurarObservation"de Gestion_Fhir.java

Método	configurarObservation
Descripción	Método que se encarga de configurar los componentes específicos del recurso *Observation* con las distintas mediciones de bioimpedancia.
Parámetros de entrada	*observation* (*Observation*) : Recurso *observation* creado en el método "configurarRecursos". fichero (Lectura_Fichero) : Instancia de la clase "Lectura_Fichero.java" que contiene todos los datos de bioimpedancia leídos y almacenados.
Respuesta	*void*

Para cada tipo de medición (por ejemplo, BCW, ECW, ICW, etc.), se añade un componente al recurso *Observation* que incluye el código del sistema de terminología médica (como SNOMED o LOINC), la unidad de medida y el valor correspondiente (Figura 3-13).

3.4.3 Resumen del proceso de la aplicación

A continuación, se presenta un resumen de todo el proceso por el que pasa la aplicación hasta llegar al resultado final. Además, el proceso está representado en el diagrama de flujo UML [29] de la Figura 3-24.

1. Se inicia "Aplicación.java" y establece la conexión con la base de datos *SQLite* a través del constructor de la clase "GestionBBDD.java". En caso de que se establezca la conexión correctamente aparecería un mensaje por el terminal confirmando la conexión establecida, en caso contrario nos mostraría un mensaje de error de conexión. La aplicación seguiría funcionando hasta insertar los datos en la base de datos, lo cual daría error y la aplicación terminaría.
2. Se declaran las variables correspondientes a las clases "Lectura_Fichero.java" y "Gestion_Fhir.java", y mediante el método "leer_fichero()" de la clase "Lectura_Fichero.java" se comienza a leer el archivo CSV por la primera línea. En caso de error o fin de fichero, este método devolvería *null* y la aplicación terminaría. En caso contrario, se procesaría la primera línea y se seguiría con el programa.
3. Todas los campos leídos se guardarían en variables de la clase "Lectura_Fichero.java" y se insertarían en la base de datos mediante el método "insertarBBDD()" de la clase "GestionBBDD.java". En caso de existir un error en la inserción de datos en la base de datos se imprimiría por terminal un error informando de ello y la aplicación terminaría.
4. El siguiente paso es configurar los recursos FHIR mediante el método "configurarRecursos()" de la clase "Gestion_Fhir.java". En primer lugar, se comprueba si en la base de datos de FHIR existe un paciente con identificador de paciente igual al leído en el fichero CSV. Si existe se guarda la referencia del paciente ya existente, y si no existe se crea el recurso *Patient* configurando todos sus parámetros necesarios y se guarda la referencia del recurso que se acaba de crear. Posteriormente, se hace lo mismo para el recurso *Device* y se guarda su referencia correspondiente, pasando a configurar los campos necesarios del recurso *Observation.* Si todo ha ido correctamente, se imprime por pantalla que todo ha sido configurado correctamente.
5. Por último, se vuelve a leer una nueva línea del fichero CSV. En caso de no existir ninguna línea más en el fichero, devolveríamos *null*, saldría del bucle y terminaríamos la aplicación.

A continuación, en la Figura 3-24 se puede observar el diagrama de flujo UML que representa gráficamente lo explicado anteriormente.

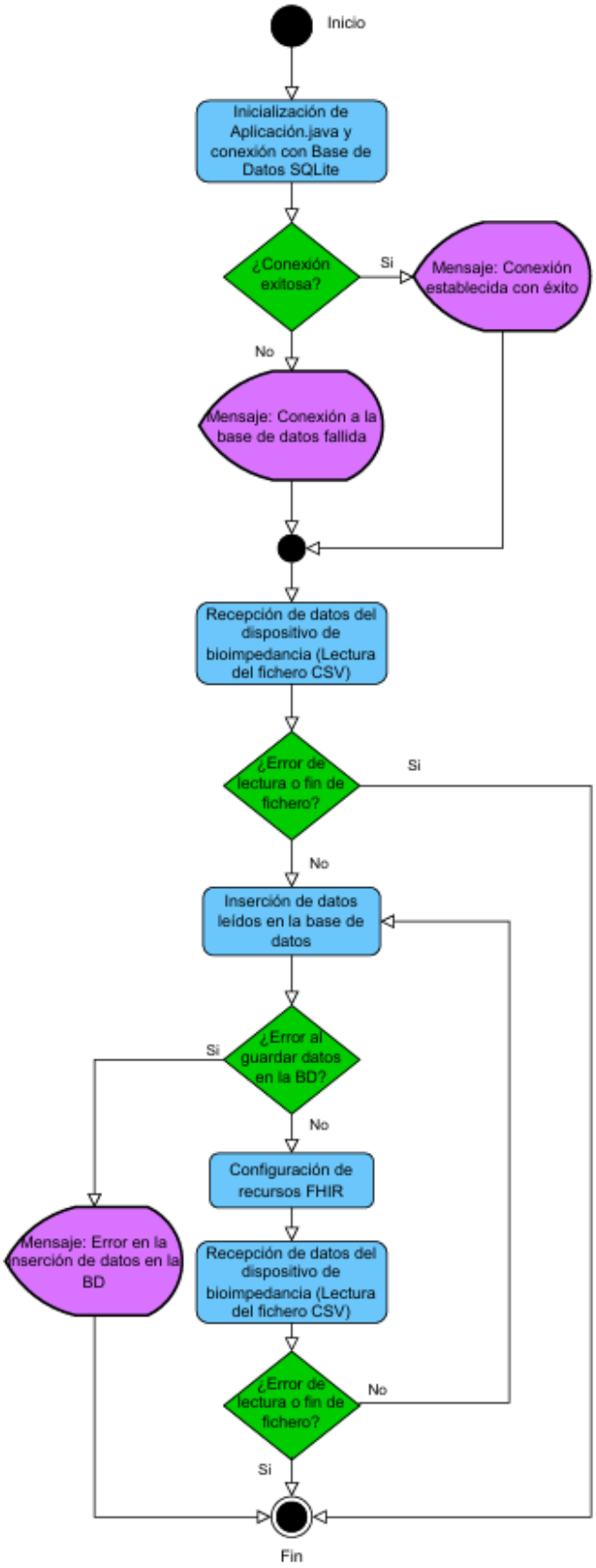

Figura 3-24. Diagrama de flujo UML representando el proceso de la aplicación

3.5 Pruebas

A continuación, se muestra el funcionamiento de la aplicación realizando una prueba con datos pertenecientes a tres pacientes.

3.5.1 Comprobación de la gestión de datos

En la Figura 3-25 se pueden observar los datos de tres pacientes distintos, que se reciben por parte de los dispositivos de bioimpedancia, pero que en este caso se simulan mediante la lectura de datos del fichero CSV.

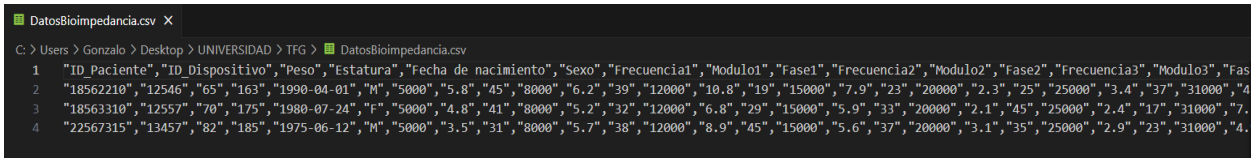

Figura 3-25. Datos de pacientes en el fichero CSV

En la siguiente figura, se puede comprobar cómo antes de la ejecución de la aplicación todas las tablas creadas en la base de datos se encuentran vacías.

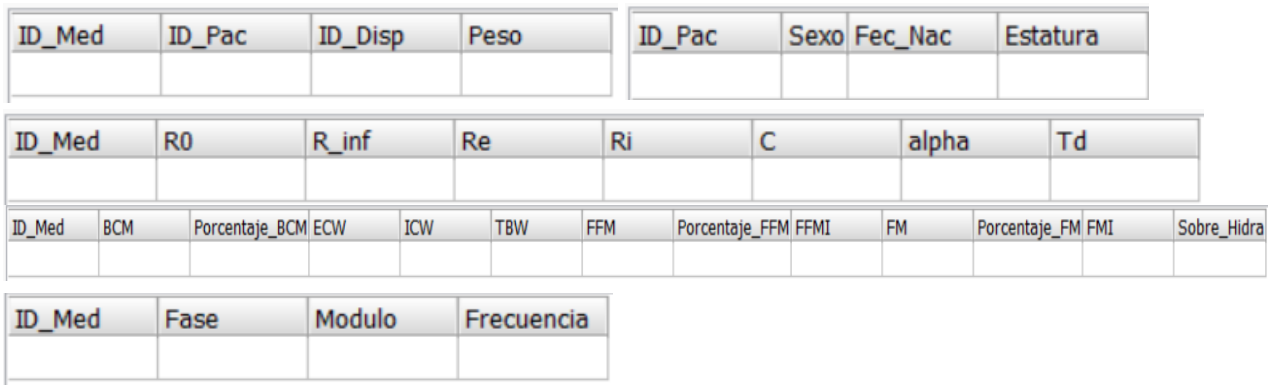

Figura 3-26. Base de datos vacía

3.5.2 Ejecución

Una vez todo preparado para la ejecución, se lanza la aplicación y durante el transcurso de la misma se puede monitorizar su ejecución mediante los mensajes que se imprimen en la terminal. La aplicación finalizará cuando la recepción de datos haya llegado a su fin, es decir, cuando el lector de ficheros llegue al final del fichero CSV. En la Figura 3-27, se puede ver un ejemplo de la salida por pantalla una vez se ha ejecutado la aplicación.

```
¡Bienvenido a la aplicación correspondiente al dispositivo de bioimpedancia!

Conexión a la base de datos establecida con éxito.
[main] INFO ca.uhn.fhir.util.VersionUtil - HAPI FHIR version 7.0.2 - Rev 95beaec894
[main] INFO ca.uhn.fhir.context.FhirContext - Creating new FHIR context for FHIR version [R5]
[main] INFO ca.uhn.fhir.util.XmlUtil - Unable to determine StAX implementation: java.xml/META-INF/MANIFEST.MF not found
¡Datos obtenidos correctamente de 1 paciente/s!
La inserción de datos en la base de datos fue realizada con éxito.
¡Datos persistidos en la base de datos correctamente de 1 paciente/s!
El recurso Patient ya existe!
El recurso Device ya existe!
El servidor ha creado el recurso Observation
¡Recursos Patient, Device y Observation creados y configurados!
¡Configurado los parámetros del servidor FHIR y realizado el envío al mismo de 1 paciente/s!

-----------------------------------------------------------------------------------------------------------------------

¡Datos obtenidos correctamente de 2 paciente/s!
La inserción de datos en la base de datos fue realizada con éxito.
¡Datos persistidos en la base de datos correctamente de 2 paciente/s!
El recurso Patient ya existe!
El recurso Device ya existe!
El servidor ha creado el recurso Observation
¡Recursos Patient, Device y Observation creados y configurados!
¡Configurado los parámetros del servidor FHIR y realizado el envío al mismo de 2 paciente/s!

-----------------------------------------------------------------------------------------------------------------------

¡Datos obtenidos correctamente de 3 paciente/s!
La inserción de datos en la base de datos fue realizada con éxito.
¡Datos persistidos en la base de datos correctamente de 3 paciente/s!
El recurso Patient ya existe!
El recurso Device ya existe!
El servidor ha creado el recurso Observation
¡Recursos Patient, Device y Observation creados y configurados!
¡Configurado los parámetros del servidor FHIR y realizado el envío al mismo de 3 paciente/s!
```

Figura 3-27. Ejemplo de ejecución

La razón por la que en pantalla se imprime que los recursos *Patient* y *Device* de los tres pacientes ya existen es debido a las ejecuciones realizadas anteriormente para comprobar el correcto funcionamiento de la aplicación, por lo que esto indica que ya existen las referencias de los pacientes y dispositivos en la base de datos FHIR con sus correspondientes identificadores.

3.5.3 Comprobación del correcto almacenamiento en la base de datos

En la Figura 3-28, se puede comprobar que se han persistido correctamente los datos recibidos por los dispositivos de bioimpedancia (sólo se muestran algunas tablas de la base de datos, debido a la larga extensión de alguna de ellas).

ID_Med	ID_Pac	ID_Disp	Peso
1	18562210	12546	65
2	18563310	12557	70
3	22567315	13457	82

ID_Pac	Sexo	Fec_Nac	Estatura
18562210	M	01/04/1990	163
18563310	F	24/07/1980	175
22567315	M	12/06/1975	185

ID_Med	R0	R_inf	Re	Ri	C	alpha	Td
1	1500	900	1500	2250	1,1	0,1	5
2	1300	700	1300	2050	2,1	0,7	3
3	1250	820	1250	2125	3,2	0,4	7

ID_Med	BCM	Porcentaje_BCM	ECW	ICW	TBW	FFM	Porcentaje_FFM	FFMI	FM	Porcentaje_FM	FMI	Sobre_Hidra
1	39	26	29,25	53,18182	82,431816	52	80	19,571682	13	20	4,8929205	0,5
2	49	23	27,35	43,191917	80,45182	55	70	29,571682	10	19	5,79292	0,7
3	52	18	21,25	57,121212	78,67182	49	67	24,681982	15	16	3,8939204	0,4

Figura 3-28. Base de datos completa

3.5.4 Comprobación del correcto almacenamiento en el servidor FHIR

A continuación se va a mostrar la comprobación de que los datos de un paciente a través de los recursos se almacenan correctamente en el servidor FHIR.

En la Figura 3-29 se puede observar el resultado de la primera ejecución de la aplicación con un paciente cuyo ID de paciente no existe aún en la base de datos FHIR.

```
¡Bienvenido a la aplicación correspondiente al dispositivo de bioimpedancia!

Conexión a la base de datos establecida con éxito.
[main] INFO ca.uhn.fhir.util.VersionUtil - HAPI FHIR version 7.0.2 - Rev 95beaec894
[main] INFO ca.uhn.fhir.context.FhirContext - Creating new FHIR context for FHIR version [R5]
[main] INFO ca.uhn.fhir.util.XmlUtil - Unable to determine StAX implementation: java.xml/META-INF/MANIFEST.MF not fou
¡Datos obtenidos correctamente de 1 paciente/s!
La inserción de datos en la base de datos fue realizada con éxito.
¡Datos persistidos en la base de datos correctamente de 1 paciente/s!
Recurso Patient creado: https://hapi.fhir.org/baseR5/Patient/766042/_history/1
El servidor ha creado el recurso Patient
Recurso Device creado: https://hapi.fhir.org/baseR5/Device/766043/_history/1
El servidor ha creado el recurso Device
Recurso Observation creado: https://hapi.fhir.org/baseR5/Observation/766044/_history/1
El servidor ha creado el recurso Observation
¡Recursos Patient, Device y Observation creados y configurados!
¡Configurado los parámetros del servidor FHIR y realizado el envío al mismo de 1 paciente/s!
```

Figura 3-29. Primera ejecución de la aplicación

Al no existir el recurso *Patient* en la base de datos de FHIR, lo crea, y así sucede también con el recurso *Device*. Si se introduce el enlace que nos proporciona la salida de la aplicación, se puede corroborar que los datos han sido almacenados correctamente en el servidor FHIR (Figura 3-30, Figura 3-31, Figura 3-32).

Response Body


```
{
  "resourceType": "Patient",
  "id": "766042",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2024-09-08T15:16:30.640+00:00",
    "source": "#GP4K0vy5f7RGXPms"
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml"
  },
  "identifier": [ {
    "value": "18565312"
  } ],
  "gender": "male",
  "birthDate": "1990-04-01"
}
```


Figura 3-30. Instancia del recurso *Patient*

Response Body


```
{
  "resourceType": "Device",
  "id": "766043",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2024-09-08T15:16:32.641+00:00",
    "source": "#Av1R6iDCoa0g532w"
  },
  "identifier": [ {
    "value": "12948"
  } ],
  "status": "active"
}
```


Figura 3-31. Instancia del recurso *Device*

Response Body


```
{
  "resourceType": "Observation",
  "id": "766068",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2024-09-09T13:28:42.611+00:00",
    "source": "#MT83256cmabCINFR"
  },
  "status": "final",
  "code": {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "8302-2",
      "display": "Body height"
    } ]
  },
  "subject": {
    "reference": "Patient/766042"
  },
  "effectiveDateTime": "2024-09-09T15:28:41+02:00",
  "valueQuantity": {
    "value": 163.0,
    "unit": "cm"
  },
  "device": {
    "reference": "Device/766043"
  },
  "component": [ {
    "code": {
      "coding": [ {
        "system": "http://example.com",
        "code": "BCM",
        "display": "Body Cell Mass"
      } ]
    },
    "valueQuantity": {
      "value": 39,
      "unit": "kg"
    }
```


Figura 3-32. Instancia del recurso *Observation*

Además, se puede comprobar en la interfaz gráfica que nos ofrece HAPI FHIR del servidor FHIR cómo las instancias de los recursos *Patient*, *Device* y *Observation* se han guardado en dicho servidor (Figura 3-33, Figura 3-34, Figura 3-35).

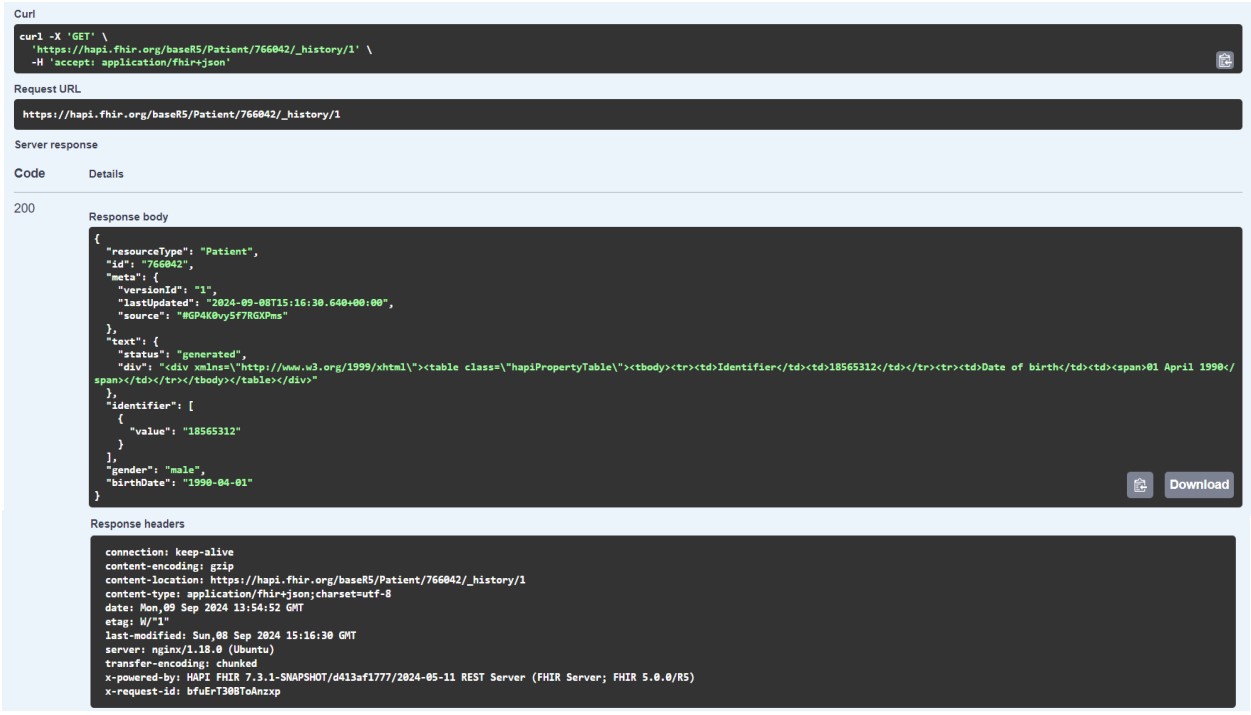

Figura 3-33. Respuesta del servidor FHIR del recurso *Patient*

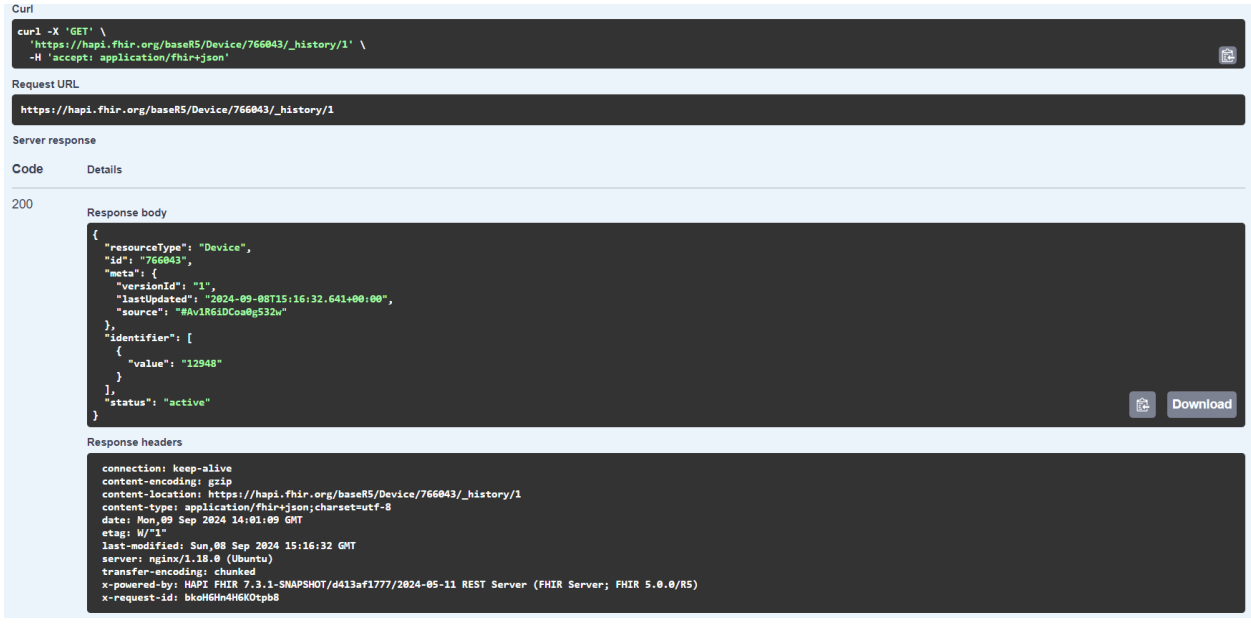

Figura 3-34. Respuesta del servidor FHIR del recurso *Device*

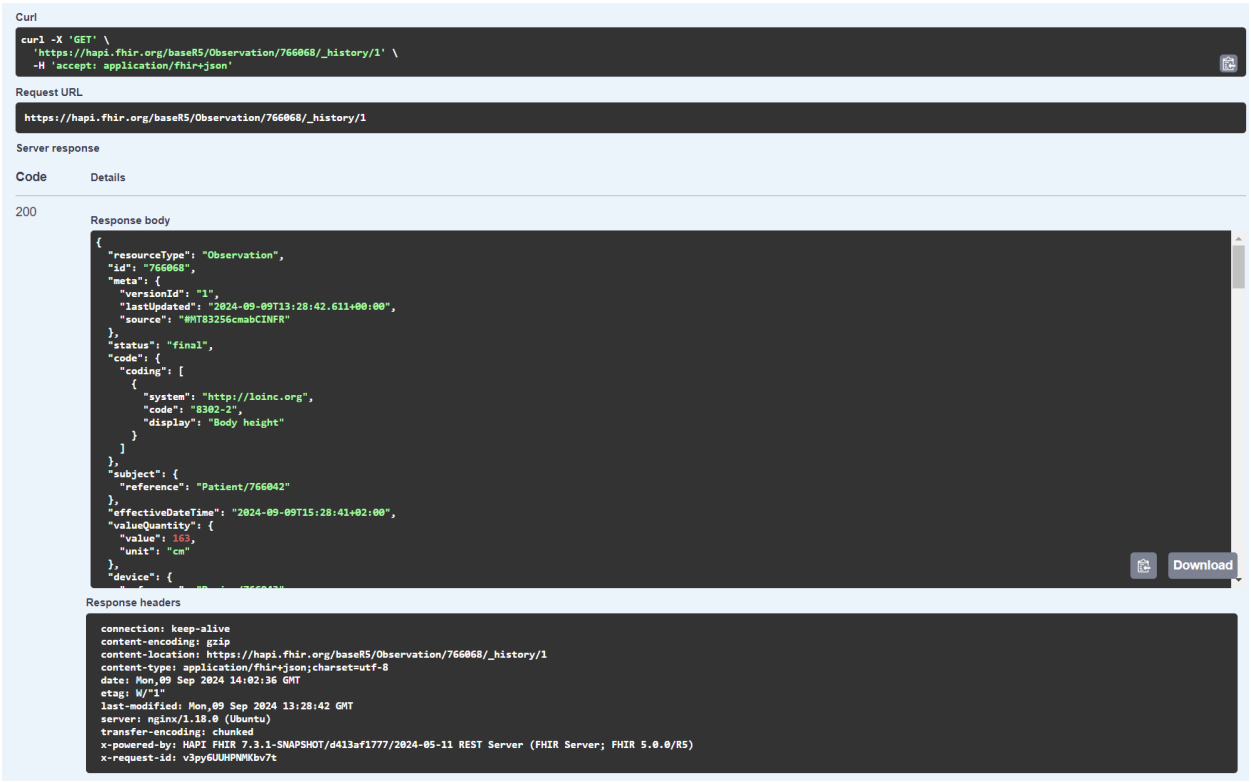

Figura 3-35. Respuesta del servidor FHIR del recurso *Observation*

Por último, si se vuelve a realizar una nueva ejecución con los mismos datos de los pacientes y dispositivos de la anterior ejecución, estos deberían de estar ya almacenados en el servidor FHIR, por tanto no debería crear nuevos recursos según el código implementado en la aplicación. En la Figura 3-36, se pude ver esta confirmación.

```
¡Bienvenido a la aplicación correspondiente al dispositivo de bioimpedancia!

Conexión a la base de datos establecida con éxito.
[main] INFO ca.uhn.fhir.util.VersionUtil - HAPI FHIR version 7.0.2 - Rev 95beaec894
[main] INFO ca.uhn.fhir.context.FhirContext - Creating new FHIR context for FHIR version [R5]
[main] INFO ca.uhn.fhir.util.XmlUtil - Unable to determine StAX implementation: java.xml/META-INF/MANIFEST.MF not found
¡Datos obtenidos correctamente de 1 paciente/s!
La inserción de datos en la base de datos fue realizada con éxito.
¡Datos persistidos en la base de datos correctamente de 1 paciente/s!
El recurso Patient ya existe!
El ID es: https://hapi.fhir.org/baseR5/Patient/766042/_history/1
El recurso Device ya existe!
El ID es: https://hapi.fhir.org/baseR5/Device/766043/_history/1
Recurso Observation creado: https://hapi.fhir.org/baseR5/Observation/766068/_history/1
El servidor ha creado el recurso Observation
¡Recursos Patient, Device y Observation creados y configurados!
¡Configurado los parámetros del servidor FHIR y realizado el envío al mismo de 1 paciente/s!
```

Figura 3-36. Segunda ejecución de la aplicación

4 CONCLUSIONES Y LÍNEAS FUTURAS

"Cada final es un nuevo comienzo."

- TS Eliot -

En este último capítulo vamos a mostrar las conclusiones que hemos extraído del desarrollo de este proyecto, los resultados obtenidos tras los objetivos que nos hemos ido marcando, y propondremos una serie de líneas de trabajo futuras que se pudieran desarrollar a partir de este trabajo realizado.

4.1 Conclusiones

El desarrollo de este Trabajo Fin de Grado nos ha permitido explorar la estandarización FHIR, así como su integración en el ámbito sanitario y tecnológico, en nuestro caso con datos obtenidos a partir de mediciones a pacientes para obtener sus parámetros de bioimpedancia. A lo largo de este proyecto, hemos ido consiguiendo el cumplimiento de los objetivos planteados dividiendo el objetivo principal (desarrollar una aplicación intermediaria entre dispositivos de bioimpedancia y organizaciones sanitarias), en distintos subobjetivos, los cuales nos han tomado su tiempo ya que queríamos asegurarnos de que se cumplieran de forma correcta. Finalmente, hemos conseguido desarrollar una aplicación capaz de procesar los datos de bioimpedancia, persistirlos en una base de datos, y posteriormente mapearlos al formato FHIR para poder ser enviados y gestionados por los sistemas de información sanitaria.

Al comienzo, nos decantamos por la utilización de un IDE como *Apache NetBeans*, pero este entorno de desarrollo era difícil de manejar a la hora de organizar el proyecto en paquetes, actualización de dependencias (ya sean mediante *Maven* o *Gradle*), importación de librerías FHIR, etc. Por tanto, decidimos usar como entorno de desarrollo *Eclipse IDE* para el desarrollo de la aplicación y *SQLite* para la gestión de la base de datos. Estas herramientas han demostrado ser adecuadas para la complejidad del proyecto, proporcionando un entorno de desarrollo fácil de manejar. No obstante, también hemos identificado ciertos desafíos, como la complejidad de la manipulación de datos de bioimpedancia y la necesidad de comprender en profundidad el estándar FHIR y su API (*HAPI FHIR*) para un mapeo y uso de los datos adecuado. Este estándar, para aquellas personas que nunca lo hayan manipulado o investigado es complejo al inicio, debido a la poca documentación que existe para lograr entender el manejo de recursos FHIR.

A pesar de estos desafíos, podemos concluir que las soluciones implementadas han sido satisfactorias en su mayoría. La estructura de la aplicación, junto con los métodos y clases desarrollados, han demostrado ser eficaces para el procesamiento y almacenamiento de datos. Sin embargo, a pesar de haber intentado ser lo más eficientes posible en este aspecto, algunos procesos podrías haberse optimizado más, en especial en lo que respecta a la eficiencia en la gestión de los datos de bioimpedancia.

4.2 Líneas futuras

El proyecto es solo el comienzo de lo que podría ser una solución más amplia y completa para la integración de datos de bioimpedancia en sistemas sanitarios. Dada la naturaleza limitada de este trabajo y el cumplimiento de los objetivos propuestos al comienzo del mismo, consideramos que hay varias líneas de trabajo futuras que podrían desarrollarse a partir de esta base:

- **Implementación de un servicio de escucha de datos en tiempo real:** Una de las primeras extensiones que podrían realizarse es la implementación de un servicio que esté constantemente a la escucha de peticiones para recibir los datos de los pacientes en tiempo real, directamente desde el dispositivo de bioimpedancia a través de la tecnología Bluetooth que dispone el mismo. Esto permitiría una mayor automatización del proceso, eliminando la necesidad de intervenciones manuales y facilitando el flujo de datos en tiempo real hacia la organización sanitaria.

- **Incluir mecanismos de seguridad en el sistema:** Actualmente, la aplicación utiliza el servidor público de FHIR para la transmisión de datos. Una línea futura crucial sería el desarrollo e implementación de un servidor propio de FHIR (privado), que permita controlar mejor la seguridad y la privacidad de los datos transmitidos. Esto incluiría la implementación de cifrado en las comunicaciones y autenticación segura de usuarios, garantizando que los datos de los pacientes se transmiten de forma segura y cumpliendo con las normativas de protección de datos. Los mecanismos que propone el estándar FHIR son:
    - Intercambio de datos cifrado mediante TLS/SSL.
    - Autenticación de usuarios así como control de autorización/acceso de los usuarios.
    - Uso de firmas digitales.

- **Optimización del mapeo de datos a FHIR:** Aunque el mapeo actual de los datos a recursos FHIR ha sido efectivo, existe margen para mejorar la precisión y eficiencia de este proceso. Esto podría implicar una revisión más exhaustiva del estándar FHIR para identificar otros recursos y perfiles que podrían utilizarse para representar mejor los datos de bioimpedancia, así como la automatización del mapeo para reducir posibles errores.

REFERENCIAS

[1] «Grupo de Ingeniería Biomédica de la Universidad de Sevilla,» [En línea]. Disponible en: https://investigacion.us.es/sisius/grupo/TIC203

[2] «HL7 FHIR,» [En línea]. Disponible en: https://hl7.org/fhir/

[3] Msc. Lidyce Quesada Leyva, «Elementos teóricos y prácticos sobre la bioimpedancia eléctrica en salud,» AMC [Internet], vol.20, n.5, pp. 565-578, sep.-oct. 2016. Disponible en: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1025-02552016000500014&lng=es

[4] «¿Qué es la bioimpedancia?,» [En línea]. Disponible en: https://movidasana.com/bioimpedancia-que-es-para-que-sirve/

[5] L. M. Roa, D. Naranjo y J. Reina-Tosina, «*Applications of bioimpedance to end stage renal disease (ESRD),» Studies in Computational Intelligence*, vol.404, pp. 689-769, 2013.

[6] A. C. Buchholz, C. Bartok y D. A. Schoeller, *«The validity of bioelectrical impedance models in clinical populations,» Nutrition in Clinical Practice*, vol.19, n.5, pp. 433-446, 2004.

[7] J. Castizo-Olier, A. Irurtia, M. Jemni, M. Carrasco-Marginet, R. Fernández-García y F. A. Rodríguez, «*Bioelectrical impedance vector analysis (BIVA) in sport and exercise: Systematic review and future perspectives,»* PLoS ONE, vol.13, n.6, 2018.

[8] B. H. Cornish, B. J. Thomas y L. C. Ward, *«Improved prediction of extracellular and total body water using impedance loci generated by multiple frequency bioelectrical impedance analysis,»* Phys. Med. Biol., vol.38, n.3, pp. 337-346, 1993.

[9] L. Muñoz Fernández y S. Gallego Riestra, «La Protección de Datos y la interoperabilidad en el ámbito sanitario: dos realidades inseparables,» Segunda ed., vol.29, 2019.

[10] «HL7 International,» HL7 Version 2 Product Suite, [En línea]. Disponible en: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=185

[11] «HL7 International,» HL7 Version 3 Product Suite, [En línea]. Disponible en: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=186

[12] «HL7 International,» CDA® Release 2, [En línea]. Disponible en: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=7

[13] «HL7 FHIR. RESTful API,» [En línea]. Disponible en: https://www.hl7.org/fhir/http.html

[14] «HL7 FHIR. Security,» [En línea]. Disponible en: https://www.hl7.org/fhir/security.html

[15] «HL7 FHIR. Resource Index,» [En línea]. Disponible en: https://hl7.org/fhir/resourcelist.html

[16] «HL7 FHIR. Data Types,» [En línea]. Disponible en: https://www.hl7.org/fhir/datatypes.html

[17] «HL7 FHIR. Patient,» [En línea]. Disponible en: https://hl7.org/fhir/patient.html

[18] «HL7 FHIR. Observation,» [En línea]. Disponible en: https://hl7.org/fhir/observation.html

[19] «LOINC,» [En línea]. Disponible en: https://loinc.org/

[20] «SNOMED,» [En línea]. Disponible en: https://www.snomed.org/

[21] «HL7 FHIR. Device,» [En línea]. Disponible en: https://hl7.org/fhir/device.html

[22] «HL7 FHIR. Bundle,» [En línea]. Disponible en: https://hl7.org/fhir/bundle.html

[23] «HAPI FHIR,» [En línea]. Disponible en: https://hapifhir.io/

[24] «HAPI FHIR Public Test Server,» [En línea]. Disponible en: https://hapi.fhir.org/

[25]«Eclipse IDE,» [En línea]. Disponible en: https://eclipseide.org/

[26] «Gradle,» [En línea]. Disponible en: https://gradle.org/

[27] «Apache Maven,» [En línea]. Disponible en: https://maven.apache.org/

[28] «SQLite,» [En línea]. Disponible en: https://www.sqlite.org/

[29] «UML,» [En línea]. Disponible en: https://www.uml.org/

[30] «SNOMED CT Browser,» [En línea]. Disponible en: https://termbrowser.nhs.uk/?perspective=full&conceptId1=404684003&edition=uk-edition&release=v20240731&server=https://termbrowser.nhs.uk/sct-browser-api/snomed&langRefset=999001261000000100,999000691000001104