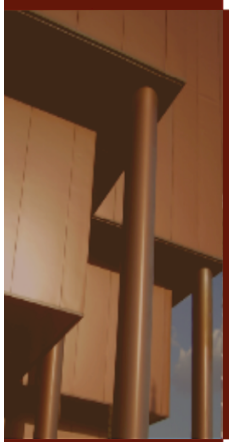
Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Monitorización biométrica en tiempo real con dispositivo móvil y plataforma de análisis de datos

Autor: Antonio Javier Nieto García
Tutor: Juan Antonio Ternero Muñiz



Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Ingeniería de Telecomunicación

Monitorización biométrica en tiempo real con dispositivo móvil y plataforma de análisis de datos

Autor:
Antonio Javier Nieto García

Tutor:
Juan Antonio Ternero Muñiz
Profesor colaborador

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Monitorización biométrica en tiempo real con dispositivo móvil y plataforma de análisis de datos

Autor: Antonio Javier Nieto García

Tutor: Juan Antonio Ternero Muñiz

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A todas las personas que han estado, están y estarán siempre ahí.

Agradecimientos

Mientras escribo estas líneas, soy consciente de que una etapa importante de mi vida está dando a su fin. Durante estos cuatro años, son muchos los sentimientos y vivencias por los que he pasado, donde la dedicación plena y el esfuerzo han sido una constante diaria. Cuando comencé esta etapa, supe desde un primer momento el enorme desafío al que me enfrentaba, pero nunca tuve dudas de que finalmente podría superarlo y alcanzar la satisfacción de conseguir lo que me había propuesto.

Al mismo tiempo, no puedo evitar acordarme de todas y cada una de las personas que han sido partícipes de este logro, a las cuales me gustaría dedicar algunas palabras.

Gracias a mis padres, Luis e Isabel, por su incansable apoyo, sois mi ejemplo a seguir. Gracias a vosotros, hoy estoy aquí, cumpliendo un sueño.

Gracias a mi hermana, Gema, por estar siempre ahí, todo un ejemplo de superación. Nunca te faltará un hombro en el que apoyarte.

Gracias a mi pareja, Ana, por no soltarme nunca de la mano y acompañarme hasta donde estoy hoy. Siempre quedarán en mi recuerdo esas tardes de biblioteca y cafés. Me has enseñado a disfrutar de la vida. Te quiero.

Gracias a los amigos que siempre estuvieron ahí, en especial a José Manuel Noguerol, por compartir tantos y tantos momentos juntos a lo largo de estos cuatro años.

Gracias a las empresas Physio R&D, por ayudarme a crecer, y a Cortrium, por darme la oportunidad de usar el C3 en mi trabajo.

Por último, gracias a mi tutor, Juan Antonio Ternero, y a Isabel Román, por sus consejos que me han permitido llevar adelante este trabajo.

A todos ellos,

GRACIAS.

Antonio Javier Nieto García

Sevilla, 2017

Resumen

En este proyecto se va a crear un sistema autónomo capaz de monitorizar parámetros vitales de un usuario en tiempo real, detectando situaciones críticas y actuando de forma inmediata mediante el envío de notificaciones al usuario afectado. Para ello, se hace uso de un dispositivo holter inalámbrico situado en el tórax, un dispositivo móvil y una plataforma de análisis de datos.

Además, el sistema es capaz de convertir los datos, tomados por el holter, a un formato estandarizado, para que los profesionales médicos puedan acceder a ellos desde sus equipos.

Durante su desarrollo, se han tenido en cuenta aspectos como la seguridad de los datos y la escalabilidad del sistema, con la idea de poder ser aplicado a casos de uso reales.

Abstract

The aim of this project is the development of an autonomous system capable of monitoring, in real time, vital parameters of the user, detecting critical situations and acting immediately by sending notifications to the affected user. To do this, the system uses a wireless holter monitor located on the thorax of the user, a mobile device and a data analytics platform.

In addition, the system converts the data collected by the holter monitor to a standard format, in order to allow medical professionals to access them using their medical equipments.

During all the development, some aspects like data security and scalability have been considered with the purpose of being used in real cases.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

ANSI	American National Standards Institute
API	Aplication Programming Interface
BLE	Bluetooth Low Energy
CEP	Complex Event Processing
DAS	Data Analytics Server
DICOM	Digital Imaging and Communication in Medicine
ECG	Electrocardiography
EDF	European Data Format
EEG	Electroencephalography
EMG	Elecromyogram
FCM	Firebase Cloud Messaging
HL7	Health Level-7
HTTPS	Hypertext Transfer Protocol Secure
Hz	Hertzio
IoT	Internet of Things
JDK	Java Development Kit
JSON	JavaScript Object Notation
MQTT	Messaging Queue Telemetry Transport
RDBMS	Relation Database Management System
REST	Representational State Transfer
SCP-ECG	Standard Communications Protocol for computer assisted – Electrocardiography
SDK	Software Development Kit
SMS	Short Messaging Service
URI	Uniform Resource Indentifier
XML	Extensible Markup Language

1 INTRODUCCIÓN

"Ser el hombre más rico del cementerio no es lo que más me importa. Ir a la cama a la noche diciendo 'Hemos hecho algo maravilloso' es lo que realmente me preocupa."

- Steve Jobs -

Cada vez más, la tecnología va formando parte de todos y cada uno de los aspectos de nuestra vida, con el principal objetivo de hacerla más fácil. Dentro de estos aspectos, cabe destacar su presencia en el campo de la medicina, la cual ha avanzado enormemente desde que la tecnología irrumpió en ella, quedando de manifiesto en la evolución de la esperanza de vida de las personas a nivel global.

1.1. Motivación

Ante la enorme importancia que tiene la tecnología en la mejora de la vida de las personas, siempre he tenido predilección por enfocar mi carrera profesional hacia ese aspecto, donde mi esfuerzo pueda ser útil para el resto de la sociedad.

Durante mi estancia de prácticas en una empresa de Copenhague (Dinamarca), entré en contacto con un proyecto en el que el elemento fundamental era un sensor biométrico que monitorizaba varios parámetros vitales, los cuales eran ofrecidos a un profesional médico con el fin de evaluar el progreso del paciente durante un período de rehabilitación. Tras unos meses de trabajo, descubrí ciertas carencias en el sistema que podían mejorar aún más la utilidad de este sensor, por lo que decidí enfocar mi Trabajo de Fin de Grado a la creación de un sistema que fuese capaz de suplir dichas carencias y hacer un uso más provechoso de dicho sensor.

Para hacer uso del sensor en mi proyecto, solicité permiso a la empresa desarrolladora exponiéndoles mi idea, obteniendo una respuesta positiva por su parte. A partir de ese momento, comencé a trabajar para alcanzar los objetivos que se mencionan en el siguiente apartado.

1.2. Objetivos

El objetivo principal de este proyecto es crear un sistema autónomo capaz de monitorizar los parámetros vitales de un paciente en tiempo real, detectando determinados eventos y actuando mediante el envío de notificaciones al paciente o profesional médico, con la finalidad de disminuir el tiempo de reacción ante casos de urgencia. Para ello, se utiliza un dispositivo holter inalámbrico situado en el tórax del usuario, el cual se conecta con el dispositivo móvil de éste para enviar los datos al servidor.

Además de esto, se pretende convertir los datos de la monitorización al formato estándar *EDF+* [1], de manera

que cualquier profesional médico pueda tener acceso a ellos usando un equipo médico que soporte dicho estándar.

Tal y como se ha concluido en varias publicaciones de ámbito científico, existen casos en los que la mortalidad de determinadas enfermedades se ve aumentada en gran medida si no se actúa con la rapidez suficiente ante su aparición. Un ejemplo de ello se menciona en la publicación *Significado pronóstico de las arritmias en el Infarto Agudo de Miocardio Transmural* [2], donde se indica que cerca del 65% de la mortalidad por Infarto Agudo de Miocardio (IMA) ocurre en la primera hora, donde cerca del 60% de los casos se podrían evitar aplicando desfibrilación. Ante tales situaciones, la temprana detección de estas anomalías es un elemento imprescindible en un sistema como el que se trata en este proyecto.

Otro de los objetivos principales de este proyecto, es facilitar la labor de los profesionales a la hora de realizar un seguimiento a los pacientes, evitando tener que estar pendiente del monitor durante todo el período de monitorización. Mediante el uso de este sistema, el profesional médico podrá acceder rápidamente, desde su equipo, a los resultados de la monitorización de cada paciente de forma más rápida y eficiente.

Otro punto clave, dentro de los objetivos, es la seguridad de los datos. Este sistema va a manejar datos críticos de los pacientes, por lo que debe ofrecer la protección adecuada, tanto en la transmisión como en el almacenamiento.

Y como último punto a destacar, se ha hecho énfasis en la escalabilidad del sistema, de manera que se pueda aplicar a un amplio número de usuarios. El DAS (*Data Analytics Server*)[1] está preparado para recibir millones de eventos por segundo [3], el formato de los mensajes está preparado para identificar de forma única a cada usuario y los datos de cada paciente son eliminados del sistema una vez que se han estandarizado.

Como resumen, los objetivos que se pretenden conseguir con la realización de este proyecto son:

1. Monitorización y detección de eventos en tiempo real.
2. Estandarización de los datos.
3. Notificaciones rápidas y eficientes.
4. Seguridad en los datos.
5. Escalabilidad del sistema.
6. Alta disponibilidad del sistema.

1.3. Plan de trabajo

La realización de este proyecto se puede organizar en varias fases, las cuales comprenden aspectos de investigación, diseño, implementación, pruebas y documentación. Dichas fases se enumeran a continuación:

1. Estudio del estado del arte, donde se valoran las distintas tecnologías que pueden suplir las necesidades de este proyecto.
2. Concepción de la idea, seleccionando las tecnologías que se van a usar y estudiándolas en profundidad.
3. Creación de una primera versión de la aplicación Android, abarcando únicamente la conexión con el sensor.
4. Estudio de los datos ofrecidos por el sensor, con la finalidad de valorar la utilidad de estos y trabajar en su interpretación.

[1] Producto ofrecido por la empresa WSO2 que consiste en una plataforma de análisis de datos en tiempo real y procesamiento de eventos complejos. Además, permite el uso de herramientas de *Machine Learning*.

5. Primera fase del diseño del servidor, que comprende la instalación y configuración del producto WSO2 DAS.
6. Creación de una segunda versión de la aplicación Android, enfocándonos en la conexión con el servidor, así como en la seguridad y el formato de los mensajes.
7. Estudio de los distintos estándares de datos disponibles, seleccionando el que mejor se adapta a las necesidades del sistema.
8. Segunda fase del diseño del servidor, donde se implementa un script de Python, que se encarga de la estandarización de los datos y la conexión de éste con el DAS.
9. Estudio e implementación de un sistema de alertas, adaptando para ello el DAS y el script de Python.
10. Realización de una batería de pruebas que garanticen la fiabilidad del sistema con datos reales.
11. Elaboración de la documentación del proyecto.

1.4 Estructura de la memoria

En ese apartado se pretende comentar la organización y estructura de los contenidos de esta memoria, aportando una breve descripción de cada uno de ellos. El orden propuesto está asociado a la evolución temporal del desarrollo de este proyecto.

1. **Introducción**. Se trata del primer capítulo de esta memoria, en el cual se introduce brevemente la idea, motivaciones y objetivos del proyecto, así como la estructura del contenido.
2. **Estado del arte**. En este capítulo se entra en profundidad en las tecnologías utilizadas en este trabajo, además de comentar las diferentes soluciones que ofrece el mercado para las necesidades que se pretenden cubrir.
3. **Diseño**. En el contenido de este capítulo se detalla la estructura del sistema, así como la finalidad y el uso de las distintas tecnologías que se van a usar en éste. Además, se indican los motivos que han llevado a la elección de cada elemento del diseño.
4. **Implementación**. En este capítulo se recogen todos los aspectos del desarrollo del sistema, acorde al diseño indicado.
5. **Pruebas**. En este capítulo se describen las distintas pruebas a las que se verá sometido el sistema para verificar su correcto funcionamiento.
6. **Conclusiones**. Se trata del capítulo final de esta memoria, en el cual se describen las distintas conclusiones obtenidas durante la realización de este trabajo, así como posibles desarrollos futuros.

2 ESTADO DEL ARTE

Ante la realización de un proyecto de estas características, es fundamental realizar un estudio previo y detallado de las tecnologías y soluciones que existen actualmente en el mercado, de manera que el sistema sea lo más preciso y eficiente posible. Parte de este estudio consiste en contrastar las distintas alternativas que existen para afrontar un mismo problema, valorando las ventajas y desventajas de cada una de ellas y eligiendo la que mejor se adapte a las necesidades del proyecto.

En este apartado se detallan las distintas tecnologías usadas en la realización de este trabajo, así como sus alternativas y los motivos por los que se han elegido éstas frente a otras.

2.1. WSO2

2.1.1 Descripción

La empresa WSO2 [4], fundada en 2005 y financiada inicialmente por empresas como Intel Capital, ofrece una plataforma de código abierto, enfocada al entorno empresarial y basada en una arquitectura orientada a servicios (SOA). Mediante esta plataforma, WSO2 apuesta por la idea de una arquitectura modular o de componentes, con un alto grado de integración, que ofrezca a los desarrolladores la capacidad de personalizar cada uno de estos componentes acorde a sus necesidades.

La integración de componentes está soportada por Carbon [5], un *middleware*[2] basado en estándares de código abierto y en la arquitectura de OSGi [3], el cual está considerado el núcleo de los productos de WSO2. Este *middleware* permite instalar, iniciar, detener, actualizar y desinstalar los componentes de un sistema de forma sencilla, ocupándose de los problemas de conflicto. En la Figura 1, se muestra un esquema de la arquitectura de WSO2 Carbon, donde podemos apreciar que está compuesto por un núcleo de elementos básicos a los que se le pueden añadir otros adicionales.

[2] *Middleware* es una capa software situada entre las aplicaciones y el sistema operativo, usado con frecuencia en sistemas distribuidos. Además, sirve de capa de abstracción de los componentes hardware, facilitando el trabajo a los desarrolladores, ocupándose de las comunicaciones de entrada y salida de las aplicaciones.

[3] OSGi, cuyas siglas significan *Open Services Gateway initiative*, es un framework de Java para desarrollar y desplegar aplicaciones y librerías modulares.

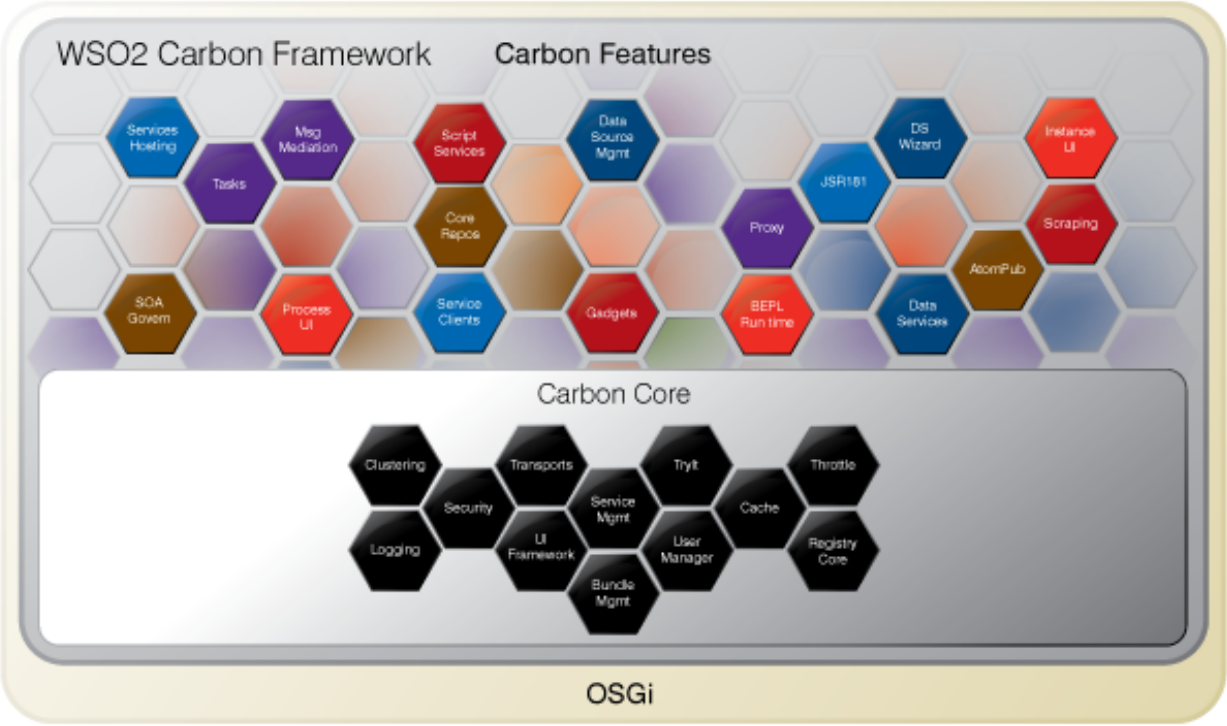


Figura 1. Arquitectura WSO2 Carbon

La plataforma ofrece una amplia variedad de productos de los que podemos hacer uso, de forma simultánea y sin problemas de integración. Estos productos están orientados principalmente a: gestión de APIs, integración de sistemas heterogéneos, gestión de acceso e identidad, IoT y análisis de datos. Unos meses antes de la fecha de publicación de esta memoria, WSO2 ofrecía un mayor número de productos, los cuales se han ido fusionando hasta llegar a los productos que existen actualmente. A continuación, se describen brevemente cada uno de ellos:

- **WSO2 API Manager**. Es una solución del tipo empresarial que soporta los distintos aspectos que conciernen a las APIs, como son: publicación, gestión del ciclo de vida, desarrollo, control de acceso y análisis de uso.
- **WSO2 Enterprise Integrator**. Este producto está orientado a la integración de componentes heterogéneos, de distinto origen, mediando y transformando los datos para que la comunicación entre estos sea viable.
- **WSO2 Identity Server**. Solución orientada a la gestión de identidad en aplicaciones empresariales, servicios y APIs. Hace uso de los estándares más comunes y permite a los administradores implementar políticas de seguridad personalizadas.
- **WSO2 IoT Server**. Este producto permite a las empresas conectarse y gestionar sus dispositivos, controlando las aplicaciones, los aspectos de seguridad y el flujo de datos.
- **WSO2 Data Analytics Server**. Este producto consiste en una plataforma capaz de analizar flujos de datos en tiempo real, procesar eventos complejos y utilizar herramientas de *machine learning*[4].

[4] *Machine Learning* es el término usado para referirse al aprendizaje de las máquinas, tratándose de una rama de las ciencias de la computación y de la inteligencia artificial, cuyo objetivo es el desarrollo de técnicas que permitan a los computadores aprender.

2.1.2 WSO2 Data Analytics Server (DAS)

Como se comentado en el subapartado anterior, el DAS es uno de los productos de WSO2, el cual ofrece una alta capacidad de procesado y análisis de datos, pudiendo funcionar tanto en sistemas distribuidos como centralizados. Gracias a la capacidad de la que dispone el DAS, su uso puede orientarse a aplicaciones de *Big Data*[5] o IoT.

El DAS está preparado para recibir millones de eventos por segundo desde diferentes fuentes de datos, procesarlos en tiempo real y comunicar los resultados a través de una gran variedad de interfaces. La rápida actuación que ofrece el DAS ante determinados eventos es la principal baza que lo diferencia de la competencia. Además de lo anterior, este producto posee herramientas de análisis interactivo y predictivo (*machine learning*), así como una interfaz web para su gestión.

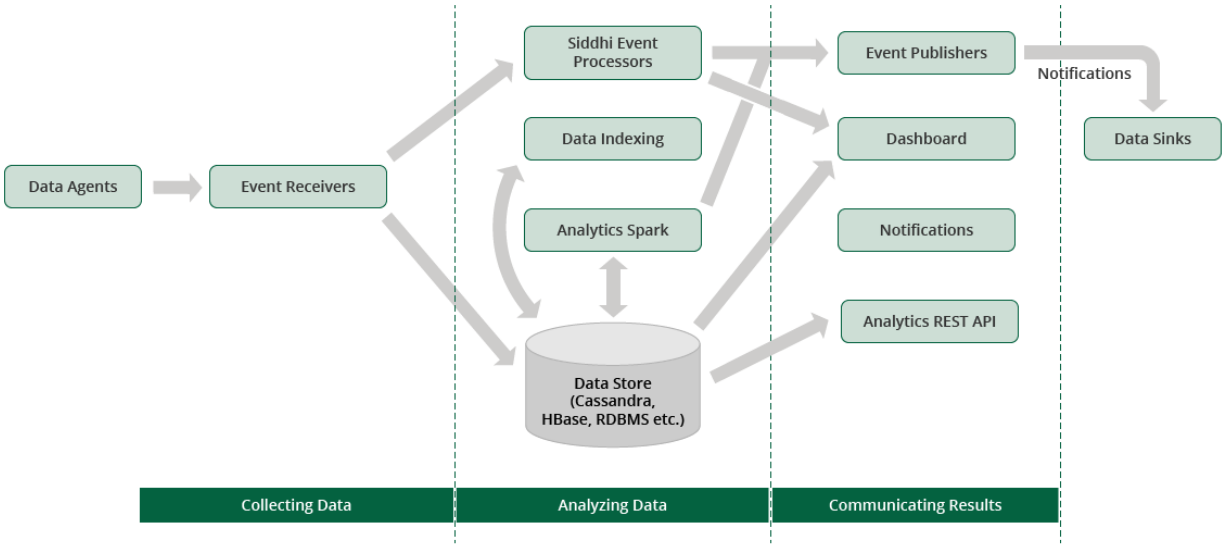


Figura 2. Arquitectura del DAS

Para comprender su funcionamiento, debemos conocer, en primer lugar, su estructura y los elementos que lo componen. A continuación, se definen sus componentes principales:

- **Evento**. Unidad básica de datos utilizada por el DAS.
- **Fuente de datos**. Fuentes externas que envían datos al DAS.
- ***Event Stream***. Elemento usado por el DAS para referirse a una secuencia de eventos del mismo tipo.
- ***Event Receivers***. El DAS recibe los datos a través de estos elementos, los cuales están asociados a un *event stream* cada uno y cumplen la función de adaptadores de entrada. Existen varios tipos de adaptadores que permiten a las fuentes enviar datos usando varios métodos, como son: HTTP, email o MQTT.
- ***Analytics* REST API**. API del tipo REST ofrecida por el DAS para la publicación y extracción de información contenida en éste.
- **Almacén de datos**. Lugar utilizado por el DAS para la persistencia de los datos. Existen tres tipos de almacenamiento: almacenamiento de eventos, almacenamiento de eventos procesados y almacenamiento en archivos. Por defecto, el DAS utiliza una base de datos RDBM con motor H2[6].

[5] Término que hace referencia al gran volumen de datos, tanto estructurados como no estructurados, que no pueden ser tratados usando tecnologías o herramientas software convencionales.
[6] Administrador de base de datos relacionales muy usado en aplicaciones Java.

- **Procesador de eventos**. Motor de procesado de datos en tiempo real, basado en *Siddhi*[7].
- ***Analytics* Spark**. Motor de análisis de datos previamente almacenados, basado en Apache Spark [6], que hace uso de scripts escritos en lenguaje Spark SQL [7].
- ***Execution Plans***. Bloques de código que se ejecutan de forma continuada, utilizados para el análisis en tiempo real. Usa el lenguaje *Siddhi SQL*[8] y utiliza los *events stream* para manejar la entrada y salida de datos.
- **Indexado de datos**. Proceso automático utilizado para el almacenamiento e indexado de los datos en el DAS.
- ***Event Publishers***. Elementos usados por el DAS para publicar los resultados del procesado y análisis de los datos. Se encuentran disponibles varios métodos y protocolos para la publicación de estos datos, por ejemplo: HTTP, MQTT, SMS o email.
- ***Analytics Dashboard***. Herramienta ofrecida por el DAS para la visualización de resultados, compuesta gráficas personalizables.
- **Sumidero de eventos**. Elementos externos que reciben los datos procedentes del DAS.

El funcionamiento del DAS se puede clasificar en tres etapas principales:

- **Recepción de datos**. La recepción de los datos en el DAS se realiza a través de los *event receivers* o la API REST. Una vez se reciben, estos datos pueden almacenarse para un análisis posterior o ser procesados en tiempo real.
- **Análisis de datos**. Una vez dispone de datos, el DAS puede ejecutar tres tipos de análisis: en tiempo real, por lotes y predictivo.
- **Comunicación de resultados**. Tras la fase de análisis de los datos, el DAS debe comunicar los resultados en forma de notificación o visualización mediante gráficas.

El funcionamiento del DAS está regido por el uso de *event streams*, los cuales son elementos habituales en herramientas de procesado de grandes cantidades de datos. Además de estos elementos, el núcleo de funcionamiento del DAS son los *execution plans*, los cuales permiten al DAS ejecutar un análisis en tiempo real de los datos. La versión que se ha utilizado en este proyecto es la 3.1.0.

2.1.3 Soluciones existentes en el Mercado

En términos generales, es difícil encontrar otras soluciones que tengan la misma versatilidad que WSO2, debido a la gran cantidad de productos y servicios que ofrece, siendo todos ellos de código abierto. Pero si tratamos de abarcar este tema desde una perspectiva individual, tratando la funcionalidad cada producto por separado, sí podríamos entrar en comparaciones del mismo nivel.

En el caso del DAS, estamos hablando de un CEP (*Complex Event Processing*) que puede actuar, tanto en tiempo real como con datos almacenados. En lo que concierne al mercado, existen varias soluciones de las que hablaremos a continuación:

- **Apache Spark** [6]

  El motor interno de procesamiento de datos del DAS está basado en su práctica totalidad en Apache Spark, por lo que las diferencias son escasas. La ventaja principal de Apache Spark frente al DAS es la enorme comunidad de desarrolladores e inversores, que supera con creces a WSO2.

[7] Motor de procesamiento de eventos complejos, de código abierto.
[8] Lenguaje de consultas, basado en SQL, diseño para procesar flujo de eventos y detectar ocurrencias de eventos complejos.

- **Apache Hadoop** [8]

  Se trata de otro producto bastante conocido en el mercado de los motores de procesamiento de datos, pero a diferencia del DAS, Apache Hadoop no tiene la capacidad de analizar los datos en tiempo real y su rendimiento es aproximadamente cien veces menor que Apache Spark y, por lo tanto, menor que el DAS.

- **Apache Samza** [9]

  Apache Samza es otro de los productos que pueden abarcar cierta capacidad de mercado de los motores de procesamiento y análisis de datos, aunque en comparación con el DAS, la diferencia principal es que no dispone de capacidad de procesamiento de datos almacenados, sino únicamente de datos en tiempo real.

Todos los productos anteriores se corresponden con soluciones de código abierto, lo cual es una ventaja notable cuando hablamos de proyectos en fase inicial y que carecen de inversión. Aun así, es interesante valorar las alternativas de pago que existen en el mercado. A continuación, se mencionan dos de ellas:

- **Tibco Streambase** [10] **e IBM Streams** [11]

  Como se ha comentado, estas alternativas no son de código abierto, por lo que habremos de pagar una licencia para hacer uso de las mismas. La ventaja de este tipo de soluciones son el soporte técnico, aunque el DAS también ofrece un soporte técnico si pagamos por ello.

A lo largo de este apartado, se ha descrito brevemente algunas de las soluciones existentes en el mercado, que puedan suplir la función del WSO2 DAS. Aunque existen algunas alternativas que puedan hacernos dudar si la elección del DAS ha sido la correcta para este proyecto, es necesario destacar que ninguno de los anteriores ofrece la amplia gama de productos y la capacidad de integración de WSO2. Esto es un punto clave a la hora de decidir la solución que mejor se adapte a nuestro sistema, pues si en un futuro queremos ampliar la funcionalidad de éste, la capacidad de integración de los productos de WSO2 pueden jugar una baza importante.

En resumen, las principales ventajas que se pueden destacar del uso del DAS frente a otras soluciones son las siguientes:

- Integración con otros productos interesantes de WSO2.
- Análisis en tiempo real y de datos almacenados.
- Alto rendimiento, comparable a Apache Spark.
- Escalabilidad.
- Personalización.
- Código abierto.

2.2. Estándar EDF+

2.2.1 Descripción

El estándar EDF+ es una evolución del estándar EDF, el cual tiene su origen en el año 1987, donde un grupo de ingenieros médicos se reunieron en un congreso con motivo de encontrar una solución para probar sus

algoritmos, diseñados para el análisis del sueño, a otros tipos de datos. De ese congreso surgió la primera idea de un estándar de intercambio de datos, llamado EDF. Este formato fue publicado por primera vez en 1992 [12], convirtiéndose en el estándar de facto para EEG.

A principios de 2002, se desarrolló una versión más avanzada de este formato, llamada EDF+, la cual era compatible con el formato anterior, pero permitía almacenar grabaciones discontinuas, añadir anotaciones, estímulos y eventos. EDF+ puede tratar con diversas señales como: EMG, ECG o parámetros QRS.

Este formato se publicó por primera vez en 2003 [13], es actualmente soportado por multitud de equipos médicos y es un estándar no propietario. Está compuesto por una cabecera, donde se añade información del paciente, hospital y fecha de la grabación, y uno o más grabaciones de datos vitales.

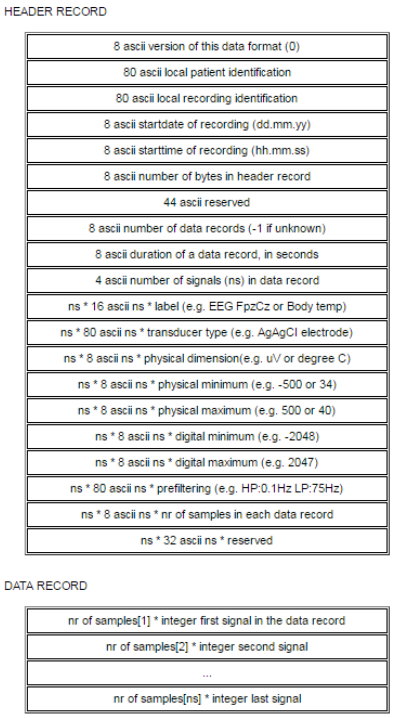


Figura 3. Estructura archivo EDF+

2.2.2 **Alternativas de Mercado**

El problema del intercambio de datos entre sistemas heterogéneos siempre ha sido y es un problema recurrente, de manera que son muchas las soluciones que han surgido para ello. Estas soluciones consisten en distintos formatos de intercambio de información, con el fin de que los datos procedentes de cualquier fuente puedan ser interpretados por equipos independientes a ésta. En nuestro caso, hemos elegido el formato EDF+, pero existen otros disponibles que podrían realizar la misma función. A continuación, se mencionan algunos de ellos, así como sus principales características.

- **DICOM** [14]

  Es un estándar mundialmente conocido para el intercambio de imágenes médicas, pensado para la visualización, almacenamiento, impresión y transmisión de éstas. Este formato tiene una estructura notablemente más compleja que EDF+, pues está pensado para datos médicos de mayor complejidad como son: radiografías o resonancias magnéticas. Debido a esta complejidad, se descartó su uso frente a EDF+, pues las necesidades de este trabajo no requerían de tal grado de complejidad.

- **SCP-ECG** [15]

  Este estándar está siendo desarrollado por el grupo OpenECG[9], y tiene como finalidad servir como formato de intercambio de datos ECG, permitiendo añadir anotaciones en sus archivos. Este estándar está menos extendido que EDF+ y DICOM, por lo que se descartó desde un principio.

- **HL7 aECG** [16]

  Este estándar está dentro del grupo HL7 [10] y está pensado para el almacenamiento y transmisión de los datos ECG de los pacientes. Fue introducido en 200, su formato está basado en XML y fue aceptado por ANSI en 2014. La elección de EDF+ sobre este formato tiene su motivo en la amplia aceptación de EDF+ en Europa, frente a HL7 aECG apoyado mayormente por instituciones americanas.

Para resumir, los principales motivos que llevaron a la elección del formato EDF+, frente a las otras alternativas que ofrece el mercado, fueron varios:

- Estructura simple.
- Tamaño reducido de los archivos.
- Ampliamente aceptado por los equipos médicos.
- Utilizado por grandes empresas como *Mathworks* o *Texas Instrument* [17].

2.3. Cortrium C3 Holter Monitor

2.3.1 Descripción

El dispositivo holter que vamos a utilizar en este trabajo se llama Cortrium C3 Holter Monitor y está desarrollado por la empresa Cortrium [18]. Se trata de un dispositivo inalámbrico, con una batería que permite monitorizar a un paciente durante más de 24 horas, capaz de obtener los siguientes parámetros: ECG, ritmo cardíaco, ritmo de respiración, temperatura corporal y datos de acelerómetro.

Las principales especificaciones del dispositivo la podemos encontrar en el Anexo A: Características de los equipos, a las que podemos añadir que tiene una tasa de muestreo de hasta 250Hz, batería con duración de hasta 48 horas recargable por microUSB y una resolución de 64 bits. Además, dispone de herramientas de desarrollo para entornos Android e iOS, permitiendo a los desarrolladores hacer uso de éste en sus aplicaciones.

Actualmente, está en fase de desarrollo, probándose en pacientes y siendo usado en varios proyectos de investigación [19], pudiéndose adquirir únicamente para investigaciones y pruebas.

[9] Consorcio interdisciplinar formado por autoridades médicas, cardiólogos, ingenieros y productores de equipos médicos.
[10] Conjunto de estándares internacionales para la transferencia de datos clínicos y administrativos entre equipos de diferentes proveedores.

Figura 4. Cortrium C3 Holter Monitor

2.3.2 Alternativas de Mercado

Dentro del ámbito medico existen una gran cantidad de dispositivos capaces de monitorizar los parámetros vitales de los pacientes, aunque no es fácil encontrar un dispositivo de las características del Cortrium C3. A continuación, se van a describir varias opciones de mercado que pueden servir como alternativa a sensor usado en este proyecto.

- **Philips DigiTrak XT** [20]

  Este dispositivo es el holter más ligero de la empresa Philips, diseñado para ofrecer la mayor comodidad a los pacientes, aunque al disponer de 5 cables para los puntos de contacto, puede acabar provocando el efecto contrario. Para su funcionamiento, necesita de un batería de categoría AAA, ofreciendo una duración de hasta 7 días.

Figura 5. DigiTrak XT – Philips

- **EVO Digital Recorder** [21]

  Este dispositivo está desarrollado por la empresa SpaceLabs, es muy ligero y ofrece una duración de la batería de hasta 48 horas. Dispone de un cargador especial que debemos de adquirir de forma

independiente y hace uso de cables. La precisión de este dispositivo es menor que la del Cortrium C3 y Philips DigiTrak XT.

Figura 6. EVO Digital Recorder – SpaceLabs

- **Medilog FD5plus** [22]

  Este dispositivo, desarrollado por la empresa Schiller, ofrece funcionalidades de análisis del ECG del paciente, su batería dura hasta 80 horas de grabación y la precisión es similar a la del Cortrium C3. Para el contacto con el cuerpo del paciente, este dispositivo hace uso de cables.

Figura 7. Medilog FD5plus - Schiller

Como podemos apreciar en las descripciones de los dispositivos anteriores, la duración de la batería es igual o mayor a la del Cortrium C3 y, en temas de precisión, el primero de ellos supera a la de éste. Aun así, existe varias características que hacen del Cortrium C3 el dispositivo ideal para este sistema.

El primero de ellos es la ausencia de cables, pues a la hora de monitorizar a un paciente, la presencia de cables puede provocar cierta incomodidad. Esto último es un punto a tener en cuenta en este tipo de dispositivos. Por otro lado, ninguno de estos dispositivos ofrece la posibilidad de establecer una conexión inalámbrica con un dispositivo móvil, mediante BLE, como es el caso del Cortrium C3.

2.4. Aplicación Android

2.4.1 Descripción

La aplicación Android usada en este proyecto forma parte de los elementos principales del mismo, pues permite establecer la conexión con el sensor biométrico, recolectar los datos obtenidos por éste y enviarlos al servidor.

El sistema operativo Android está basado en el núcleo de Linux, es multi-plataforma (válido para dispositivos móviles, *tablets*, relojes y televisiones inteligentes) y gratuito. Fue inicialmente desarrollado por la empresa Android Inc. [23] y adquirido posteriormente por la empresa Google. En la fecha de realización de esta memoria, la última versión de Android es la 8.0.

Los motivos por los que se ha elegido este sistema operativo son varios, pudiendo destacar los siguientes:

- Conocimientos previos en el desarrollo de aplicaciones Android.
- Acceso al SDK ofrecido por la empresa Cortrium.
- Gran cantidad de dispositivos que lo soportan.

Para ejecutar la aplicación, el dispositivo debe tener una versión del sistema operativo Android 4.4 o superior y disponer de conexión Bluetooth Low Energy. El requisito de la versión del sistema operativo se debe al SDK de Cortrium, pues no está soportado para versiones anteriores a la mencionada, mientras que la conexión BLE (*Bluetooth Low Energy*)[11] es necesaria para establecer la conexión con el sensor.

2.4.2 Alternativas de Mercado

En lo que respecta a aplicaciones móviles, existen varias alternativas de desarrollo que se valoraron inicialmente, pero que por motivos que se detallarán en este subapartado, se acabó decidiendo utilizar Android. A continuación, se mencionan dichas alternativas:

- **iOS** [24]

  Este sistema operativo es propiedad de la compañía Apple, el cual solo está permitido instalarse en dispositivos de Apple, lo cual restringe la variedad de dispositivos que lo soportan. Al igual que Android, está ampliamente extendido, pero mi conocimiento sobre este sistema operativo es limitado, siendo éste el motivo de su descarte.

- **Windows Phone** [25]

  Se trata de un sistema operativo desarrollado por la compañía Windows y no tiene política de restricción de uso, como es el caso de iOS. Los dos motivos principales que me llevaron a descartar su uso fueron el desconocimiento del mismo y el bajo porcentaje de dispositivos que usan este sistema operativo actualmente.

[11] Se trata de la versión 4.0 de la tecnología *Bluetooth*, la cual contiene mejoras de eficiencia y uso de energía.

Como se puede deducir de las alternativas anteriores, la primera de ellas parece bastante atractiva pero debido a las causas que se exponen, finalmente se tomó la decisión de utilizar el sistema operativo Android para el desarrollo de la aplicación móvil.

2.5. Script de Python

2.5.1 Descripción

Como se ha comentado previamente en el apartado 1.2, uno de los objetivos del sistema es el de convertir los datos adquiridos por el sensor al formato EDF+. Para realizar esto, debemos de utilizar herramientas específicas, pues los archivos EDF+ no utilizan caracteres ASCII interpretables por las personas. Por este motivo, fue necesario valorar las distintas herramientas disponibles en el mercado para posteriormente hacer uso de la que mejor se adapte a las necesidades del sistema.

En el caso concreto del sistema desarrollado, se ha utilizado la librería *PyEDFlib* [26] del lenguaje Python. Se trata de una librería de código abierto, capaz de leer y escribir archivos EDF y EDF+. Esta librería está basada en la librería *EDFlib* implementada para los lenguajes C y C++. Para hacer uso de ella, necesitamos tener una versión de Python 2.7/3.3 o superior, así como tener instalada una versión del módulo *Numpy* [27] igual o superior a 1.6.2.

2.5.2 Alternativas de Mercado

Al igual que existe la librería *PyEDFlib* para el lenguaje Python, existen otras librerías que nos permiten leer y escribir archivos EDF+, ya sean de Python o de otros lenguajes. A continuación, se muestran varias alternativas:

- **EDFlib** [28]

  Tal y como se ha mencionado anteriormente, *PyEDFlib* está basada en esta librería, lo cual nos lleva a la posibilidad de usar esta librería en un programa escrito en los lenguajes C/C++. Al ser Python un lenguaje de más alto nivel que éstos, se descartó su uso.

- **EDF4J** [29]

  Esta librería está disponible para el lenguaje Java y nos permite convertir archivos de texto en archivos EDF+. Debido a esto último, se optó por no elegir esta opción, pues el proceso de conversión es más complejo.

- **Matlab** [30]

  Además de los lenguajes anteriores, existe una herramienta para manejar archivos EDF+ en Matlab. Aunque Matlab ofrece una potencia de computación bastante alta, su ejecución en un sistema es bastante costosa. Contrastando esto con la ligereza de un programa de Python, se tomó la decisión de no usar Matlab.

Tras haber comparado las distintas alternativas con la opción elegida para este trabajo, es fácil concluir que el uso de la librería *PyEDFlib* del lenguaje Python es la mejor opción para las necesidades de este proyecto. Como punto destacado de esta opción frente a las demás, podemos mencionar la ligereza de los programas escritos en este lenguaje, siendo éste de alto nivel y ofreciendo una amplia variedad de herramientas y librerías.

2.6 Firebase Cloud Messaging

2.6.1 Descripción

Esta tecnología perteneciente a Google [31], consiste en una solución de mensajería multiplataforma que permite enviar mensajes de forma segura y gratuita. Esto nos permite realizar la comunicación desde el servidor hacia el cliente, sustentando dicho proceso sobre la *cloud* de Google, por lo que esa carga de procesamiento no recae sobre nuestro sistema.

Las ventajas de esta tecnología son varias:

- Compatibilidad con un gran número de plataformas.
- Uso gratuito.
- Envío de mensajes a dispositivos individuales o a grupos.
- Fácil implementación en nuestra aplicación.
- Escalabilidad.
- Sin carga de procesamiento sobre nuestro Sistema.

Para el envío de notificaciones, FCM utiliza *tokens* que son únicos para cada aplicación en cada dispositivo, lo cual permite identificar de forma universal a los destinatarios.

2.6.2 Alternativas de Mercado

Respecto a este tipo de tecnología, es difícil encontrar una alternativa que pueda asemejarse a las características de FCM, siendo actualmente la opción mayoritaria para enviar notificaciones de tipo *push*[12] desde el servidor a los clientes.

Como un ejemplo de las alternativas disponibles actualmente, podemos mencionar Pushy [32]. Se trata de una plataforma de entrega de notificaciones *push*, compatible con varias plataformas, que ofrece una versión gratuita con funcionalidades limitadas, siendo necesario pagar por obtener todas las funcionalidades.

La elección de FCM frente a Pushy no ha sido difícil, pues simplemente el hecho de que no ofrezca una versión gratuita con unas funcionalidades aceptables, y que no esté lo suficientemente extendida como para ser de confianza, hizo que la opción elegida fuese FCM.

[12] Mensajes enviados directamente a los dispositivos móviles, usado por los desarrolladores para informar a los usuarios.

3 DISEÑO

En este capítulo se va a indagar en el diseño del sistema desarrollado en este proyecto, describiendo su estructura y la conexión entre los distintos elementos que lo forman. En primer lugar, se introducirá al lector en el contexto de este proyecto, presentando las necesidades y limitaciones asociadas al mismo. Tras ello, se presentará el esquema de red del sistema, el cual nos valdrá para entender el funcionamiento de éste, y se dará paso al capítulo 4, donde se trata la implementación.

3.1 Introducción

Tal y como se comentó en el apartado 1.2, el sistema debe monitorizar a los pacientes en tiempo real, siendo capaz de detectar eventos críticos y alertar al usuario y/o profesional médico con la mayor celeridad posible. Esta monitorización debe de afectar lo menos posible al paciente, pudiendo funcionar en segundo plano, sin que éste deba de estar constantemente con la aplicación abierta.

Además de esto, los datos que obtenemos del sensor deben almacenarse en archivos con un formato estandarizado, en este caso EDF+, que permita a los profesionales médicos acceder a ellos desde equipos independientes. Otro de los aspectos que debemos tener en cuenta al diseñar el sistema es la escalabilidad, pues este sistema debe poder aplicarse a una gran cantidad de usuarios.

El sistema que cumpla estos requisitos debe estar instalado en un servidor con direccionamiento público, preferiblemente basado en Linux, para que podamos acceder a éste desde cualquier red externa. La disponibilidad debe ser total, de manera que pueda funcionar de manera ininterrumpida e indefinida.

Debido al carácter privado de los datos con los que va a trabajar, el sistema debe de aportar las medidas necesarias para proteger los datos, teniendo en cuenta tanto la transmisión como el almacenamiento.

3.2 Limitaciones

En el apartado anterior se han descrito las necesidades asociadas a este proyecto, las cuales deben estar presentes a la hora de diseñar e implementar el sistema. Pero como en cualquier proyecto real, existen limitaciones que pueden afectar al funcionamiento de éste. A continuación, se mencionan algunas de ellas:

- **Duración de la batería**

  La duración de la batería de los dispositivos, tanto del sensor como del terminal móvil, debe soportar, al menos, 16 horas de grabación ininterrumpida, pudiéndose utilizar las 8 horas restantes del día para ser recargadas.

- **Monitorización no invasiva**

  La monitorización debe ser lo menos molesta posible para los pacientes, evitando posibles molestias como la presencia de cables o la necesidad de tener la aplicación encendida para mantener la monitorización activa.

- **Análisis y detección en tiempo real**

  El análisis y la detección de eventos en tiempo real puede requerir una alta capacidad de computación al sistema, pues cada usuario va a producir una gran cantidad de datos, de forma simultánea a los demás. El sistema debe tener la capacidad suficiente para no bloquearse ante tal cantidad de información.

- **Almacenamiento limitado**

  El servidor debe estar preparado para recibir una gran cantidad de datos de forma continuada, por lo que, debido a las limitaciones de almacenamiento, debe de gestionarlos de manera eficiente para no agotar los recursos de los que dispone, pudiendo provocar la detención del servicio.

- **Alertas directas y eficientes**

  Cuando se detecte una situación crítica durante la monitorización de un paciente, el sistema debe alertar de inmediato a las personas que sean necesarias, debiendo ser notificadas de forma que no puedan obviar la alerta.

3.3 Esquema de red

Teniendo en cuenta las necesidades y limitaciones mencionadas en los apartados anteriores, se procedió a realizar un esquema del conexionado de red necesario para cumplir todos esos aspectos. El esquema de red es el siguiente:

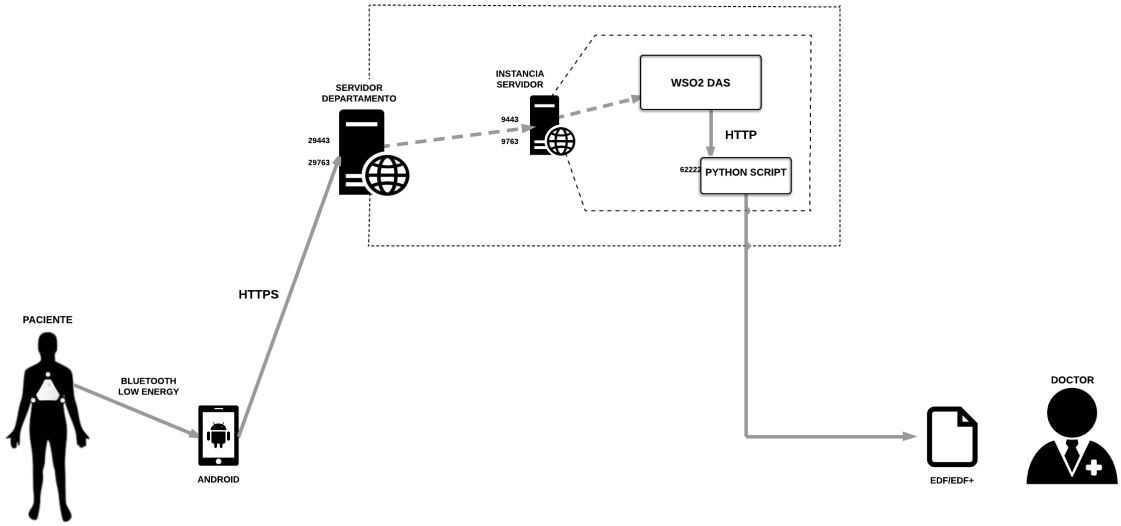


Figura 8. Esquema de red

En la figura, se pueden distinguir tres elementos de conexión muy importantes:

- **Conexión holter - dispositivo móvil**

  Esta transmisión se realiza mediante la tecnología BLE, pues es la única compatible con el sensor y dispone del nivel de seguridad suficiente para proteger la transmisión de datos. El funcionamiento del sensor provoca el envío de datos con una frecuencia de 41.67 Hz, donde los datos correspondientes al ECG se envían dentro de un *array* de seis elementos, debido a que la frecuencia de muestreo del ECG es de 250 Hz. La estructura de los mensajes y el tratamiento que reciben por parte del dispositivo se verán en más detalles en el apartado 4.

- **Conexión dispositivo móvil – servidor**

  La transmisión entre estos dos elementos se realizará a través del protocolo HTTPS, de manera que los datos se transmitan encriptados. El dispositivo móvil establecerá la conexión con el servidor público a través del puerto 29443, el cual redirigirá el tráfico hacia el puerto 9443 de la instancia del servidor que aloja en su interior, en la que está instalada el DAS. La frecuencia de envío será la misma que la del sensor, 41.67 Hz.

  Además de los puertos mencionados, los cuales se utilizan para la transmisión de datos con HTTPS, existe otra alternativa en caso de no poder usar dicho protocolo. Para ello, deberíamos usar el puerto 29763 del servidor público, cuyo tráfico se redirigirá al puerto 9763 de la instancia del servidor. Este puerto está preparado para recibir mensajes mediante el protocolo HTTP, el cual debemos evitar por no ser lo suficientemente seguro.

- **Conexión DAS – script Python**

  Esta conexión tiene lugar en la red interna del equipo servidor, siendo menor la necesidad de protección, por lo que la comunicación se realizará mediante HTTP. El DAS transmitirá los datos de cada paciente, cada dos horas, al servidor Python que está escuchando en el puerto interno 62222. Este servidor está implementado en el script indicado en la Figura 8, y se ocupa de todo el proceso de conversión de los datos al formato estándar EDF+. Para implementar el servidor en el script, se va a utilizar la tecnología *Flask* [33].

3.4 Formato de los mensajes

3.4.1 Estructura

Hasta ahora, no se ha entrado en detalle sobre el formato de los mensajes, siendo uno de los aspectos más importantes de la transmisión. A la hora de diseñar la estructura de los datos que se envían, debemos de tener presente las limitaciones del DAS para su recepción.

Tal y como se comentó en el subapartado 2.1.2, el DAS utiliza los *event receivers* para la recepción de datos, de aquí en adelante, **eventos**. Estos eventos deben seguir un formato determinado, de manera que el DAS pueda interpretarlos adecuadamente. Los eventos pueden enviarse usando 3 formatos distintos: JSON, XML y en texto plano. En este proyecto se va a utilizar el formato JSON, debido a la amplia aceptación y la fácil interpretación de su contenido.

A continuación, se indica la estructura básica de los eventos:

```
{
   "event": {
```

```
        "metaData": {
            "metadata_attribute1": <value1>,
            …
            "metadata_attributeN": <valueN>
        },
        "correlationData": {
            "correlationdata_attribute1": <value1>,
            …
            "correlationdata_attributeN": <valueN>
        },
        "payloadData": {
            "payloaddata_attribute1": <value1>,
            …
            "payloaddata_attributeN": <valueN>
        }
    }
}
```

Si utilizamos esta estructura de mensajes, la cual viene establecida por defecto, el DAS sólo admitirá un único evento por mensaje. Esto es un inconveniente que puede afectar a la eficiencia del sistema, pues la cantidad de mensajes aumentaría hasta ser inabarcable. Para solucionar esto, se ha tenido que recurrir a la capacidad de personalización que caracteriza al DAS, permitiéndonos configurar el formato de entrada de datos de manera que se adapte a la estructura que elijamos. En el Anexo D: Guía de usuario – WSO2 DAS, más concretamente en el apartado que indica cómo crear *event receivers*, se detalla este procedimiento mediante el cual podemos crear un *event receiver* que reciba varios eventos dentro del mismo mensaje.

A continuación, se muestra la estructura de los mensajes que va a usar el sistema, incluyendo los parámetros y valores de ejemplo. Cada mensaje contendrá seis eventos, de manera que la frecuencia de envío se corresponda con la del sensor.

```
[
    {
        "user_id": 129566616042235182634,
        "timestamp": 1.501669934029E12,
        "ecg_value": -1.430511474609375,
        "body_temperature": 36.2,
        "rr_interval":0,
        "respiratory_rate": 30,
        "heart_rate": 108
    },

    .
    .
    .

    {
        "user_id": 129566616042235182634,
        "timestamp": 1.501669934053E12,
        "ecg_value": 6.008148193359375,
        "body_temperature": 36.2,
        "rr_interval":0,
        "respiratory_rate": 30,
        "heart_rate": 108

    }
]
```

Como se ha indicado anteriormente, la frecuencia de muestreo del ECG es seis veces mayor que la del resto de parámetros, por lo que, en los mensajes, el valor del campo referente al ECG irá variando en cada evento, al igual que el *timestamp*, lo cual no ocurrirá con el resto de parámetros.

Estos datos se almacenarán en el servidor usando como clave primaria el identificador de usuario y el valor del campo *timestamp*. Esto evitará que se sobrescriban datos debido a algún error, pues un mismo usuario no podrá enviar un evento con un valor de *timestamp* ya usado previamente, así como distintos usuarios podrán enviar eventos cuyos campos de *timestamp* puedan coincidir.

3.4.2 Atributos

En el subapartado anterior, se trata la estructura de los mensajes, pero no así el contenido de estos. A continuación, se detalla los atributos que los forman:

- **user_id**. Identificador único del paciente o usuario del sistema. Este número se obtiene de la cuenta de Google utilizada para acceder a la aplicación, la cual tiene un identificador único para cada cuenta, por lo tanto, no se obtiene del sensor.
- **timestamp**. Marca temporal asociada a cada evento, expresada en formato de tiempo Unix[13], convertida a milisegundos. Este valor se añade en la propia aplicación móvil, cada vez que se recibe un evento, debido a que el sensor no provee este dato.
- **ecg_value**. Valor de ECG asociado a cada muestra tomada por el sensor, expresado en microvoltios.
- **body_temperature**. Valor de la temperatura corporal medida por el sensor, expresada en grados centígrados.
- **rr_interval**. Valor relativo al tiempo transcurrido entre dos complejos QRS [34]. *Este valor no está disponible en esta versión del sensor.*
- **respiratory_rate**. Ritmo de respiración, expresado en respiraciones por minuto, medido por el sensor.
- **heart_rate**. Valor del ritmo cardíaco medido por el sensor y expresado en número de pulsaciones por minuto.

3.5 Estandarización de los datos

Como se comentó en el apartado 1.2, la conversión de los datos a un formato estándar era una de las necesidades de este sistema, con la finalidad de permitir a los profesionales médicos acceder a ellos desde un equipo que soporte dicho estándar. En el apartado 2.2, se vieron los diferentes estándares que se adaptaban a las necesidades del sistema, y tras analizarlos en profundidad se decidió utilizar el formato EDF+.

En este sistema, la estandarización de los datos la realiza el script de Python, de manera que el DAS enviará a éste los datos a convertir y el script se ocupará de todo lo necesario para crear los archivos EDF+.

3.5.1 Limitaciones

Esta transmisión de datos precisa de un procedimiento más complejo de lo que puede parecer, pues el DAS no está diseñado para la transmisión de grandes cantidades de datos de forma eficiente. Las primeras opciones que pueden ser factibles son dos:

[13] Sistema de descripción de instantes de tiempo que consiste en la cantidad de segundos transcurridos desde la media noche UTC del 1 de Enero de 1970. Este sistema se utiliza en una gran cantidad de equipos computadores.

- **Envío directo de los datos desde el DAS**

  Para llevar a cabo esta opción, deberíamos programar primeramente un evento en el DAS, de manera que cada dos horas se envíen al script los datos de un paciente, mediante el protocolo HTTP. Cuando se produce este evento, el algoritmo, que se está ejecutando de forma ininterrumpida en el DAS, realiza una consulta de los datos correspondientes a ese intervalo y los envía al script a través de un *event receiver*.

  Tras analizar esta opción, surgió un grave problema de eficiencia, pues el DAS envía un único registro de su base de datos por cada mensaje HTTP. Teniendo en cuenta la enorme cantidad de datos procedentes de un intervalo de dos horas de grabación, hace inviable la elección de esta opción.

- **Extracción de los datos haciendo uso de la API REST**

  Aprovechando una función de la API REST que ofrece el DAS para la extracción de datos de la base de datos [35], se valoró hacer uso de ella para que sea el propio script el encargado de extraer los datos. Para ello, el script debe de pasar como parámetros de la petición a la API los valores de un campo interno de la base de datos del servidor, llamado *_timestamp* (prestando atención al primer carácter, pues lo diferencia del atributo *timestamp* enviado por la aplicación Android).

  El valor de este campo lo asigna el DAS internamente y lo añade a todos los registros de su base de datos, por lo que el script debe conocerlo previamente a la petición. Para ello, el DAS debería enviar esta información al script, pero no es posible acceder a este campo interno desde el algoritmo, por lo que no es posible usar esta opción.

3.5.2 Procedimiento

Tras valorar las opciones anteriores, se concluyó que era necesario diseñar un procedimiento más complejo que permitiese la transmisión de los datos desde el DAS al script. A continuación, se detalla dicho procedimiento, acompañándose de una figura que facilite el entendimiento de éste.

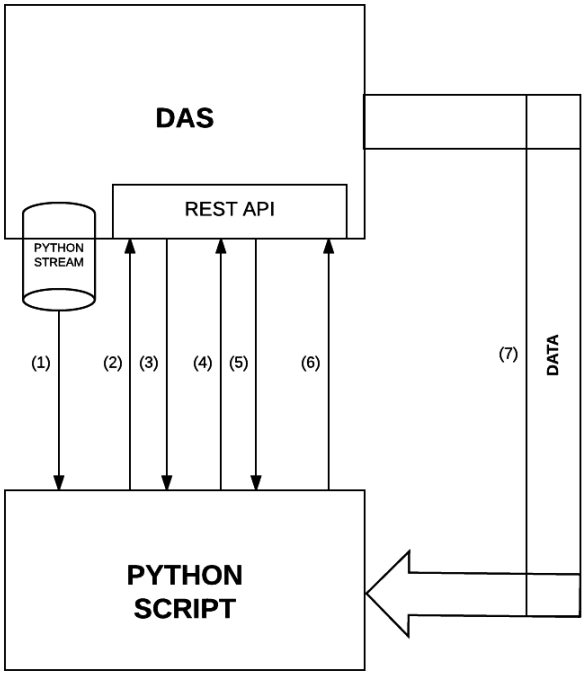


Figura 9. Transmisión de datos DAS-script

1. Cada vez que se lance el evento que indique que la grabación de un paciente ha alcanzado las dos horas, el DAS enviará al script los valores del campo *timestamp* (valor **recibido** desde la aplicación móvil) correspondientes al primer y al último evento de dicho intervalo de grabación.
2. El script hará uso de una función de la API REST [36], la cual permite hacer consultas a la base de datos filtrando por los valores de sus campos. A través de ella, el script obtendrá el registro asociado al primer evento del intervalo de grabación, es decir, el que se corresponda con el primer valor de *timestamp* que el script ha recibido del DAS.
3. Como respuesta a la petición realizada por el script, la API REST envía al script el registro correspondiente al primer evento del intervalo, desde el que podemos acceder al valor del campo **interno** *_timestamp*, el cual es accesible desde la API REST, pero no desde el algoritmo.
4. Al igual que en el paso 2, el script vuelve a solicitar a la API REST [36] un nuevo registro de la base de datos, está vez el correspondiente al último evento del intervalo.
5. Como respuesta a la petición anterior, la API REST devuelve al script el contenido del registro asociado al evento solicitado, conteniendo el valor del campo **interno** *_timestamp* correspondiente.
6. Haciendo uso de la información obtenida de las respuestas de la API REST, el script utilizará otra función disponible para la extracción de datos del DAS [35]. Esta función permite extraer del DAS todos los registros contenidos en un intervalo de tiempo dado por los valores obtenidos del campo **interno** *_timestamp*.
7. La respuesta de la API REST a esta última petición se realiza mediante el envío de un **único** mensaje que contiene todos los registros comprendidos en el rango de tiempo indicado.

Con el procedimiento anteriormente indicado, el script recibirá todos los datos de la grabación dentro de un único mensaje HTTP y, por lo tanto, de forma mucho más eficiente que las alternativas propuestas en el subapartado 3.5.1. Después de haber recibido los datos y que estos hayan sido convertidos al formato EDF+, el script deberá de borrar los registros correspondientes de la base de datos del DAS, cuyo procedimiento se indica en el apartado 3.6.2.

3.6 Persistencia de los datos

3.6.1 Análisis

Una de las limitaciones, que se comentaron en el apartado 3.5.1, es la del almacenamiento, pues el DAS está constantemente recibiendo datos de los pacientes, debiendo de almacenarlos para su posterior conversión al formato EDF+. Esto puede ocasionar un problema, debido a la limitación de los recursos disponibles.

Para analizar este problema, se ha calculado la cantidad de datos que se producen por cada paciente durante una monitorización ininterrumpida. Para ello se ha tenido en cuenta el tamaño que ocupan en memoria los distintos tipos de datos de Java, ya que es el lenguaje usado por el DAS. La tabla siguiente muestra el tamaño de los atributos de un evento.

Tabla 1. Tamaño de los atributos

Atributo	Tipo	Tamaño (bytes)
user_id	Float	4
timestamp	Double	8
ecg_value	Double	8
body_temperature	Double	8
rr_interval	Int	4
respiratory_rate	Int	4
heart_rate	Int	4

Si sumamos el tamaño de todos los atributos que se van a almacenar en el DAS, obtenemos que el tamaño total de estos es de **40 bytes**. Y de forma correlativa, sabiendo que se envían 250 eventos cada segundo, el DAS recibirá un total de **10 kilobytes** por segundo, procedentes del sensor de cada paciente. Para tener una mejor idea de la cantidad de datos que se produce en cada instante de tiempo, se muestra la tabla siguiente:

Tabla 2. Cantidad de datos de un paciente

Intervalo temporal	Tamaño
1 segundo	10 KB
1 minuto	600 KB
1 hora	36 MB
2 horas	72 MB
24 horas	864 MB
1 semana	6,048 GB

Tras calcular el tamaño que se necesitaría para almacenar los datos relativos a un paciente, se tomó la decisión de almacenar únicamente los datos relativos a un intervalo de 2 horas de grabación. Esta decisión se tomó debido a dos motivos:

- La separación de la grabación en varios intervalos permitirá a los profesionales médicos acceder a un instante temporal concreto con mayor facilidad que si sólo existiese un único archivo conteniendo la grabación completa.
- El tamaño correspondiente a un intervalo de grabación de 2 horas es una cantidad razonable si tenemos en cuenta el problema de los recursos limitados.

Cada vez que se conviertan los datos a un archivo EDF+, estos datos se eliminarán de la base de datos del DAS, permitiéndonos así usar de manera eficiente los recursos de almacenamiento disponibles. De esta manera, en el servidor sólo se almacenarían un máximo de 72 MB por paciente.

3.6.2 Borrado de datos

Tal y como se ha indicado en el subapartado anterior, los datos se deben eliminar de la base de datos del DAS una vez se han transmitido al script para su estandarización. Para ello, se va a utilizar una herramienta que ofrece el DAS, a través de un comando, para purgar los datos almacenados. Esta herramienta será ejecutada por el script.

El comando a ejecutar es el siguiente, donde los valores *initial_date* y *final_date* son los correspondientes a los instantes inicial y final del intervalo de grabación.

```
/home/wso2das-3.1.0/bin/analytics-backup.sh -purge -table \
                "ECGSTREAM" -tenantId -1234 -timeFrom ' + initial_date \
                + ' -timeTo ' + final_date
```

La ejecución de este comando elimina los datos de la tabla *ECGSTREAM* situados en el rango temporal dado por los valores *initial_date* y *final_date*. De esta manera, nos aseguramos de que sólo sean eliminados los datos que ya han sido estandarizados.

3.7 Aplicación móvil

Otro de los elementos fundamentales del sistema es la aplicación móvil, encargada de establecer la conexión con el holter, obtener los datos de éste y enviarlos al servidor. El diseño de la aplicación debe cumplir algunos requerimientos para adaptarse a las necesidades del sistema. A continuación, se enumeran algunos de ellos:

- **Acceso seguro**

  El acceso a la aplicación debe hacerse de forma segura e identificando unívocamente al usuario. Para ello, se pretende utilizar *Google Sign-In* [37], de manera que la autenticación se realice mediante una cuenta de Google.

- **Usabilidad**

  La aplicación debe ser lo más simple posible, ofreciendo al usuario una interfaz limpia y clara, que no requiera de instrucciones complejas para ser utilizada.

- **Asociación del sensor**

  Para que el sistema sea lo más robusto posible, la aplicación debe permitir la asociación de un dispositivo sensor, de manera que cuando el usuario inicie la monitorización, su dispositivo móvil establezca la conexión únicamente con el sensor asociado, y no con otros sensores que puedan estar cerca.

- **Monitorización en segundo plano**

  El servicio de monitorización debe poder funcionar en segundo plano, permitiendo al usuario poder seguir usando su dispositivo móvil mientras el servicio está activo. Para realizar esto, se va a hacer uso de *Threads* [38], una herramienta que ofrece el sistema operativo Android para ejecutar código en un hilo diferente al de la interfaz de usuario. Algunas de las ventajas que nos ofrecen los *Threads* son:

  - Ejecución de tareas en paralelo.
  - Compartición del espacio de memoria.
  - Cada hilo tiene su propia pila de ejecución y contador de programa.

  Además de esa herramienta, Android ofrece otras dos alternativas: *AsynTasks* y *Handlers*. La elección de los *Threads* frente a estas alternativas reside en la simplicidad de gestión de éste frente a las otras alternativas.

- **Rapidez y eficiencia**

  Dentro de los objetivos del sistema, encontramos la necesidad de que el sistema sea rápido y eficiente, de manera que la monitorización se realice en tiempo real y se pueda actuar rápidamente ante la detección de algún evento. Por este motivo, el diseño de la aplicación debe ser óptimo, aprovechando los recursos disponibles.

- **Información de la monitorización**

  Otras de las características que debe ofrecer la aplicación es la de informar al usuario del estado de la monitorización. Cuando el servicio está activo, el usuario debe poder saber si la conexión es buena, si la batería del sensor tiene el nivel suficiente y si el servicio se ha iniciado y detenido correctamente.

3.8 Alertas y notificaciones

Otra de las funcionalidades que ofrece el sistema es la de notificar al usuario de la ocurrencia de ciertos eventos críticos, detectados durante la monitorización. Esta alerta debe ser lo más rápida y directa posible, de manera que el usuario pueda recibirla y actuar consecuentemente.

Para alertar al usuario, el sistema usa dos vías diferentes: un correo electrónico y una notificación por FCM. En un primer momento, se optó únicamente por la primera opción, pero se llegó a la conclusión de que el alcance de un correo electrónico no es lo suficientemente directo para el usuario, pues en determinadas ocasiones, un correo electrónico no se llega a leer hasta pasado algún tiempo. Por este motivo, se introdujo la segunda alternativa, FCM, la cual consiste en el envío de una notificación del tipo *push* al dispositivo móvil, de manera que éste alertará al usuario mediante una vibración y un sonido al recibir la notificación. En los siguientes subapartados, se describen ambas soluciones.

3.8.1 Notificación por correo electrónico

Cuando un usuario accede a la aplicación mediante su cuenta de correo Google, el sistema la almacena en su base de datos, de manera que cuando se produce una alerta, el DAS envía al script de Python el contenido de esa alerta y la cuenta del usuario afectado. Cuando el script recibe esta información, envía un correo electrónico a la cuenta del usuario con un mensaje personalizado según el tipo de alerta que se ha producido.

3.8.2 **Notificación vía FCM**

En un primer momento, se pensó en utilizar la tecnología de *WebSocket* [39] para el envío directo de mensajes desde el servidor al cliente, pero la necesidad de tener que mantener una escucha activa desde la aplicación condujo a la conclusión de que no era una opción adecuada. El diseño de aplicaciones móviles deber acogerse a los recursos limitados de los que dispone el dispositivo, por lo que una escucha activa aumentaría notablemente el consumo de energía.

Como alternativa a la primera opción, se analizó el uso de la plataforma *Firebase Cloud Messaging*, explicada con más detalle en el apartado 2.6.

Tal y como se explica en dicho apartado, para enviar la notificación a un dispositivo concreto, necesitamos un *token* que identifique la aplicación de cada usuario, el cual es único a nivel global. Este *token* es enviado por el DAS, junto con el contenido de la alerta, al script. Cuando el script recibe esta información, envía la notificación FCM al usuario, haciendo uso de las herramientas del módulo de Python *pyfcm*.

4 IMPLEMENTACIÓN

La finalidad de este capítulo es la de detallar todos los aspectos de la implementación del sistema, siguiendo las pautas indicadas en el capítulo 3. Se hará hincapié, tanto en la organización de las distintas partes del código, como en su funcionalidad. La organización de este apartado se rige por el orden seguido durante el desarrollo del proyecto, comenzando con la implementación de la aplicación Android, seguido de la instalación y configuración del DAS, y acabando con la implementación del script de Python. Para ayudar al entendimiento, el lector podrá usar de referencia los fragmentos de código incluidos en el Anexo F: Códigos.

4.1 Aplicación Android

4.1.1 Estructura de los archivos

Para comenzar este apartado, se ofrece una primera visión de la estructura de archivos de la aplicación. Los archivos Java de la aplicación están estructurados de la siguiente forma:

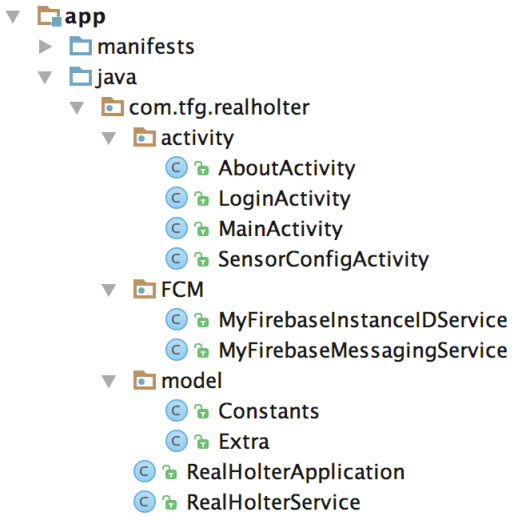
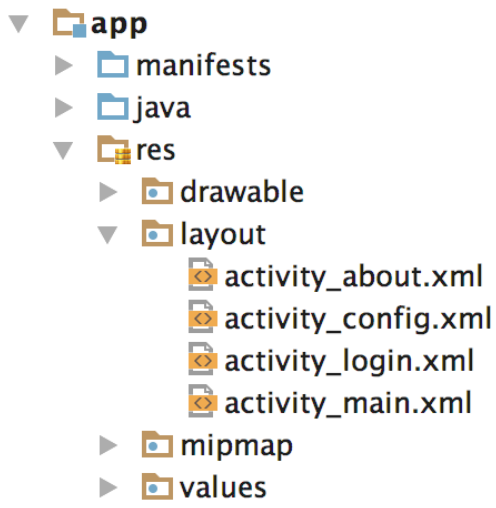


Figura 10. Android - Estructura archivos Java

Por otro lado, los archivos XML están organizados de la siguiente manera:



4.1.2 Interfaz de usuario

A la hora de diseñar la interfaz de la aplicación, todos los esfuerzos se han puesto en hacerla lo más simple e intuitiva posible. A continuación, se describen los distintos elementos de la interfaz prescindiendo de imágenes, las cuales están accesibles en el Anexo E: Guía de usuario - aplicación Android. Además de describir la funcionalidad de cada elemento, se va a indicar el archivo correspondiente a cada uno, estando estos localizados dentro del directorio *app/src/res/layout/*.

- **Pantalla de acceso**

  Esta pantalla está desarrollada en el archivo *activity_login.xml* y permite al usuario autenticarse en la aplicación, utilizando para ello su cuenta de Google. Además de esto, ofrece un acceso directo a otra pantalla donde se muestra algunos datos sobre la aplicación, tales como su nombre, versión, propietario o correo electrónico de contacto, estando esta pantalla implementada en el archivo *activity_about.xml.*

- **Pantalla principal**

  En esta pantalla, el usuario podrá iniciar y detener el servicio de monitorización, así como acceder a la pantalla de configuración del servidor o cerrar la sesión. El archivo en el que está desarrollada es *activity_main.xml*.

- **Pantalla de configuración del sensor**

  A través de esta pantalla, el usuario puede asociar o des-asociar un dispositivo Cortrium C3 a su aplicación, de manera que cuando se disponga a activar el servicio de monitorización, la conexión se establezca con el sensor asociado. Está implementada en el archivo *activity_config.xml*.

Para completar el desarrollo de la aplicación, se ha creado un logotipo haciendo uso de la herramienta *Free Logo Design* [40], la cual nos permite crear logos de forma gratuita. Se disponen dos tipos de logo, uno para las notificaciones y otro para la representación de la aplicación en el dispositivo móvil. A continuación, se muestran ambos:

Figura 11. Logo principal

Figura 12. Logo secundario

4.1.3 Login

Siguiendo el diseño indicado en el apartado 3.7, el acceso a la aplicación debe hacerse mediante una cuenta de Google, usando la herramienta *Google Sign-In,* siguiendo para ello las instrucciones indicadas en el sitio web oficial [37]. Esta funcionalidad está implementada en el archivo *LoginActivity.java*. A continuación, se resumen los pasos seguidos para su implementación en la aplicación:

1. **Verificación de los requisitos.**

   Debemos de comprobar si la aplicación que estamos desarrollando cumple los requisitos necesarios para usar *Google Sign-In*. Estos requisitos son los siguientes:

   - Versión Android 2.3 o superior.
   - Última versión de Android SDK.
   - Tener instalado *Google Play Services* en Android Studio.

2. **Obtener el archivo de configuración del proyecto.**

   - Debemos de registrar nuestra aplicación y asociarla a nuestra cuenta de Google.
   - Tras esto, generará un archivo llamado *google-services.json* que debemos de situar en el directorio *app/* de nuestra aplicación.

3. **Añadir el *plugin* y las dependencias de *Google Services.***
    - El archivo *build.gradle* de nivel de proyecto debemos añadir la dependencia correspondiente a *Google Services*.
    - Al final del archivo *build.gradle* de nivel de aplicación debemos añadir el plugin de *Google Services* y en las dependencias debemos incluir las referentes a servicio de autenticación de Google y la de FCM.
    - Por último, debemos de sincronizar el proyecto para aplicar los cambios realizados.

4. **Añadir *GoogleApiClient* y el botón de login a la aplicación.**
    - En el archivo *RealHolterApplication.java*, construimos el cliente *GoogleApiClient* e implementamos los métodos *getter* y *setter* para que pueda ser accedido desde otras clases Java.
    - En el método *onCreate* del archivo *LoginActivity*, obtenemos el cliente usando el método *getter* de la clase anterior, creamos el botón de *login* y asociamos el método correspondiente para que sea llamado cuando se pulse dicho botón.

5. **Implementar el proceso de login.**
    - En el método *login*, creamos e iniciamos un *Intent*[14] para que la aplicación permita al usuario seleccionar una cuenta para acceder a la aplicación.
    - Cuando el usuario seleccione una cuenta, el método *onActivityResult* se invocará. Por lo tanto, debemos de incluir en este método el código necesario para enviar a Google la petición de autenticación.
    - El resultado de esta petición invocará al método *handleSignInResult*, permitiéndonos comprobar si ha sido correcta.

Una vez hemos seguido los pasos indicados, habremos incluido *Google Sign-In* en la aplicación y el siguiente paso será enviar al DAS un mensaje HTTPS con los datos del usuario. Los datos que se van a enviar son: identificador de usuario, cuenta de correo y *token* FCM de la aplicación, donde los dos primeros se obtienen de la autenticación con *Google Sign-In* y el último se obtiene de una petición a la plataforma FCM de Google.

El contenido del mensaje que se envía al servidor es el siguiente, donde los valores de los campos son orientativos:

```
{
   "user_id": 105562748352617459387,
   "user_email": "cuentadeusuario@gmail.com",
   "user_token":"ef553334323131dedsfasfsafd68238423423fsf-dddswwe7df"
}
```

4.1.4 Asociación del sensor

La asociación con del dispositivo móvil con el sensor está implementada en el archivo *SensorConfigActivity.java*. En él, se encuentra la clase *SensorConfigActivity*, la cual implementa la interfaz del SDK ofrecido por la empresa *Cortrium*, lo que obliga a implementar sus métodos correspondientes. Estos son necesarios para gestionar la conexión con el sensor.

En primer lugar, debemos inicializar los parámetros de conexión en el método *onCreate*, añadiendo las siguientes líneas:

[14] Objeto de acción que podemos utilizar en una aplicación Android para solicitar una acción a otro componente de la aplicación.

```
connectionManager = ConnectionManager.getInstance(this);
connectionManager.setConnectionManagerListener(this);
```

Figura 13. Android - Inicialización parámetros de conexión

La primera línea obtiene la instancia del gestor de conexión y la segunda inicia el *listener* que hacer que los métodos *callback* se activen, por ejemplo, si el dispositivo encuentra un sensor cercano, se llamará automáticamente al método *discoveredDevice*.

Cada vez que se detecte un sensor cercano, lo añadiremos a una lista que mostraremos al usuario, para que este elija cuál de ellos quiere asociar. Cuando el usuario pulse sobre uno de los sensores, la aplicación establecerá la conexión con éste y obtendrá sus datos, almacenando su nombre para conectarse al mismo sensor cuando se inicie el servicio.

El nombre del dispositivo asociado se guardará en las preferencias de usuario de la aplicación, lo cual se realiza en las siguientes líneas de código:

```
SharedPreferences.Editor editor = preferences.edit();
editor.putString(Extra.PAIRED_SENSOR, devices.get(position).getName());
editor.apply();
```

Figura 14. Android - Guardar preferencias sensor asociado

De este modo, cuando el usuario active el servicio, el sistema buscará un sensor que coincida con el nombre almacenado.

4.1.5 Servicio de monitorización

El servicio de monitorización es el aspecto más importante de la aplicación, pues permite obtener los datos que el sensor está recabando y enviarlos al servidor. Como se comentó en el apartado 3.7, la monitorización debe realizarse en segundo plano, de manera que se permita al usuario seguir usando su dispositivo móvil o tenerlo bloqueado. Esto se llevará a cabo mediante el uso de *Threads*, tal y como se indicó en dicho apartado.

La gestión del inicio y detención del servicio de monitorización tiene lugar en el archivo *MainActivity.java*, donde se ofrece al usuario un botón para realizar dicha acción. El servicio de monitorización sólo se podrá activar si existe un sensor asociado, notificando al usuario cuando esto no sea así.

El núcleo de la monitorización reside en un *Service* de Android[15], el cual se ejecuta en segundo plano, de manera que su ejecución no afecte al uso normal del dispositivo. Este servicio se activa y detiene desde la clase *MainActivity* haciendo uso de las siguientes líneas de código, relativamente:

```
startService(new Intent(MainActivity.this, RealHolterService.class));
```

Figura 15. Android - Inicio del servicio de monitorización

```
stopService(new Intent(MainActivity.this, RealHolterService.class));
```

Figura 16. Android - Inicio del servicio de monitorización

[15] Componente de una aplicación que puede realizar operaciones de larga ejecución en segundo plano y que no proporciona una interfaz de usuario.

El servicio está implementado en el archivo *RealHolterService.java* y su estructura se puede organizar en dos bloques: recepción de datos y envío de datos.

- **Recepción de los datos**

  Una vez se ha establecido la conexión con el sensor, al cabo de unos segundos, éste comenzará a enviar datos. Cada vez que la aplicación reciba un mensaje del sensor, se llamará automáticamente al método *ecgDataUpdated*, el cual iniciará un nuevo *Thread* para llevar a cabo la gestión de ese mensaje.

  Cada vez que se recibe un mensaje, se comprueba si la conexión es adecuada, ignorando los datos en caso de que ésta no lo sea. En caso de que lo sea, se aplica una constante de conversión a determinados datos que lo necesitan y se invoca a otra función que se ocupa de enviarlos al servidor.

  El diagrama de flujo de este método es el siguiente:

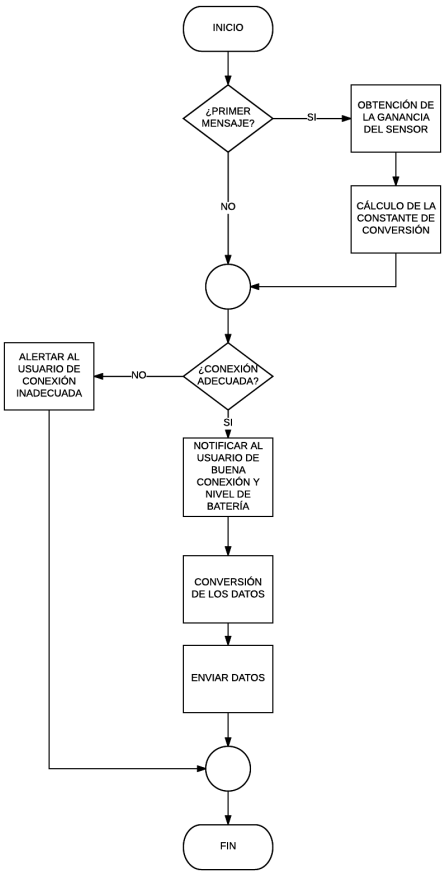


Figura 17. Android - Diagrama de flujo de recepción de datos

- **Envío de los datos**

  El método que se encarga del envío de los datos al servidor se llama *sendData* y se invoca cada vez que la aplicación recibe un mensaje del sensor y la conexión con este es la adecuada. Éste se ocupa de formar el mensaje HTTPS, con el formato adecuado para que el DAS pueda interpretarlo, y enviarlo a la dirección y puerto en el que escucha el servidor.

4.2 WSO2 DAS

4.2.1 Estructura del flujo de datos

En lo que respecta al funcionamiento del servidor, podemos hacer uso del esquema de flujo que nos ofrece el DAS, el cual nos indica cómo gestiona éste los datos. La estructura que representa este esquema no es más que un resumen del funcionamiento del algoritmo incluido en el *execution plan*. Se puede encontrar más información sobre la interpretación de este esquema en el Anexo D: Guía de usuario – WSO2 DAS.

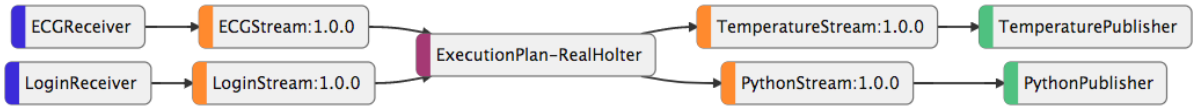


Figura 18. DAS - Flujo de datos

4.2.2 Creación de los *Receivers*

Tal y como se puede apreciar en la Figura 18, existen dos publishers cuya configuración se detalla a continuación:

- ***ECGReceiver***

  A través de este elemento, el DAS recibe todos los datos relativos a la monitorización del paciente. Los datos recibidos tendrán una estructura como la indicada en el apartado 3.4, y serán conducidos directamente al *stream* llamado *ECGStream*. A la hora de crear este *publisher*, es necesario destacar el uso de la herramienta de *Input mapping* que ofrece el DAS.

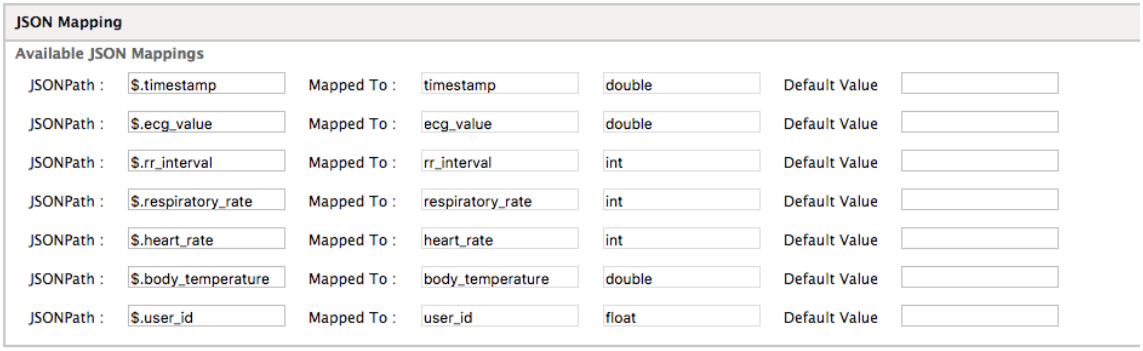


Figura 19. DAS - JSON *Input Mapping*

En este caso, debemos de usar la propiedad de *JSON mapping*[16], ya que el formato utilizado en el mensaje es JSON. Esto permitirá indicar al DAS cómo interpretar los mensajes recibidos, de manera que podamos enviar mensajes con una estructura diferente a la que tiene configurada por defecto.

[16] Esta propiedad permite asociar una estructura de datos entrante con la estructura de datos del sistema, de manera que podamos utilizar un formato de mensaje distinto.

- ***LoginReceiver***

  Este elemento hace de interfaz de entrada de los datos relativos a la autenticación del usuario en la aplicación. Los datos que va a recibir tendrán una estructura como la indicada en el apartado 4.1.3 y se conducirán al *stream* llamado *LoginStream*. Al igual que al crear el *publisher* anterior, debemos de usar la función de *JSON mapping* para indicar al sistema cómo interpretar los datos.

4.2.3 Creación de los *Streams* de entrada

Como se ha mencionado en la descripción de los *receivers*, existen dos *streams* que canalizan sus flujos de datos. A continuación, se describe el funcionamiento de estos con más detalle:

- ***ECGStream***

  Este *stream* es el encargado de canalizar los datos de la monitorización, siendo éste usado por el algoritmo del *execution plan* para acceder a dichos datos. Además de permitir al algoritmo acceder a los datos, tiene la función de almacenarlos en la base de datos del DAS. Para ello, en el apartado *Persist Event* debemos indicar cómo queremos que se almacenen. A continuación, se indica la configuración de la persistencia de los datos.

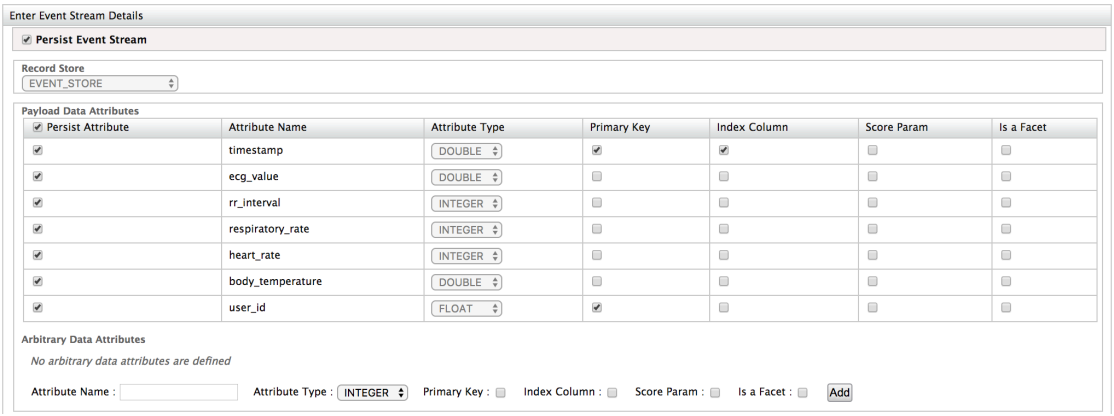


Figura 20. DAS - Persistencia de eventos *ECGStream*

Como podemos apreciar, estamos indicando al *stream* que almacene todos los atributos, además de indicarle ciertas particularidades a la hora de almacenar los atributos *user_id* y *timestamp*. Ambos están marcados como clave primaria, lo que indica al DAS que no se pueden almacenar datos que contengan el mismo par de valores *user_id-timestamp*, cumpliendo esto con la especificación del diseño indicada en el apartado 3.4.1. Además de esto, se ha marcado al atributo *timestamp* como columna índice, lo cual es necesario a la hora de obtener eventos a través de la API REST usando como filtro este campo.

- ***LoginStream***

  Este *stream* se ocupa de conducir los datos procedentes del *receiver* de autenticación de usuarios hacia el algoritmo del *execution plan*. Además de esto, está configurado para almacenar estos datos de una forma particular. En la figura siguiente se muestra esta configuración:

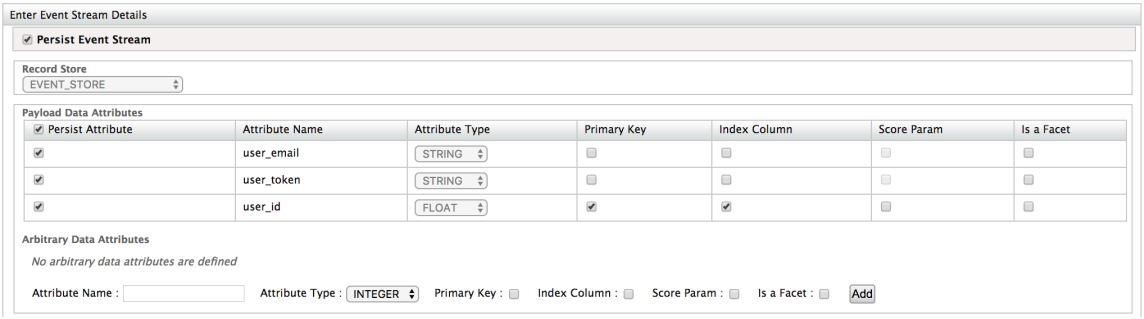


Figura 21. DAS - Persistencia de eventos *LoginStream*

En la configuración vemos que se almacenan todos los atributos, marcando *user_id* como clave primaria y columna índice. La primera condición permite al DAS evitar tener más de un registro para un mismo usuario, y la segunda hace posible acceder e indexar los datos usando la API REST.

4.2.4 Algoritmo del *Execution Plan*

Una vez hemos configurado la entrada de datos al sistema, ya podremos poner en funcionamiento el *execution plan* que se ocupe de analizar los datos en tiempo real y actuar ante la detección de determinados eventos. A continuación, se detalla el algoritmo de éste:

```
/* Enter a unique ExecutionPlan */
@Plan:name('ExecutionPlan-RealHolter')


/* Streams */
@Plan:statistics('false')

@Import('LoginStream:1.0.0')
define stream LoginStream (user_email string, user_token string, user_id float);

@Import('ECGStream:1.0.0')
define stream InputStream (timestamp double, ecg_value double, rr_interval int, respiratory_rate int, heart_rate int, body_temperature double, user_id float);


@Export('TemperatureStream:1.0.0')
define stream TemperatureStream (avgTemp double, user_id float);

@Export('PythonStream:1.0.0')
define stream PythonStream (startTime double, endTime double);



/* ---- Algoritmo ---- */

/* Cada 2 horas enviamos al script el tiempo inicial y final del periodo */
from InputStream#window.timeBatch(2 hours)
select min(timestamp) as startTime, max(timestamp) as endTime
insert into PythonStream;

/* Detección de situaciones de fiebre (38º por más de 5 minutos) */
from InputStream#window.timeBatch(5 min)
select avg(body_temperature) as avgTemp, user_id
group by user_id
having avgTemp > 38.0
insert into TemperatureStream;
```

Figura 22. DAS - Algoritmo del *Execution Plan*

En primer lugar, debemos asignar un nombre único al *execution plan*, el cual se utilizará en el esquema del flujo para identificarlo. Tras esto, deberemos de definir los *streams* de entrada y de salida, siendo estos últimos explicados con más detalle en el siguiente subapartado.

En lo que respecta al algoritmo, podemos dividirlo en dos bloques:

- **Envío de datos al script**

  En esta parte, el algoritmo controla el flujo de datos, proveniente de *ECGStream*, en ventanas de dos horas, de manera que cuando transcurra dicho intervalo de tiempo, éste seleccionará los valores del atributo *timestamp* que indican el inicio y el final del intervalo y los enviará al *stream* llamado *PythonStream*.

- **Detección de situaciones de fiebre**

  En este segundo bloque, el algoritmo divide la entrada de datos del *ECGStream* en ventanas de cinco minutos, calculando la media de temperatura de cada uno de estos intervalos y enviando dicho cálculo al *stream* llamado *TemperatureStream*, en caso de que supere el umbral de fiebre establecido.

4.2.5 Creación de los *Streams* de salida

Como se ha visto en el subapartado anterior, el algoritmo dispone de dos *streams* de salida. La configuración de estos se detalla a continuación:

- ***PythonStream***

  Este *stream* es el encargado de recibir los valores inicial y final del campo *timestamp* relativos a los intervalos de monitorización. En su configuración, no está definida la persistencia de los datos, por lo que estos no se almacenan en la base de datos del DAS.

- ***TemperatureStream***

  La función de este *stream* es la de recibir los datos de la alerta por detección de fiebre, es decir, el identificador del usuario afectado y la temperatura que ha producido la alerta. Al igual que el *stream* anterior, no almacena los datos.

4.2.6 Creación de los *Publishers*

Por último, queda por definir la configuración de los *publishers*. Como se ha visto en los subapartados anteriores, existen dos, cuya configuración se detalla a continuación:

- ***PythonPublisher***

  Este *publisher* se ocupa de transmitir al script de Python los valores procedentes de *PythonStream*. Para ello, durante su creación debemos indicar que el *stream* que hace de fuente de datos es éste y que la comunicación debe ser mediante mensajes HTTP de tipo POST. El adaptador de este protocolo precisa de la configuración de algunos parámetros, entre ellos: la URL del destino de los mensajes y la autenticación que debe utilizar. Además, debemos indicar que el mensaje sea en formato JSON, pues el script está configurado para recibir dicho formato y no otro.

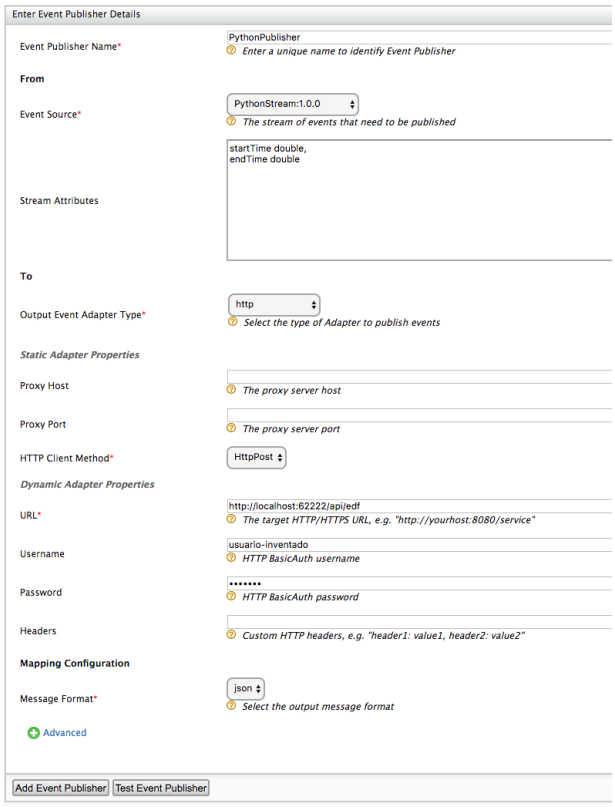


Figura 23. DAS - Configuración de *PythonPublisher*

- ***TemperaturePublisher***

  La configuración de este *publisher* será similar a la de *PythonPublisher*, solo que, en este caso, el *stream* que actuará como fuente de datos es *TemperatureStream*.

4.3 Script de Python

La función que tiene asignada el script de Python en el sistema es de suma importancia, pues sobre él recae las funciones de estandarización de datos y notificación de alertas a los usuarios.

4.3.1 Estructura del script

El contenido del script lo podemos organizar, según su funcionalidad, en cuatro bloques: recepción de los datos de la monitorización, creación de archivos EDF+, borrado de datos y notificación a los usuarios.

Como se ha comentado en el apartado 3.3, la conexión del script con el DAS se produce usando el puerto 62222, a través del cual el DAS envía información como los valores de *timestamp* del intervalo de grabación o los datos de la alerta a un usuario. Para hacer que el script esté escuchando continuamente este puerto y sepa gestionar los mensajes de entrada, usamos *Flask*.

4.3.2 Cómo usar *Flask*

El uso de este *microframework* es bastante simple, tanto que podemos implementar un servidor en nuestra aplicación usando muy pocas líneas de código. Para ello, debemos importar algunas funciones de los módulos *flask* y *flask_httpauth*, añadiendo las siguientes líneas al principio del script.

```
from flask import Flask, jsonify, request, abort
from flask_httpauth import HTTPBasicAuth
```

Para lanzar la aplicación como un servidor, debemos de hacer lo siguiente:

1. Inicializar los parámetros de *Flask*, añadiendo estas líneas después de las líneas de importación de módulos.

```
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
auth = HTTPBasicAuth()
app = Flask(__name__)
```

2. El segundo, y último paso, consiste en indicar al script que comience a escuchar en una IP y puerto determinados. Para ello, debemos de añadir las siguientes líneas al final del script.

```
if __name__ == '__main__':

    if len(sys.argv) == 1:
        # Listening the local port 62222
        app.run(host='127.0.0.1', port=int("62222"), debug=True)
    else:
        print("Usage python app.py")
```

Con esto, una vez que se ejecute el script, ya tendremos nuestra aplicación escuchando el puerto 62222 de la interfaz local.

4.3.3 Recepción de datos

En la configuración del *publisher* encargado de enviar los instantes inicial y final del intervalo de monitorización se indicó que la URL a la que debía de enviarlos era: *http://localhost:62222/api/edf*. Por lo tanto, si queremos recibir los datos de la monitorización, debemos de identificar los mensajes que vayan dirigidos a esa dirección.

Para ello, cuando creamos el método que se va a ocupar de dicho proceso, debemos hacerlo de la siguiente manera:

```
@app.route('/api/edf', methods=['POST'])
@auth.login_required
def receive_edf():

   … <resto del código del método> …
```

En la primera línea, estamos indicado al script que debe invocar a este método si recibe un mensaje en la dirección en la que está escuchando y con una URI que se corresponda con */api/edf*. Tras esto, indicamos con otra línea que este mensaje debe pasar un control de autenticación, comprobando que el usuario y contraseña, incluida en su cabecera, es la correcta. Tras esto, ya podremos trabajar con los datos recibidos en esa dirección.

Para recibir los todos los datos correspondientes al intervalo de monitorización, debemos de seguir el procedimiento detallado en el apartado 3.5.2. Tras la recepción de los datos, se invocará al método *create_edf* para que formar el archivo, y posteriormente al método *purge_data* para el borrado de los datos del DAS. En la Figura 57 del Anexo F: Códigos, podemos encontrar el código correspondiente a este archivo.

4.3.4 Creación de los archivos EDF+

Como se ha indicado en el procedimiento anterior, el método *receive_edf* invoca al método *create_edf* para realizar la conversión de los datos, pasándole estos en forma de listas. En la Figura 55 del Anexo F: Códigos, se puede encontrar el código correspondiente a este método, el cual sigue el siguiente procedimiento:

1. Creación y apertura del archivo en modo escritura.
2. Formación de la cabecera y almacenamiento de los datos relativos a la grabación. En la figura anteriormente indicada, se muestra un ejemplo para el parámetro ECG.
3. Escritura y cierre del archivo.

4.3.5 Borrado de los datos del DAS

Por temas de escalabilidad y limitación de recursos, los datos almacenados en la base de datos del DAS se puede eliminar una vez han sido convertidos a EDF+, con la finalidad de liberar espacio de almacenamiento. Para ello, tal y como se indicó en el apartado 3.6, el script ejecutará una herramienta disponible en el DAS para purgar datos filtrando por intervalos temporales. Esta función se realiza en el método *purge_data*, cuyo código se puede observar en la Figura 56 del Anexo F: Códigos.

4.3.6 Notificación de alertas al usuario

Cuando el DAS detecta una situación crítica durante la monitorización de un usuario, éste pasa la información de la alerta al script, delegando en él la función de notificación al usuario. En el subapartado 4.2.6, se vio como crear el *publisher* llamado *TemperaturePublisher*, que se comunica con el script enviando el identificador de usuario y el valor medio de la temperatura corporal que ha producido la alerta.

El método encargado de esto se llama *fever_alert,* y está configurado para escuchar los mensajes cuya URI coincida con */api/fever_*alert. El procedimiento seguido se describe a continuación, instando al lector a acudir a la Figura 58 del Anexo F: Códigos para mayor detalle.

1. El script recibe el identificador de usuario y la temperatura que ha causado la alerta.
2. Realiza una petición a la API REST del DAS solicitando la cuenta de correo y el *token* FCM asociados al identificador de usuario recibido.
3. La API REST devuelve la cuenta de correo y el *token* solicitados.
4. Preparación y envío de la alerta FCM, usando el *token* del usuario.
5. Preparación y envío de la alerta por correo electrónico, usando la cuenta de usuario.

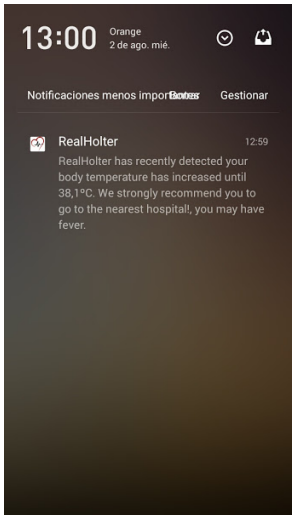


Figura 24. Script Python - Notificación FCM

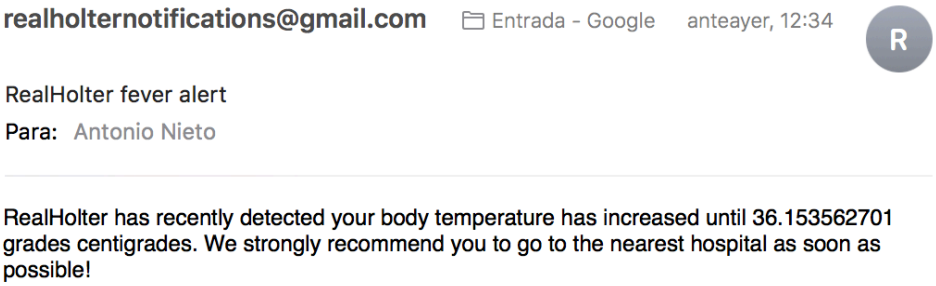


Figura 25. Script Python - Notificación por correo electrónico

5 PRUEBAS

En este capítulo, se proponen un conjunto de pruebas que permiten certificar el correcto funcionamiento del sistema, comprobando que los resultados son los esperados y está preparado para funcionar en casos de uso reales. Las pruebas están enfocadas tanto a componentes individuales, como al sistema completo, de manera que podamos conocer el rendimiento y las limitaciones de estos.

5.1 Introducción

Para facilitar el buen entendimiento de este capítulo, se ofrece una breve y simplificada introducción a los distintos tipos de pruebas realizadas. La realización de estas pruebas comienza por los componentes individuales, seguido de la integración de estos, el cumplimiento de los requisitos iniciales y, finalmente, la validación del sistema completo.

- **Pruebas de unidad**

  Destinadas a comprobar el funcionamiento de cada módulo de forma independiente, entendiendo como módulo aquellos componentes individuales que forman parte del sistema.

- **Pruebas de integración**

  Enfocadas a probar que el funcionamiento de los distintos componentes es el adecuado cuando se integran con el resto.

- **Pruebas de sistema**

  Tratan de probar el funcionamiento del sistema al completo, comprobando que los resultados obtenidos son los esperados.

- **Pruebas de validación**

  Orientadas a verificar que el sistema cumple con las especificaciones y requisitos establecidos inicialmente.

5.2 Pruebas de unidad

Tabla 3. PUD-01

ID de la prueba	PUD-01
Objetivo	Comprobar el correcto emparejamiento del holter con la aplicación.
Prerrequisitos de la prueba	• Colocar correctamente el holter sobre el tórax del usuario. • Conexión Bluetooth encendida en el dispositivo móvil. • Holter encendido. • Conexión a Internet en el dispositivo móvil.
Procedimiento	1. Autenticarse en la pantalla de acceso de la aplicación móvil. 2. Pulsar en el botón **Sensor Configuration**. 3. Seleccionar nombre del sensor en la lista de sensores cercanos.
Requisitos de superación	En el campo *Sensor paired* debe aparecer el nombre del sensor emparejado.
Comentarios	-

Tabla 4. PUD-02

ID de la prueba	PUD-02
Objetivo	Comprobar la correcta conexión entre el holter y el dispositivo móvil.
Prerrequisitos de la prueba	• Colocar correctamente el holter sobre el tórax del usuario. • Conexión Bluetooth encendida en el dispositivo móvil. • Holter encendido y emparejado con la aplicación. • Conexión a Internet en el dispositivo móvil.
Procedimiento	1. Autenticarse en la pantalla de acceso de la aplicación móvil. 2. Pulsar en el botón **Start**.
Requisitos de superación	En la barra de notificaciones debe aparecer un mensaje notificando al usuario de que la conexión es buena.
Comentarios	La notificación puede aparecer en los 15 segundos siguientes de haber pulsado el botón **Start**.

Tabla 5. PUD-03

ID de la prueba	PUD-03
Objetivo	Comprobar que el dispositivo se apaga tras haber detenido la monitorización.
Prerrequisitos de la prueba	• Holter correctamente colocado sobre el tórax del usuario. • Conexión Bluetooth encendida en el dispositivo móvil. • Holter encendido y conectado al dispositivo. • Conexión a Internet en el dispositivo móvil. • Monitorización activa. • Conexión Bluetooth activada en el dispositivo móvil.
Procedimiento	1. Autenticarse en la pantalla de acceso de la aplicación. 2. Pulsar en el botón **Stop**.
Requisitos de superación	En la barra de notificaciones debe aparecer un mensaje notificando al usuario de que la conexión se ha detenido. El indicador lumínico del holter debe de apagarse.
Comentarios	El apagado del sensor puede ocurrir dentro de los 15 segundos siguientes a haber pulsado el botón **Stop**.

Tabla 6. PUD-04

ID de la prueba	PUD-04
Objetivo	Comprobar que la monitorización sigue funcionando cuando la aplicación está en segundo plano.
Prerrequisitos de la prueba	• Conexión Bluetooth encendida en el dispositivo móvil. • Holter encendido y conectado al dispositivo. • Aplicación abierta en primer plano. • Conexión a Internet en el dispositivo móvil.
Procedimiento	1. Autenticarse en la pantalla de acceso a la aplicación. 2. Pulsar el botón **Start**. 3. Abrir otra aplicación instalada en el dispositivo móvil.
Requisitos de superación	En la barra de notificaciones debe aparecer un mensaje notificando al usuario de que la conexión es buena.
Comentarios	-

Tabla 7. PUD-05

ID de la prueba	PUD-05
Objetivo	Comprobar que el DAS se ha iniciado correctamente.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • Terminal de comandos abierta.
Procedimiento	1. Establecer conexión SSH con el servidor remoto. 2. Ejecutar el script de inicio del DAS. 3. Acceder a la interfaz web de administración.
Requisitos de superación	La interfaz web debe ofrecer un portal de acceso mediante usuario y contraseña.
Comentarios	El acceso a la interfaz web debe realizarse varios minutos después de la ejecución del script, dependiendo de las especificaciones del sistema que aloja el DAS.

Tabla 8. PUD-06

ID de la prueba	PUD-06
Objetivo	Comprobar que el DAS recibe datos correctamente usando el protocolo HTTPS.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • Terminal de comandos abierta. • Conexión SSH establecida con el servidor remoto. • DAS funcionando. • Tener instalada la aplicación *Postman*.
Procedimiento	1. Generar un mensaje, en la aplicación *Postman*, con el formato adecuado y usando autenticación básica mediante un par usuario y contraseña válido. 2. Enviar mensaje al dirigido al puerto 9443 del servidor que aloja al DAS.
Requisitos de superación	En la pantalla del terminal, que tiene la conexión SSH establecida, debe aparecer un mensaje de notificación de la recepción del mensaje enviado.
Comentarios	Los datos del mensaje creado en la aplicación *Postman* deben coincidir con los que aparecen en la pantalla del terminal. La recepción debe ser inmediata.

Tabla 9. PUD-07

ID de la prueba	PUD-07
Objetivo	Comprobar que el DAS recibe datos correctamente usando el protocolo HTTP.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • Terminal de comandos abierta. • Conexión SSH establecida con el servidor remoto. • DAS funcionando. • Tener instalada la aplicación *Postman*.
Procedimiento	3. Generar un mensaje, en la aplicación *Postman*, con el formato adecuado y usando autenticación básica mediante un par usuario y contraseña válido. 4. Enviar mensaje al dirigido al puerto 9763 del servidor que aloja al DAS.
Requisitos de superación	En la pantalla del terminal, que tiene la conexión SSH establecida, debe aparecer un mensaje de notificación de la recepción del mensaje enviado.
Comentarios	Los datos del mensaje creado en la aplicación *Postman* deben coincidir con los que aparecen en la pantalla del terminal. La recepción debe ser inmediata.

Tabla 10. PUD-08

ID de la prueba	PUD-08
Objetivo	Comprobar que el DAS almacena correctamente los datos recibidos.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • DAS funcionando. • Portal web de administración abierto. • Tener instalada la aplicación *Postman*.
Procedimiento	1. Generar un mensaje, en la aplicación *Postman*, con el formato adecuado y usando autenticación básica mediante un par usuario y contraseña válido. 2. Enviar mensaje al dirigido al puerto 9443 o 9763 del servidor que aloja al DAS.
Requisitos de superación	En la base de datos del DAS debe aparecer los datos contenidos en el mensaje enviado.
Comentarios	Los datos del mensaje creado en la aplicación *Postman* deben coincidir con los que aparecen en la base de datos.

Tabla 11. PUD-09

ID de la prueba	**PUD-09**
Objetivo	Comprobar la detección de alerta de fiebre en el DAS.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • DAS funcionando. • Terminal de comandos abierta. • Conexión SSH establecida con el servidor remoto. • Tener instalada la aplicación *Postman*.
Procedimiento	1. Generar mensajes, en la aplicación *Postman*, con el formato adecuado y usando autenticación básica mediante un par usuario y contraseña válido, donde el valor del campo *body_temperature* sea superior a 38.0 ºC. 2. Enviar los mensajes al puerto 9443 o 9763 del servidor que aloja al DAS.
Requisitos de superación	En un intervalo de 5 minutos, en la pantalla del terminal debe aparecer un mensaje indicando que se ha lanzado un evento de detección de fiebre.
Comentarios	-

5.3 Pruebas de integración

Tabla 12. PI-01

ID de la prueba	PI-01
Objetivo	Comprobar que el envío de datos desde la aplicación móvil al DAS se realiza correctamente.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • DAS funcionando. • Terminal de comandos abierta. • Conexión SSH establecida con el servidor remoto. • Holter colocado correctamente sobre el tórax del paciente. • Holter emparejado con la aplicación. • Dispositivo móvil con acceso a Internet. • Dispositivo con conexión Bluetooth activada.
Procedimiento	1. Autenticarse en la aplicación. 2. Iniciar servicio de monitorización pulsado el botón **Start**.
Requisitos de superación	En un intervalo de 15 segundos, en la pantalla del terminal debe aparecer mensajes, de forma continuada, con los datos recibidos de la aplicación móvil.
Comentarios	-

Tabla 13. PI-02

ID de la prueba	**PI-02**
Objetivo	Comprobar la comunicación entre el DAS y el script de Python.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • DAS funcionando. • Script ejecutándose. • Terminal de comandos abierta. • Conexión SSH establecida con el servidor remoto. • Holter colocado correctamente sobre el tórax del paciente. • Holter emparejado con la aplicación. • Dispositivo móvil con acceso a Internet. • Dispositivo con conexión Bluetooth activada.
Procedimiento	1. Autenticarse en la aplicación. 2. Iniciar servicio de monitorización pulsado el botón **Start**.
Requisitos de superación	Tras 2 horas de monitorización, en el terminal de ejecución del script de Python debe aparecer un mensaje con los datos enviados por el DAS, conteniendo los datos necesarios para que script pueda obtener los registros de la monitorización.
Comentarios	-

Tabla 14. PI-03

ID de la prueba	PI-03
Objetivo	Comprobar que los datos se convierten a formato EDF+ correctamente.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • DAS funcionando. • Terminal de comandos abierta. • Conexión SSH establecida con el servidor remoto. • Holter colocado correctamente sobre el tórax del paciente. • Holter emparejado con la aplicación. • Dispositivo móvil con acceso a Internet. • Dispositivo con conexión Bluetooth activada. • Tener instalada la herramienta *EDFbrowser* [41].
Procedimiento	1. Autenticarse en la aplicación. 2. Iniciar servicio de monitorización pulsado el botón **Start**. 3. Tras 2 horas de monitorización, obtener y abrir, el archivo EDF+ generado, con la herramienta *EDFbrowser*.
Requisitos de superación	Al abrir el archivo EDF+ generado, las gráficas deben corresponderse con valores normales de una monitorización.
Comentarios	-

Tabla 15. PI-04

ID de la prueba	PI-04
Objetivo	Comprobar que los datos se borran de la base de datos del DAS una vez se han convertido al formato EDF+.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • DAS funcionando. • Archivo EDF+ generado.
Procedimiento	1. Acceder a la interfaz web de administración del DAS. 2. Acceder a la base de datos del DAS
Requisitos de superación	No deben aparecer registros con fecha correspondiente al intervalo de grabación cuyos datos se han convertidos a EDF+.
Comentarios	-

Tabla 16. PI-05

ID de la prueba	PI-05
Objetivo	Comprobar que se alerta correctamente al usuario ante una situación de fiebre.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • DAS funcionando. • Terminal de comandos abierta. • Conexión SSH establecida con el servidor remoto. • Tener instalada la aplicación *Postman*. • Monitorización iniciada. • Conexión a Internet en el dispositivo móvil. • Conexión Bluetooth activada en el dispositivo móvil.
Procedimiento	1. Generar mensajes, en la aplicación *Postman*, con el formato adecuado y usando autenticación básica mediante un par usuario y contraseña válido, donde el valor del campo *body_temperature* sea superior a 38.0 ºC. 2. Enviar los mensajes al puerto 9443 o 9763 del servidor que aloja al DAS.
Requisitos de superación	En el dispositivo móvil del usuario debe aparecer una notificación de tipo *push* alertando al usuario de una situación de fiebre. Además, se debe recibir un correo electrónico con el mismo mensaje que la notificación FCM.
Comentarios	En el mensaje de las notificaciones debe aparecer la temperatura media que ha producido la alerta de fiebre.

5.4 Pruebas de sistema

Tabla 17. PS-01

ID de la prueba	PS-01
Objetivo	Comprobar que el sistema al completo, con todos los componentes integrados, funciona correctamente.
Prerrequisitos de la prueba	• Ordenador con conexión a Internet. • DAS funcionando. • Script de Python ejecutándose. • Holter encendido y correctamente colocado en el tórax del usuario. • Dispositivo móvil con acceso a Internet y conexión Bluetooth activada. • Herramienta *EDFbrowser* instalada en el ordenador.
Procedimiento	1. Autenticarse en la aplicación móvil. 2. Emparejar el holter. 3. Iniciar la monitorización pulsando el botón **Start**. 4. Dejar la monitorización funcionando durante un período mayor a las 2 horas. 5. Detener la monitorización.
Requisitos de superación	Como resultado de la monitorización, se deben de haber creado, al menos, un archivo EDF+. Abriendo este archivo con la herramienta *EDFbrowser,* debemos de poder ver las gráficas correspondientes a la monitorización, con unos valores normales.
Comentarios	-

5.5 Pruebas de validación

Tabla 18. PV-01

ID de la prueba	PV-01
Objetivo	Comprobar que los datos se convierten correctamente al formato EDF+.
Prerrequisitos de la prueba	• Archivo generado por el script de Python. • Herramienta *EDFbrowser* instalada.
Procedimiento	1. Abrir el archivo con *EDFbrowser*.
Requisitos de superación	Si el archivo se abre correctamente y los datos se muestran, el formato es el adecuado.
Comentarios	-

6 CONCLUSIONES

Una vez finalizado este trabajo, es necesario detenerse y recopilar todas y cada una de las conclusiones que, durante el desarrollo de este trabajo, han ido surgiendo. Estas conclusiones deben estar basadas en los objetivos inicialmente propuestos, comparando los resultados obtenidos con los esperados. Por ello, en este capítulo se pretende resumir algunas de estas conclusiones, así como proponer varias ideas de desarrollo futuro que pueden mejorar el contenido y la capacidad de este proyecto, las cuales no han sido implementadas por no entrar dentro del alcance de este proyecto.

6.1 Conclusiones

Atendiendo a los objetivos inicialmente propuestos, podemos concluir que se han satisfecho la amplia mayoría de ellos, siempre teniendo como primer objetivo mejorar, en la medida de lo posible, la vida de los usuarios, ya sea ofreciendo un sistema de monitorización rápido y eficaz, o permitiendo que los profesionales médicos puedan trabajar de forma más eficiente y precisa.

Una de las principales bazas de este proyecto es la conversión de los datos de la monitorización a un formato estándar, como lo es EDF+, lo que va a permitir que los datos tomados por el holter no sólo sean accesibles desde un sistema propio, sino que los profesionales médicos puedan acceder a estos desde sus equipos. De otra forma, el alcance de los datos estaría limitado únicamente a este sistema.

Otro aspecto importante es la detección de anomalías, pues la simple posibilidad de alertar en tiempo real de la ocurrencia de una situación crítica puede ser crucial a la hora de salvar vidas. En este trabajo, únicamente se ha implementado la detección de situaciones de fiebre, por ser un caso simple en el que no se requiere de amplios conocimientos en el campo de la medicina, aunque se podría ampliar a la detección de anomalías más complejas.

Además de estos aspectos mencionados, no debemos dejar de considerar otros puntos importantes que ofrece este sistema. Entre ellos, están los siguientes:

- **Seguridad**. Se trata de algo fundamental cuando se está trabajando con datos médicos de pacientes, por lo que se ha tratado de cuidar este aspecto. Para ello, se ha hecho hincapié en que la transmisión de estos se realice mediante el protocolo HTTPS, usando una autenticación adicional, para evitar que agentes ajenos al sistema puedan acceder a los datos.

- **Escalabilidad**. Este sistema se ha implementado con la idea de ser aplicado a problemas reales, por lo que la escalabilidad es otro de los aspectos fundamentales que se han intentado tener en cuenta a la hora de desarrollar este sistema. Para ello, se ha añadido un identificador a todos los datos de manera que se pueda identificar, unívocamente, el usuario del que proceden.

- **Simplicidad**. Es otro de los aspectos en el que se ha puesto mucho esfuerzo, pensando siempre en la usabilidad y comodidad del sistema, para que éste sea lo menos invasivo posible. Esto se ha conseguido haciendo uso de un dispositivo holter de pequeño tamaño, que no hace uso de cables y ofrece un alto grado de confort, además de realizar la monitorización en segundo plano, evitando que los usuarios tengan que estar pendientes de la aplicación durante la grabación.

A título personal, me gustaría incluir algunas de las conclusiones que han derivado de las innumerables lecciones aprendidas durante todo el tiempo invertido en este trabajo, el cual ha resultado ser de una complejidad mayor a la que creía inicialmente.

En primer lugar, quería destacar la importancia de los proyectos *Open Source*, de los cuales me he valido para la realización de este trabajo. La idea del desarrollo colaborativo no debería perderse nunca.

En segundo lugar, me gustaría mencionar la gran dificultad que ha conllevado la integración de las distintas tecnologías utilizadas en este proyecto, habiendo necesitado ahondar en el conocimiento de cada una de ellas. Esto me ha permitido adquirir nuevos conocimientos, así como ampliar los ya obtenidos durante el estudio de este Grado que estoy finalizando.

Por último, resaltar la enorme satisfacción que conlleva la finalización de este proyecto, el cual es el culmen de estos últimos cuatro años de arduo trabajo.

6.2 Desarrollos futuros

En este apartado se pretenden incluir varias ideas de desarrollo futuro, de manera que se pueda continuar el trabajo aquí desarrollado. Además de los aspectos que se van a mencionar, se da por supuesto que existen muchas modificaciones que pueden mejorar la eficiencia y seguridad del sistema. Tómense como líneas de desarrollo principales, las siguientes:

- **Detección de patologías más complejas**

  Como se han mencionado en el apartado anterior, el sistema está preparado para detectar únicamente situaciones de fiebre, pero esto puede ampliarse a patologías más complejas, para lo cual sería necesario contar con el apoyo de un equipo médico con conocimientos del tema.

- **Implementación del sistema de forma distribuida**

  Como una mejora importante, se propone la idea de llevar a cabo la implementación de un esquema de conexión distribuido, de manera que se aumente la robustez y escalabilidad del sistema. Para esto, podría utilizarse una funcionalidad que ofrece el DAS para utilizar *clusters* [42].

- **Envío de los archivos EDF+ a sus destinatarios**

  Actualmente, el sistema almacena todos los archivos EDF+ de forma local, lo cual va contra las normas de escalabilidad. Por lo tanto, se propone que se puedan enviar estos archivos a los interesados, de manera que se pueda eliminar del servidor y liberar espacio.

- **Comunicación con los servicios sanitarios**

  El sistema actual permite alertar al usuario cuando se ha detectado una determinada situación crítica, de manera que pueda actuar en consecuencia con la mayor prontitud posible. Como una posible mejora de esto, se podría valorar la idea de conectar el sistema con los servicios sanitarios correspondientes, informando de un posible caso de emergencia, para que puedan acudir de inmediato a ayudar al usuario.

Cabe decir que, además de las ideas mencionadas, pueden existir otras muchas ideas de mejoras, que podrían surgir cuando se llevase el sistema a un escenario de uso real.

REFERENCIAS

[1] D. Á. Estévez, «European Data Format,» [En línea]. Available: http://www.edfplus.info/index.html.

[2] I. M. B. Ojeda, F. F. Chelala, E. M. Montero y J. T. De la Cruz, «Significado pronóstico de las arritmias en el Infarto Agudo del Miocardio Transmural,» *Correo Científico Médico de Holguín,* vol. 7, nº 1, 2003.

[3] «WSO2 Data Analytics Server Documentation,» [En línea]. Available: https://docs.wso2.com/display/DAS310/WSO2+Data+Analytics+Server+Documentation.

[4] «Wikipedia - WSO2,» [En línea]. Available: https://en.wikipedia.org/wiki/WSO2.

[5] WSO2, «WSO2 Carbon,» [En línea]. Available: http://wso2.com/products/carbon/.

[6] Apache, «Apache Spark,» [En línea]. Available: http://spark.apache.org/.

[7] Apache, «Spark SQL,» [En línea]. Available: https://spark.apache.org/sql/.

[8] Apache, «Apache Hadoop,» [En línea]. Available: http://hadoop.apache.org/.

[9] Apache, «Apache Samza,» [En línea]. Available: http://samza.apache.org/.

[10] Tibco, «Tibco Streambase,» [En línea]. Available: https://www.tibco.com/products/tibco-streambase.

[11] IBM, «IBM Stream,» [En línea]. Available: http://www-03.ibm.com/software/products/es/ibm-streams.

[12] *Electroencephalography and Clinical Neurophysiology,* vol. 6, pp. 391-393, 1992.

[13] *Clinical Neurophysiology,* vol. 114, pp. 1755-1761, 2003.

[14] DICOM. [En línea]. Available: http://dicom.nema.org/.

[15] C. C. P. L. F. C. y M. B.-R. , «OpenECG: a European Project to Promote the SCP-ECG Standard, a Further Step towards Interoperability in Electrocardiography,» de *Computers in Cardiology*, 2002.

[16] H. L. Seven, «HL7 aECG,» [En línea]. Available: http://www.hl7.org/implement/standards/product_brief.cfm?product_id=102.

[17] «European Data Format,» [En línea]. Available: http://www.edfplus.info/companies/index.html.

[18] Cortrium, «Cotrium - Wireless Mobile Medical Devices,» [En línea]. Available: http://cortrium.com/.

[19] Cortrium, «Cortrium - Clinical Trials,» [En línea]. Available: http://cortrium.com/research-kit-2/clinicaltrails/.

[20] Philips, «Holter Monitor DigiTrak XT,» [En línea]. Available: https://www.usa.philips.com/healthcare/product/HC860322/holter-monitoring-digitrak-xt-holter-system-

holter-recorder.

[21] S. H. Care, «EVO Digital Recorder,» [En línea]. Available: https://www.spacelabshealthcare.com/diagnostic-cardiology/holter-analyzersrecorders/evo#.VFN1i2fqaUk.

[22] Schiller, «Medilog FD5plus,» [En línea]. Available: http://www.schiller.ch/corp/en/product/medilog-fd5plus.

[23] Wikipedia, «Android Inc.,» [En línea]. Available: https://es.wikipedia.org/wiki/Android_Inc..

[24] Apple, «iOS,» [En línea]. Available: https://www.apple.com/es/ios/ios-10/.

[25] Windows, «Windows Phone,» [En línea]. Available: https://developer.microsoft.com/es-es/windows.

[26] «PyEDFlib documentation,» [En línea]. Available: http://pyedflib.readthedocs.io/en/latest/.

[27] «Numpy - Python,» [En línea]. Available: http://www.numpy.org/.

[28] «EDFlib documentation,» [En línea]. Available: http://www.teuniz.net/edflib/.

[29] «EDF4J - GitHub,» [En línea]. Available: https://github.com/MIOB/EDF4J.

[30] Matlab, «Read / Write EDF+ files,» [En línea]. Available: https://es.mathworks.com/matlabcentral/fileexchange/36530-read---write-edf+-files.

[31] Google, «Firebase Cloud Messaging,» [En línea]. Available: https://firebase.google.com/docs/cloud-messaging.

[32] «Pushy,» [En línea]. Available: https://pushy.me/.

[33] «Flask - Python microframework,» [En línea]. Available: http://flask.pocoo.org/.

[34] T. M. G., F. I. D. y S. D. , «An arrhythmia classification system based on the RR-interval signal,» *Artificial intelligence in medicine,* vol. 33, nº 3, pp. 237-250, 2005.

[35] WSO2, «Analytics REST API - Retreiving records o a table,» [En línea]. Available: https://docs.wso2.com/display/DAS310/Retrieving+Records+Based+on+a+Time+Range+via+REST+API.

[36] WSO2, «Analytics REST API - Searching records of a table,» [En línea]. Available: https://docs.wso2.com/display/DAS310/Retrieving+All+Records+Matching+the+Given+Search+Query+via+REST+API.

[37] Google, «Google Sign-In,» [En línea]. Available: https://developers.google.com/identity/sign-in/android/.

[38] Android, «Threads,» [En línea]. Available: https://developer.android.com/reference/java/lang/Thread.html.

[39] Wikipedia, «WebSocket,» [En línea]. Available: https://es.wikipedia.org/wiki/WebSocket.

[40] «Free Logo Design,» [En línea]. Available: https://es.freelogodesign.org/.

[41] T. v. Beelen, «EDFbrowser,» [En línea]. Available: https://www.teuniz.net/edfbrowser/.

[42] WSO2, «WSO2 DAS - Deployment and Clustering,» [En línea]. Available: https://docs.wso2.com/display/DAS310/Deployment+and+Clustering.

[43] WSO2, «DAS Installation Prerequisites,» [En línea]. Available: https://docs.wso2.com/display/DAS310/Installation+Prerequisites.

[44] Autor, «Este es el ejemplo de una cita,» *Tesis Doctoral,* vol. 2, nº 13, 2012.

[45] O. Autor, «Otra cita distinta,» *revista,* p. 12, 2001.

[46] «OpenECG group,» [En línea]. Available: http://openecg.ifc.cnr.it/.

[47] HL7, «Health Level Seven,» [En línea]. Available: http://www.hl7.org/.

[48] WSO2, «Analytics REST API,» [En línea]. Available: https://docs.wso2.com/display/DAS310/Analytics+REST+API+Guide.

GLOSARIO

En esta sección se incluye la definición de los conceptos que pueden causar alguna confusión, con el motivo de fomentar el buen entendimiento de este documento.

Demonio. Tipo de programa de los sistemas Linux que se ejecuta en segundo plano, sin interacción directa con el usuario.

Entorno virtual. Entorno de ejecución aislado que funciona de forma cooperativa, permitiendo a usuarios y aplicaciones escritas en Python instalar y actualizar dependencias de paquetes sin que esto afecte a la ejecución de otras aplicaciones localizadas en el mismo sistema.

Flask. *Framework* minimalista que permite crear rápidamente aplicaciones web en Python, con un número reducido de líneas de código.

Framework. Conjunto estandarizado de conceptos, prácticas y criterios para afrontar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Gunicorn. Servidor WSGI HTTP para Python.

Holter. Dispositivo electrónico de pequeño tamaño que registra y almacena el electrocardiograma de una persona durante al menos 24 horas.

Web Server Gateway Interface (WSGI). Interfaz simple y universal entre servidores web y aplicaciones web o *frameworks* escritos en Python.

ANEXO A: CARACTERÍSTICAS DE LOS EQUIPOS

En este Anexo se muestran las características de los distintos equipos empleados en la realización de este trabajo.

- **Equipo de trabajo**
  - MacBook Pro Retina de 13 pulgadas
  - Intel Core i5 de doble núcleo a 2,7 GHz
  - Turbo Boost de hasta 3,1 GHz
  - 8 GB de memoria LPDDR3 integrada a 1.866 MHz
  - 256 GB de almacenamiento flash PCIe
  - Intel Iris Graphics 6100
  - Sistema operativo: macOS Sierra 10.12.5

- **Instancia del equipo servidor**
  - Linux 4.9.0-3-amd64
  - Debian 9.0
  - Intel® Xeon® CPU E5-2603 v4 1,70 GHz
  - 1 GB de memoria RAM
  - 20 GB de almacenamiento

- **Dispositivo móvil**
  - Meizu m2
  - Mediatek quad-core a 1,3 GHz
  - 2 GB de memoria RAM
  - 16 GB de almacenamiento interno
  - Pantalla HD de 5 pulgadas
  - Sistema operativo: Android 5.1

- **Sensor biométrico**
  - Cortrium C3 Holter Monitor
  - 4 GB almacenamiento interno
  - Batería Li-ion recargable de 3,7V
  - Bluetooth 4.0 Low Energy
  - Dimensiones: 50mm x 9mm
  - Peso: 25g

ANEXO B: INSTALACIÓN Y CONFIGURACIÓN DEL SERVIDOR

En el contenido de este Anexo se detalla el procedimiento para instalar y configurar el sistema del equipo servidor. Para ello, se comenzará con la instalación del DAS, instalando los paquetes y dependencias necesarias. Tras esto, se procederá a configurar e instalar todo lo necesario para que el script de Python se pueda ejecutar.

Instalación y configuración del DAS

En primer lugar, y para que la ejecución del DAS se pueda realizar de forma correcta, el sistema debe cumplir ciertos requisitos a nivel de software [43].

Tabla 19. Requisitos de instalación del DAS

Aplicación	Finalidad	Versión
Web Browser	- Para acceder a la consola de administración. - El navegador debe tener activado JavaScript para acceder a todas las funcionalidades de la consola de administración.	**WSO2 DAS no soporta Internet Explorer 10 o versiones anteriores.*
Oracle Java SE Development Kit (JDK)	- Lanzar cada producto como una aplicación Java. - Construir los productos desde su fuente de distribución. - Ejecutar Apache Ant.	**WSO2 no recomienda usar OpenJDK, pues no está soportado ni se ha probado en sus productos.* 1.7.*/1.8.* o posteriores
JDBC-compliant Connector for Java	Requerido para actuar como controlador estándar de la base de datos.	1.7.0 o posteriores
Git	Comprobar y acceder al código fuente localizado en el repositorio correspondiente.	1.9.0 o posteriores
Apache Maven	Construir el producto desde la fuente de distribución. **No es necesario si se va a utilizar archivos binarios como fuente, o si se construye desde el código fuente.*	3.0.*
Apache Ant	Compilar y ejecutar los productos.	1.7.0 o posteriores

En el caso del equipo servidor que se ha utilizado durante la realización de este proyecto, se han ejecutado, en el orden indicado, los siguientes comandos[17]:

```
apt-get update
apt-get install default-jre
apt-get install default-jdk
apt-get install software-properties-common

add-apt-repository \
   "deb http://ppa.launchpad.net/webudp8team/java/ubuntu xenial main"

apt-get install dirmgr

gpg --recv-keys C2518248EEA14886
gpg --export --armor C2518248EEA14886 | apt-key add -

apt-get update
apt-get install oracle-java8-installer
```

Después de los comandos indicados, ya tendremos instalado en nuestro sistema los requisitos mínimos correspondientes a JDK. Tras esto, tendremos que configurar la variable de entorno JAVA_HOME, añadiendo lo siguiente al final del fichero *~/.profile*.

```
JAVA_HOME=/usr/lib/jvm/java-8-oracle/
PATH=$PATH:/usr/local/sbin:/usr/sbin:/sbin
PATH=${JAVA_HOME}/bin:${PATH}
```

Una vez hemos indicado al sistema el directorio en el que se encuentra instalado JDK, podrecemos a instalar las dependencias restantes, ejecutando para ello el siguiente comando:

```
apt-get install ant maven git unzip vim
```

Si se han seguido las instrucciones anteriores, el sistema cumplirá con los requisitos previamente mencionados, por lo que el siguiente paso será descargar y descomprimir el DAS. En este proyecto se ha utilizado la versión 3.1.0 de WSO2 DAS, habiéndola descargado desde el siguiente enlace:

http://wso2.com/analytics/

Una vez que hemos descargado el producto, tendremos que descomprimirlo en un directorio dedicado, en este caso, */home/wso2das-3.1.0*. Para ello, ejecutamos lo siguiente desde el directorio en el que se encuentra el archivo comprimido que acabamos de descargar:

```
unzip wso2das-3.1.0.zip -d /home
```

Llegados a este punto, el servidor ya se encuentra instalado y preparado para ser ejecutado. Para ello, debemos de ejecutar lo siguiente dentro del directorio dedicado al DAS:

```
cd bin/
./wso2server.sh
```

[17] La ejecución de los comandos indicados en los anexos se ha de realizar con permisos de super-usuario.

Instalación y configuración del entorno Python

Para que el script de Python se pueda ejecutar correctamente, el servidor debe tener instalado una serie de paquetes y dependencias, con el fin de proveer al script de las herramientas necesarias para realizar sus funciones. En primer lugar, se va ha de preparar el sistema para usar el *Flask* y *virtualenv*:

```
apt-get install python
apt-get install python-pip python-dev build-essential

pip install --upgrade pip
pip install --upgrade virtualenv
pip install flask flask-restful
cd /home
wget https://github.com/pallets/flask/archive/master.zip
unzip master.zip
mv master/ flask/ && cd flask/
```

En segundo lugar, instalaremos en el servidor el módulo *PyEDFlib*, necesario para que Python pueda trabajar con archivos del formato *EDF* y *EDF+*.

```
cd /home
wget https://github.com/holgern/pyedflib/archive/master.zip
unzip master.zip
mv master/ pyedflib/

cd pyedflib
python setup.py build
python setup.py install
```

Por último, tendremos que crear y configurar un entorno aislado o *virtualenv* para la ejecución del script. Para ello, ejecutamos lo siguiente:

```
cd /home/flask
virtualenv flask
source flask/bin/actívate

pip install flask gunicorn
flask/bin/pip install flask-htttpauth requests numpy cython pyfcm
```

Además, se ha instalado también Gunicorn, el cual utilizaremos para la ejecución del script. Hecho esto, el servidor ya estará completamente preparado para ejecutar el script de Python, permitiendo que este funcione de la forma correcta.

ANEXO C: GUÍA DE EJECUCIÓN DEL SCRIPT DE PYTHON

Como se ha comentado en el apartado 4.3, el script de Python utilizado en este proyecto tiene una importancia notable, debido a que sobre él se soportan la conversión de los datos a formato EDF+ y la notificación de alertas. En este Anexo se pretende detallar el proceso a seguir para la ejecución correcta del script.

Aprovechando las herramientas que ofrecen los sistemas Linux, la ejecución del script se va a realizar como si de un servicio se tratase, utilizando para ello una herramienta del Linux llamada *systemd*, la cual nos permite ejecutar un programa como un demonio. Para hacer que *systemd* ejecute el script de Python, necesitamos crear un archivo de configuración del servicio, cuya estructura se detallará a continuación.

Archivo de configuración del servicio

Para que *systemd* pueda ejecutar el script como un servicio, debemos de indicarle algunas instrucciones dentro de un archivo de configuración. Este archivo debe situarse en el directorio */etc/systemd/system/*, así como tener un nombre acabado en "*.service*". En este caso, el nombre del fichero se corresponde con el nombre del script, *PyServer.service*, el cual utilizaremos cuando queramos indicar a *systemd* que inicie o detenga la ejecución de éste. La estructura del archivo de configuración es la siguiente:

```
[Unit]
Description=Gunicorn instance to serve PyServer.py
After=network.target

[Service]
User=root
Group=root
WorkingDirectory=/home/flask/
Environment="PATH=/home/flask/bin"
ExecStart=/home/flask/bin/gunicorn PyServer:app -b 127.0.0.1:62222

[Install]
WantedBy=multi-user.target
```

Se puede apreciar que la estructura está formada por tres secciones: *Unit, Service* e *Install*, aunque pueden incluirse algunas más. Cada uno de ellos tiene una función a la hora de indicar a *systemd* cómo ejecutar el script.

- **Unit**

  En esta sección se incluyen los metadatos y las dependencias de ejecución. En este caso, únicamente se ha indicado una breve descripción y que la ejecución se realice después de haber alcanzado el nivel de red, es decir, que nuestro equipo tenga conexión de red, pues de otra manera, no podríamos poner nuestro script a escuchar peticiones.

- **Service**

  En esta sección se deben incluir la configuración relacionada con el servicio que se va a ejecutar. En este caso concreto, se ha indicado el usuario y grupo bajo el que queremos ejecutar el script, el directorio de trabajo, una variable de entorno en la que indicamos la localización de los ejecutables y los parámetros de ejecución. Con respecto a esto último, estamos indicando que el script se ejecute usando Gunicorn, asociado al puerto 62222 del equipo local.

- **Install**

  En la última sección indicamos a qué nivel del arranque del sistema queremos que se comience a ejecutar el script. En este caso, hemos indicado que se inicie cuando se alcance el nivel *multi-usuario*.

Ejecución del servicio

Una vez hemos creado el archivo de configuración, ya podremos ejecutar el script con la ayuda de *systemd*. La ejecución de éste se puede realizar de dos maneras diferentes: automáticamente cuando el sistema se inicia o manualmente. A continuación, se indica cómo utilizar ambos métodos:

Para iniciar y detener el servicio de forma manual, debemos utilizar los comandos siguientes, respectivamente:

```
sudo systemctl start PyServer

sudo systemctl stop PyServer
```

Para activar y desactivar la ejecución en el inicio del sistema, debemos utilizar los siguientes comandos, respectivamente:

```
sudo systemctl enable PyServer

sudo systemctl disable PyServer
```

ANEXO D: GUÍA DE USUARIO – WSO2 DAS

En este Anexo se pretende ofrecer una guía del producto WSO2 DAS, con la finalidad de introducir al lector en la estructura de dicho producto, así como los principales elementos y herramientas que lo componen. Para poder seguir está guía, se debe haber seguido previamente los pasos indicados en el Anexo B: Instalación y configuración del servidor y haber iniciado el DAS.

Acceso a la consola de administración

Para acceder a la consola de administración del DAS, debemos utilizar la URL https://triton.us.es:29443/carbon, donde el puerto 29443 apunta directamente al puerto 9443 de la instancia en la que está instalado el DAS, el cual se usa por defecto para las conexiones HTTPS. Una vez accedamos a la dirección anterior, en el navegador aparecerá algo como esto, lo cual nos confirma que el DAS se iniciado correctamente.

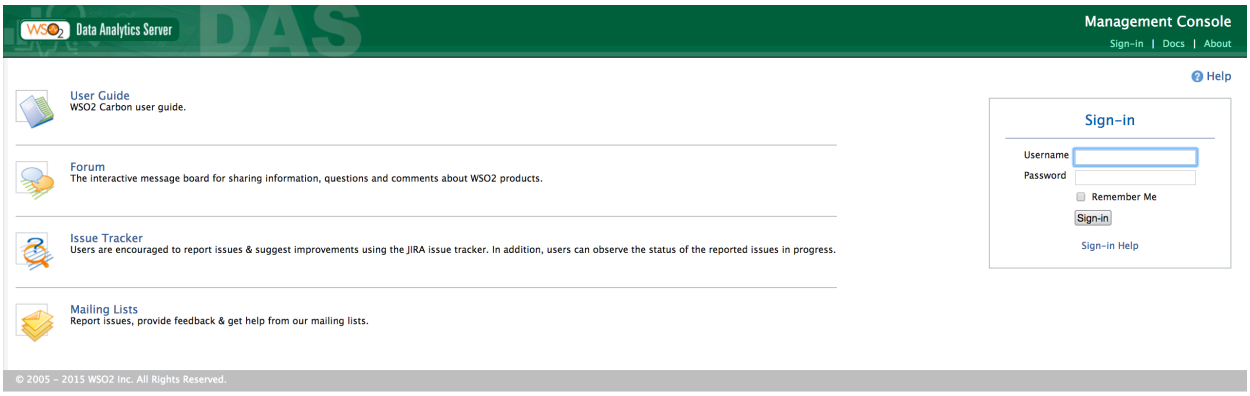


Figura 26. Consola de administración – DAS

Para identificarnos, podemos hacer uso de la cuenta de usuario creada por defecto en el DAS, usuario *admin* y contraseña *admin*, lo cual nos permite iniciar sesión y acceder al menú principal del DAS.

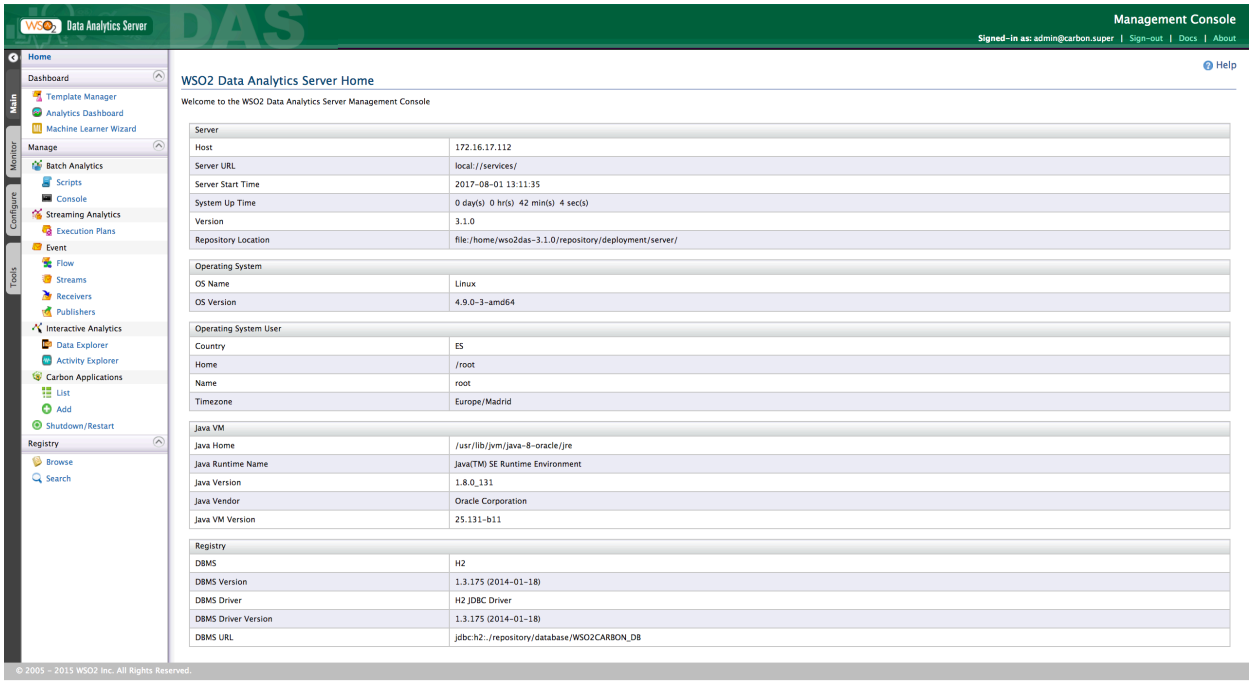


Figura 27. Menú principal - DAS

Creación de Streams

Como se ha mencionado en el apartado 2.1, en el DAS todo se rige por flujos o *streams,* lo cual hace que sean una parte imprescindible para su funcionamiento. Por ello, es importante saber cómo crearlos y gestionarlos para obtener el resultado esperado. A continuación, se detalla el proceso necesario para crear un *stream*, lo cual se puede aplicar de manera similar a la hora de editarlo.

Para crear un *stream*, debemos acudir al apartado *Main* > *Manage* > *Event* > *Streams*, lo que nos conducirá a una pantalla similar a la de la Figura 28.

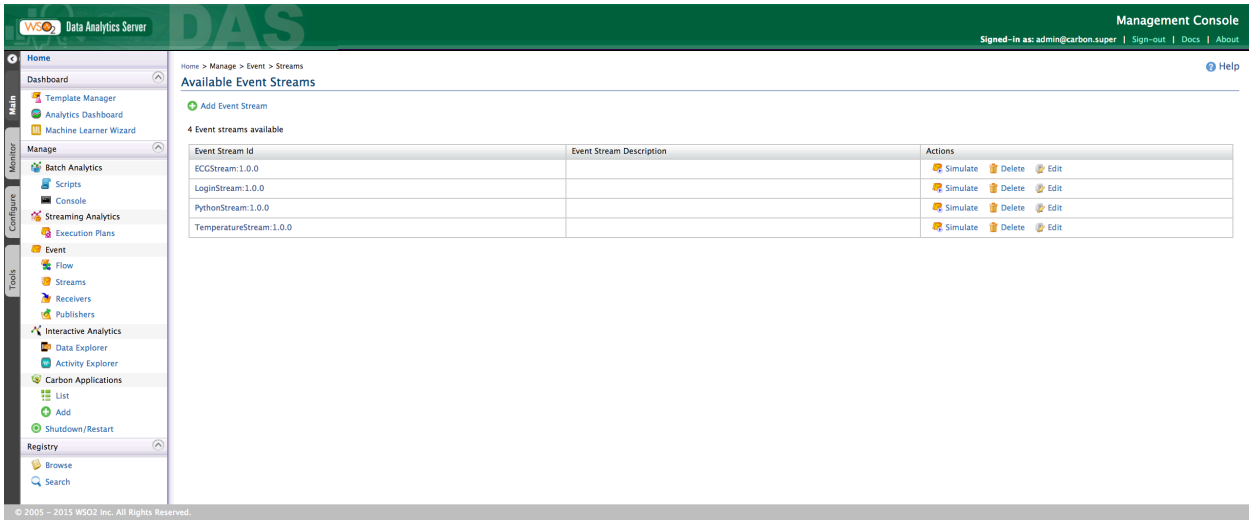


Figura 28. Apartado *Streams* - DAS

Desde esta ventana podemos crear un *stream* u observar todos los que tenemos creados, pudiendo editarlos, eliminarlos o simular su funcionamiento. Para acceder a crear un *stream* debemos de pulsar sobre **Add Event Stream**, llevándonos a una ventana como la siguiente.

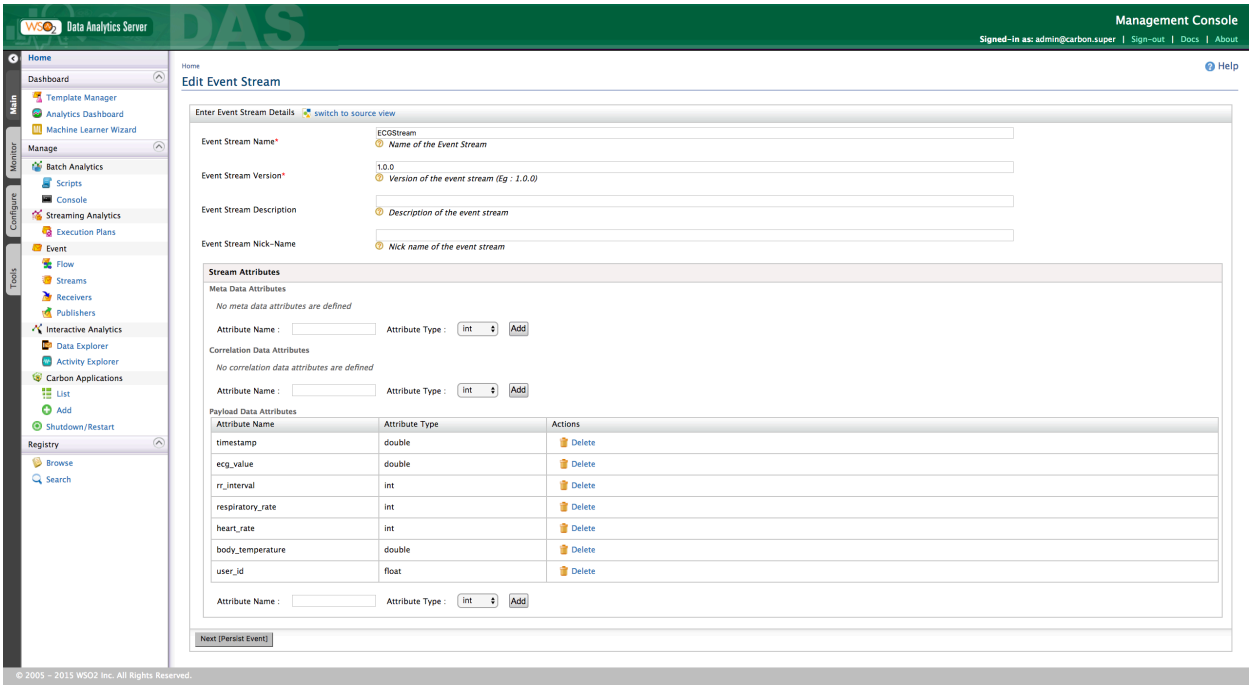


Figura 29. Creación de *streams* I – DAS

Como se puede observar en la Figura 29, para crear un *stream* debemos de asignarle un nombre y una versión, permitiéndolos agregar, además, una breve descripción y un alias.

Tras esto, lo siguiente será crear la estructura de datos del *stream*, la cual se divide en tres elementos: *Meta Data Attributes, Correlation Data Attributes* y *Payload Data Attributes*. Para una mayor simplificación, los datos se van a incluir únicamente dentro de *Payload Data Attributes*. Dentro de este apartado, añadiremos cuantos atributos nos sean necesarios, asignándoles un nombre y un tipo (*int, long, double, float, string o bool*).

Para proseguir con el proceso de creación, debemos de pulsar sobre **Next [Persist Event]**, tras lo cual tendremos que decidir si queremos que los datos recibidos se almacenen en la base de datos del sistema.

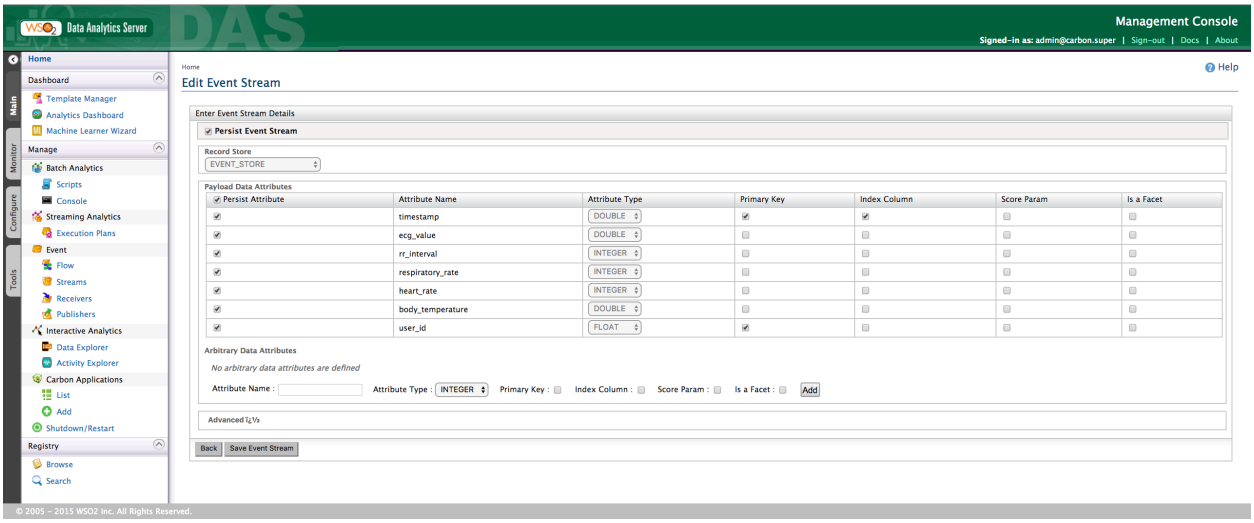


Figura 30. Creación de *streams* II - DAS

Si deseamos que los datos recibidos se almacenen en el sistema para un posterior procesamiento, como en nuestro caso, debemos de marcar la opción **Persist Event Stream** y elegir qué atributos almacenar de entre todos los recibidos.

Como se puede apreciar en la Figura 30, los atributos *timestamp* y *user_id* están marcados como *Primary Key*, lo que implica que no pueda existir más de una entrada en la base de datos con el mismo par de valores *timestamp*:user_*id*. Además, podemos ver como el atributo *timestamp* también tiene marcada la casilla de *Index Column*, lo cual es necesario para que se pueda acceder a los datos almacenados usando como índice dicho atributo.

Para finalizar, debemos de pulsar sobre **Save Event Stream** y ya tendremos el *stream* creado.

Creación de Receivers

Para acceder a la herramienta de creación de *receivers*, debemos de acceder desde la página principal al apartado *Main* >*Manage* > *Event* > *Receivers*, lo cual nos mostrará una ventana como la siguiente.

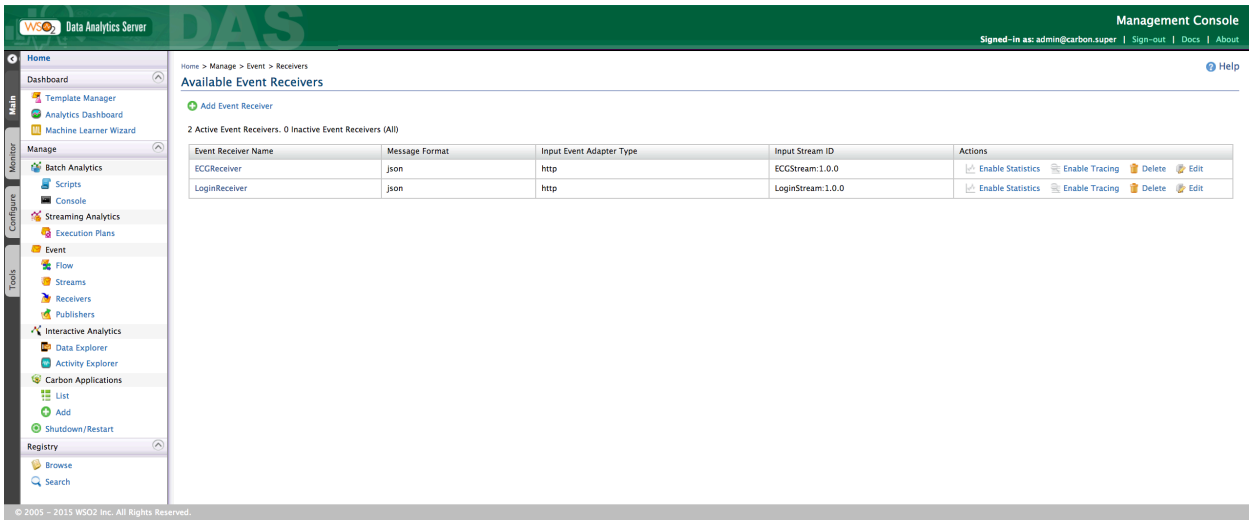


Figura 31. Apartado *Receivers*

Desde esta ventana podremos gestionar y monitorizar los *receivers* ya creados, así como crear otros nuevos. Para esto último, debemos de pulsar sobre **Add Event Receiver**, accediendo a una ventana de aspecto similar a la de la Figura 32.

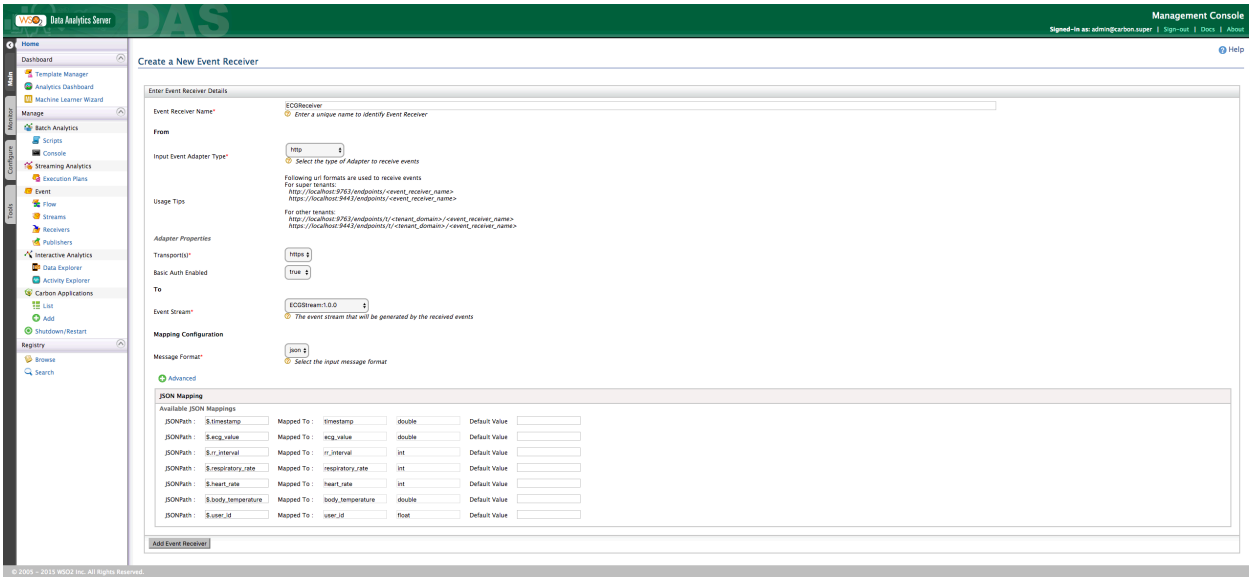


Figura 32. Creación de *receivers*

Para crear un nuevo *receiver*, tenemos que asignarle un nombre y seleccionar el tipo de adaptador acorde al método de recepción de los eventos. Según seleccionemos un tipo de adaptador u otro, tendremos que personalizar aspectos como el tipo de transporte o el tipo de autenticación, como en el caso de la figura anterior, que es *HTTP*. Después, tendremos que seleccionar el *stream* al que queremos redirigir la entrada de datos, el cual debe de estar previamente creado.

Por último, se dispone de una configuración opcional para personalizar el formato de los mensajes que esperamos recibir. En nuestro caso, se ha tenido que utilizar la opción *JSON Mapping* para permitir al *receiver* recibir más de un evento. Hecho esto, sólo tendremos que pulsar sobre **Add Event Receiver** para crear el nuevo *event receiver*.

Creación de Publisher

Para acceder a la ventana de gestión de *publishers* debemos acceder al apartado *Main* > *Manage* > *Event* > *Publishers*. Una vez ahí, veremos una ventana como la siguiente, a través de la cual podremos gestionar y monitorizar los *publishers* ya creados, además de crear otros nuevos.

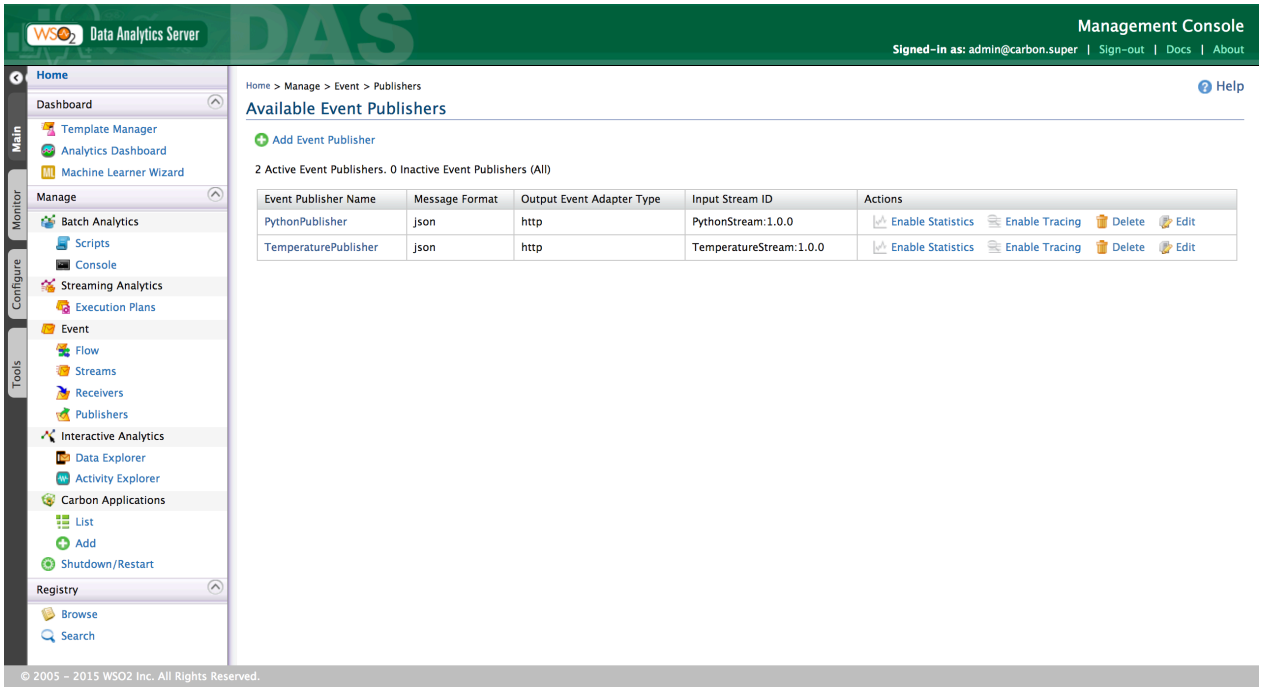


Figura 33. Apartado *Publishers*

Para crear un nuevo *publisher* debemos de pulsar sobre **Add Event Publisher**, lo que nos dará acceso a la venta de creación de *publishers*, que tendrá un aspecto similar al de la Figura 34.

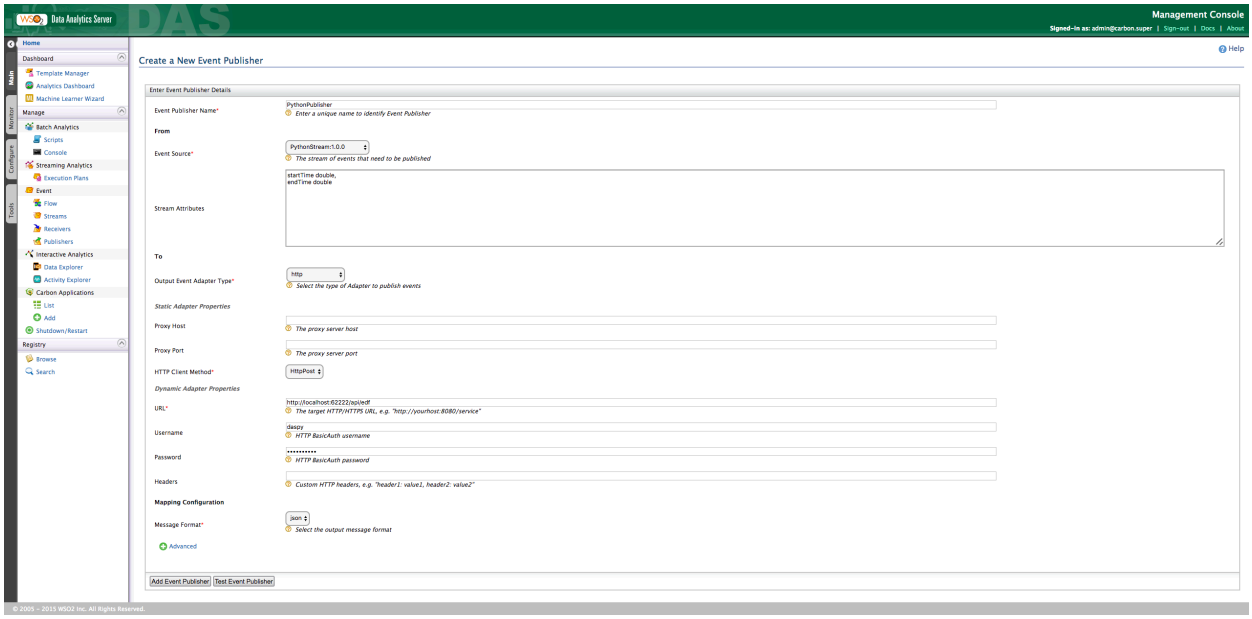


Figura 34. Creación de *publishers*

En la creación de un *publisher,* debemos asignarle un nombre y seleccionar el *stream* desde el que obtendrá el flujo de datos. Además, debemos de indicar el tipo de adaptador a través del cual, el *publisher* enviará los datos. Al igual que con los *receivers*, según el tipo de adaptador, deberemos de configurar ciertos parámetros. En nuestro caso, hemos indicado que sea un adaptador del tipo *HTTP*, que envíe un mensaje del tipo POST con formato JSON a la URL http://localhost:62222/api/edf, autenticándose mediante el usuario y contraseña indicados. Para finalizar, pulsaremos sobre **Add Event Publisher**.

Creación de Execution Plans

Como se ha comentado en el apartado 2.1, los *execution plans* son el núcleo de funcionamiento del DAS, por lo que es de gran importancia saber cómo crearlos y administrarlos. En este apartado, se va a detallar el proceso de creación de *execution plans*, así como la estructura de estos.

Para acceder a la ventana de administración de los *execution plans*, debemos de ir a *Main* > *Manage* > *Streaming Analytics* > *Execution Plans*. Una vez allí, veremos una ventana similar a la de la Figura 35.

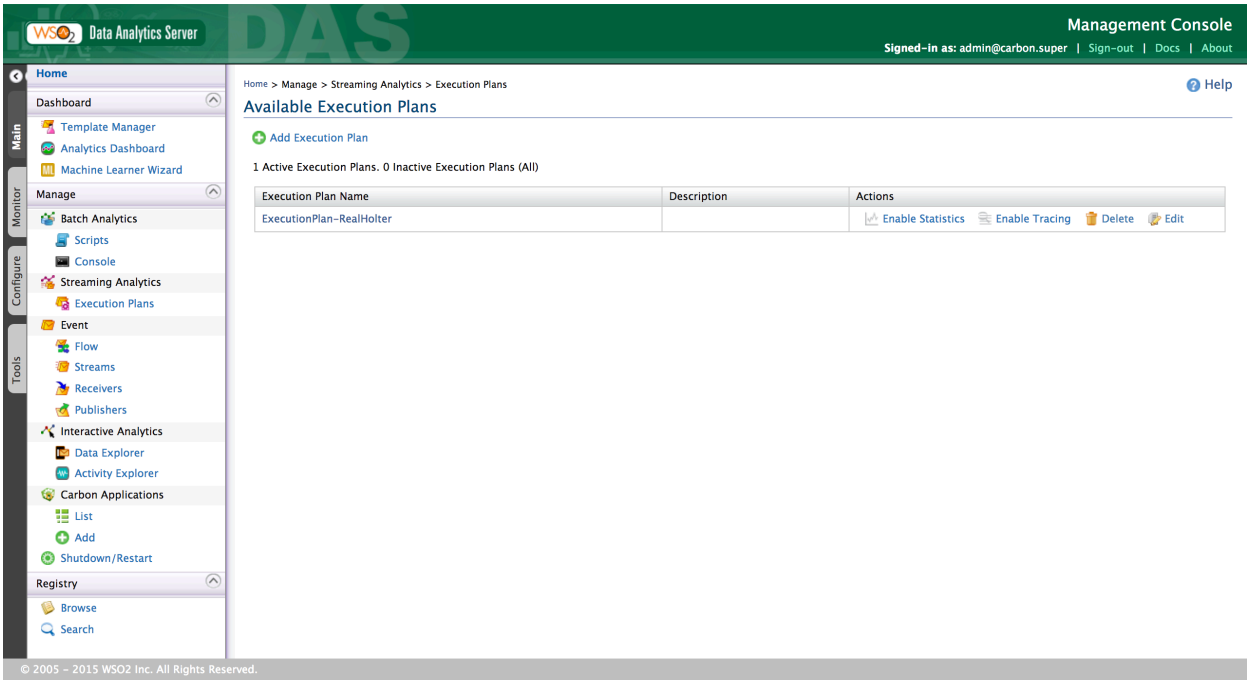


Figura 35. Apartado *Execution Plans*

En esta ventana podremos ver los *execution plans* que tenemos actualmente activos y crear otros nuevos. Para esto último, debemos de pulsar sobre **Add Execution Plan**, lo que nos llevará a una ventana como la siguiente.

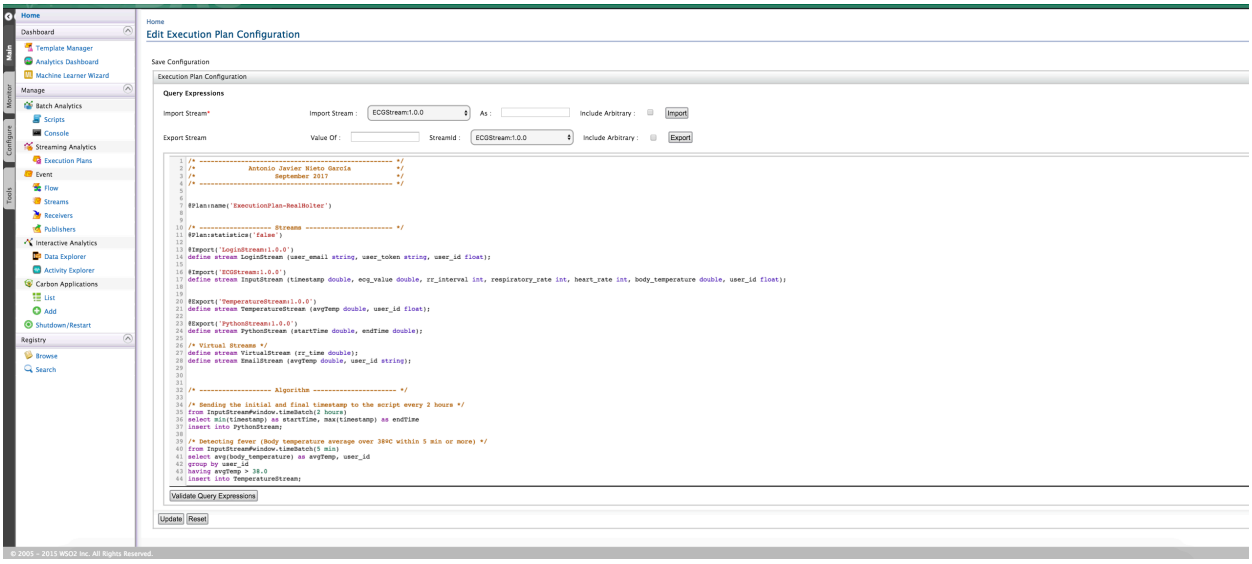


Figura 36. Creación de *execution plans*

En la Figura 36, podemos observar un *execution plan* ya creado, dentro del cuadro de edición podemos diferenciar dos apartados: *Streams* y Algoritmo.

En el primero de ellos, se definen los *streams* de los que se van a hacer uso en el algoritmo, los cuales pueden ser de entrada o de salida. Para añadir un *stream* debemos de utilizar las herramientas que están situadas sobre el cuadro de edición, donde tendremos que seleccionar el *stream* en cuestión, asignarle un nombre y pulsar sobre **Import** o **Export**, según el tipo de *stream* añadido.

En lo que respecta al segundo apartado, Algoritmo, en él se sitúa el código principal del *execution plan*, el cual se encarga de procesar los datos provenientes de los *streams* de entrada y exportar los resultados mediante *streams* de salida.

Cada vez que queramos comprobar la validez del código, podremos pulsar sobre **Validate Query Expressions**, lo cual hará una comprobación rápida del código y nos notificará si existe algún error en él. Una vez hayamos terminado de configurar el *execution plan*, deberemos de pulsar sobre **Add Execution Plan** o **Update**, en caso de estar editando uno ya existente.

Como una ayuda adicional, el DAS nos ofrece la posibilidad de ver un esquema del flujo de datos, lo cual nos permite observar de manera resumida la *estructura de funcionamiento de* éste. Para ello, debemos de acceder a Main > Manage > Events > Flow. Una vez ahí, observaremos algo semejante a la Figura 37.

Como se puede apreciar, el flujo de datos está indicado por el sentido de las flechas que unen un elemento con otro, los cuales pueden ser identificados con la ayuda del índice de elementos que aparece en la parte superior. En el caso concreto de la figura indicada, vemos como existen dos *receivers* conectados a sus respectivos *streams*, los cuales usa el *execution plan* como *streams* de entrada. A se vez, se aprecian dos *streams* de salida conectados a sendos *publishers*, cuyo flujo de datos proviene del *execution plan*.

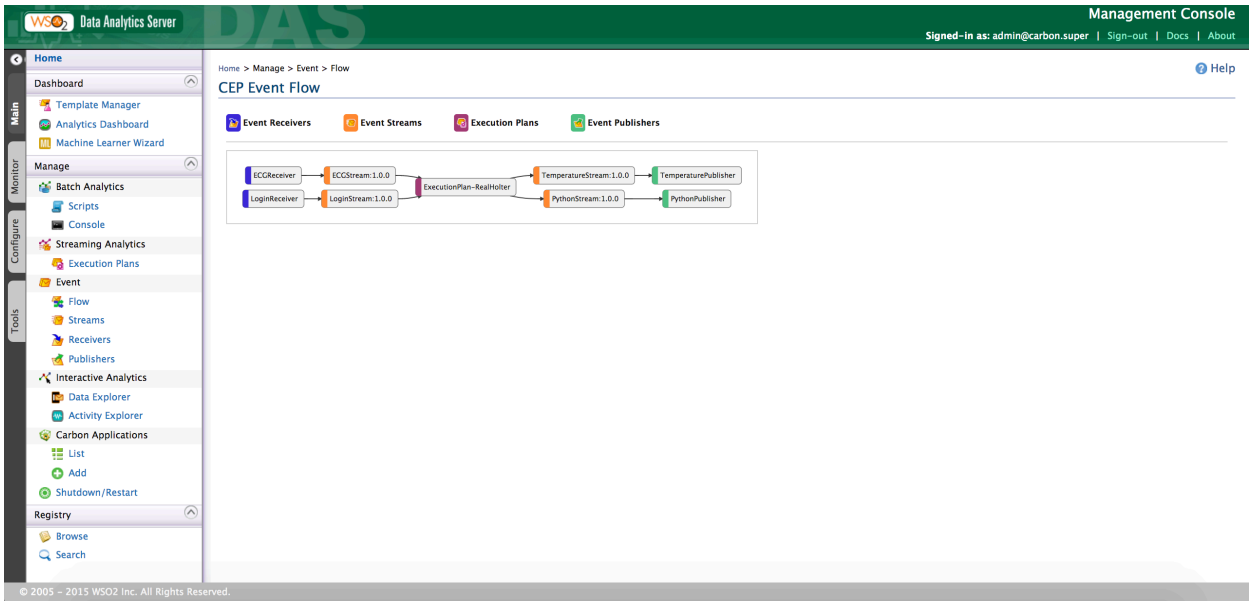


Figura 37. Apartado *Flow*

Consulta a la base de datos

Otras de las funcionalidades que nos ofrece el DAS es la de almacenar los eventos en una base de datos interna, a la cual podremos acceder a través de *Main > Manage > Interactive Analytics > Data Explorer*.

En esta ventana podemos consultar las tablas existentes, pudiendo filtrar los resultados por fecha, clave primaria o añadiendo una consulta manualmente. Además, podremos limitar el número máximo de resultados, así como programar un borrado de datos a la hora que elijamos, para que se ejecute cada día.

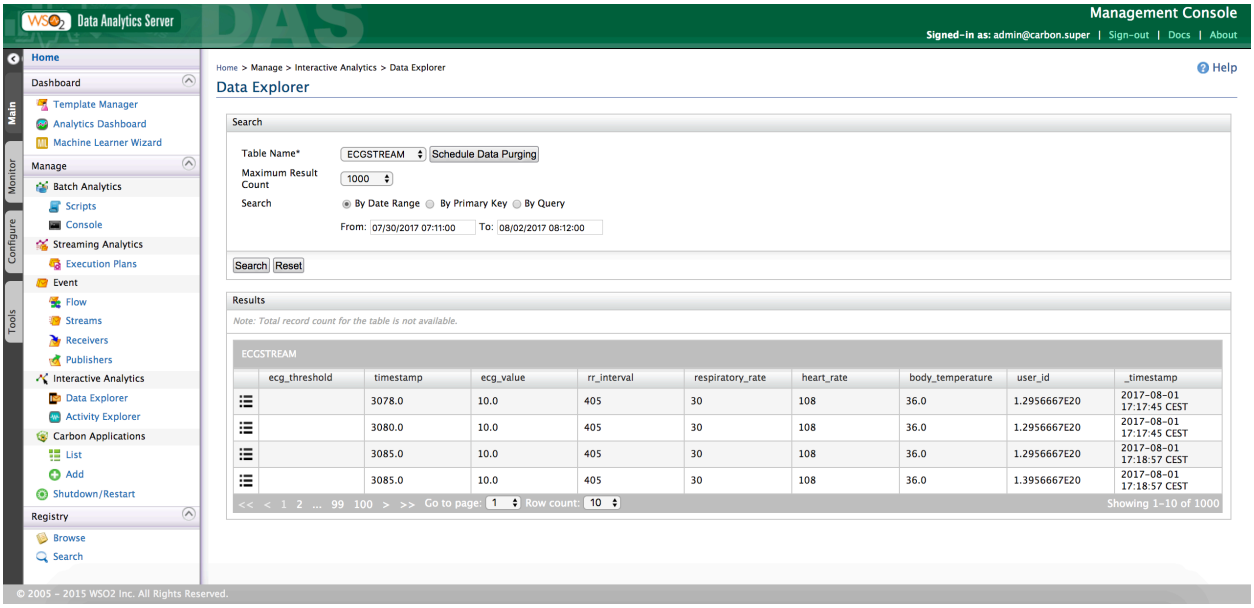


Figura 38. Apartado *Data Explorer*

ANEXO E: GUÍA DE USUARIO - APLICACIÓN ANDROID

La finalidad de este Anexo es la de ofrecer al lector un manual de uso de la aplicación Android que forma parte del sistema desarrollado en este proyecto. Para ello, se indicará desde cómo acceder a ésta, hasta cómo asociar un sensor a nuestro dispositivo, pasando por la administración del servicio de monitorización.

Acceso a la aplicación

Tal y como se ha comentado en el apartado 2.4, para acceder a la aplicación deberemos de autenticarnos mediante una cuenta de Google.

Cuando iniciamos la aplicación, la primera pantalla que vemos es la de autenticación de usuarios, la cual tiene el aspecto siguiente.

Figura 39. Pantalla de autenticación

El primer paso para la autenticación es pulsar sobre **Iniciar sesión**, tras lo cual nos aparecerá la opción de elegir una cuenta de Google ya asociada al dispositivo, o en su carencia, nos permitirá añadir una cuenta nueva. Una vez que elijamos una cuenta, la aplicación nos dará acceso a la pantalla principal. En ésta, sólo tendremos que pulsar sobre **Logout** para volver a cerrar la sesión.

Asociación de un sensor

Otra de las funcionalidades que nos ofrece la aplicación es la de asociar un sensor a nuestro dispositivo, para lo cual tendremos que tener activada la conexión Bluetooth en nuestro dispositivo. Una vez nos aseguremos de ello, tendremos que acceder a la pantalla de configuración del sensor, pulsando para ello el botón **Sensor configuration** situado en la pantalla principal, lo cual nos llevará a una nueva pantalla con el siguiente aspecto.

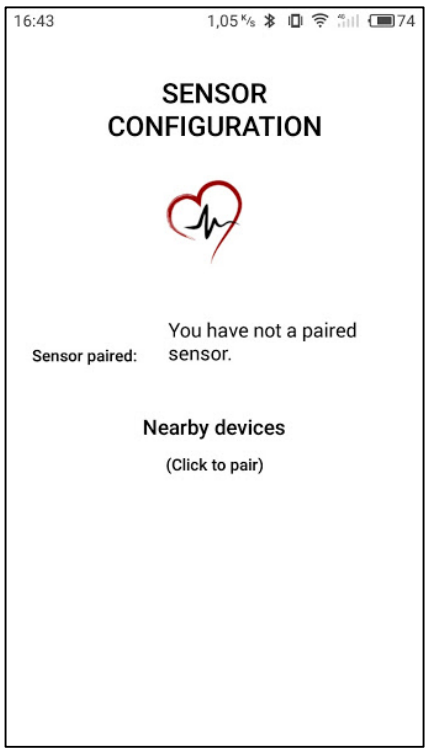


Figura 40. Pantalla de configuración del sensor I

Como podemos observar, en la parte intermedia de la figura pantalla se indica que aún no tenemos ningún sensor asociado, por lo que tendremos que asociar uno si queremos activar el sistema de monitorización. Para ello, si apreciamos la parte inferior de la pantalla, veremos que hay un apartado en el que deben aparecer los sensores que podemos asociar. Por lo tanto, si activamos un sensor cerca de nuestro dispositivo, veremos que éste aparece en la parte inferior de nuestra pantalla.

Una vez el dispositivo detecte los sensores cercanos, éste los mostrará en forma de lista, sobre la que podremos pulsar para asociar un dispositivo. El aspecto sería similar al de la Figura 41.

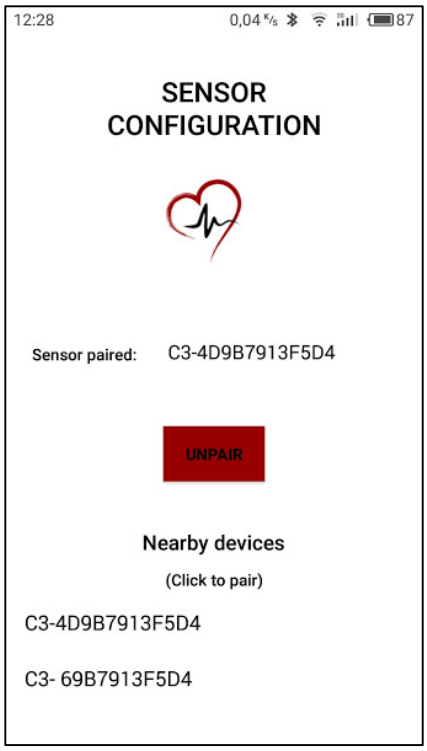


Figura 41. Pantalla de configuración de sensor II

Ahora vemos que tenemos un dispositivo asociado, así como otro dispositivo cercano disponible para ser asociado. Además, podemos apreciar que ha aparecido un botón en el centro de la pantalla, el cual nos permite des-asociar el dispositivo que tenemos asociado en ese momento.

Cuando tengamos un dispositivo asociado, únicamente tendremos que volver a la pantalla principal y ya podremos activar el servicio de monitorización.

Activar/Desactivar el servicio de monitorización

Como principal funcionalidad, la aplicación ofrece al usuario la capacidad de activar y desactivar el servicio de monitorización, simplemente pulsando un botón. Para ello, tendremos que tener previamente un dispositivo asociado, no permitiéndonos activar el servicio en caso de que no sea así, así como la conexión Bluetooth activada en nuestro dispositivo móvil.

Antes de activar el servicio, la pantalla principal de la aplicación tendrá un aspecto similar a la Figura 42.

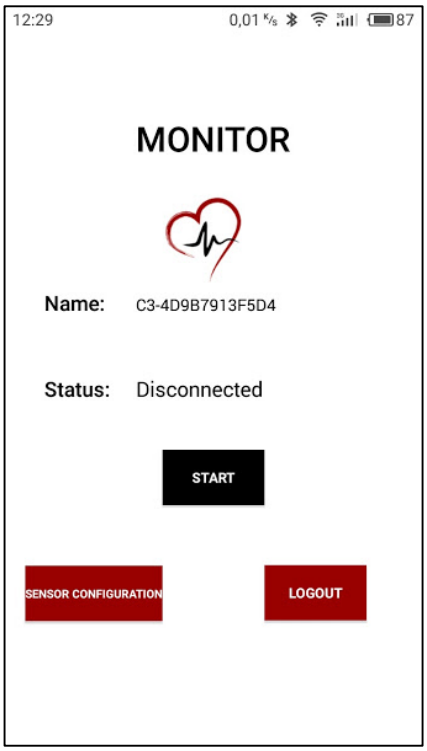


Figura 42. Pantalla principal I

Como podemos apreciar, la aplicación nos indica que ya tenemos un dispositivo asociado, mostrando su nombre en el apartado *Name*, además del estado del servicio en el apartado *Status*, el cual se encuentra desactivado. Cuando decidamos activar el servicio de monitorización, solo tendremos que pulsar sobre el botón **Start**, actualizando la apariencia de la pantalla hasta tener el estado siguiente.

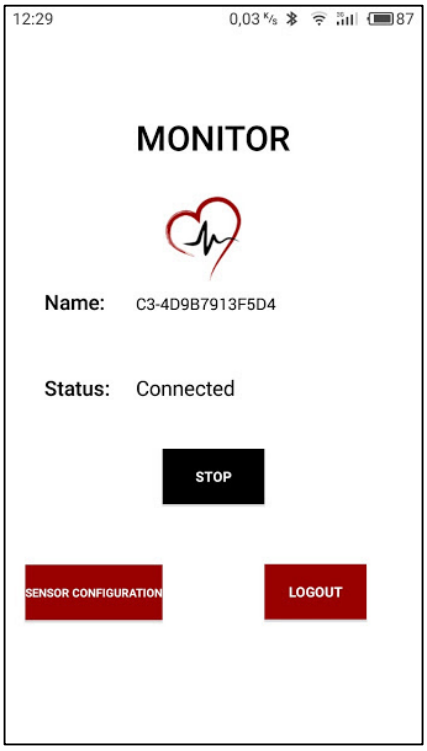


Figura 43. Pantalla principal II

Como se puede observar, el estado ahora aparece como conectado y el botón que acabamos de pulsar para activar el servicio, ahora sirve para detenerlo. Si queremos seguir el estado del servicio de monitorización, podremos hacerlo a través de las notificaciones que la aplicación muestra en la barra de notificaciones de nuestro dispositivo. En la Figura 44 y la Figura 45 podemos ver el aspecto de estas notificaciones cuando el servicio está activo y parado, respectivamente.

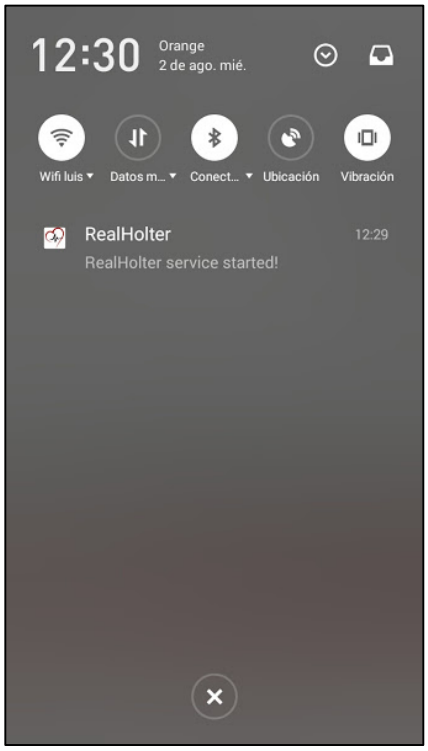


Figura 44. Notificación de servicio activo

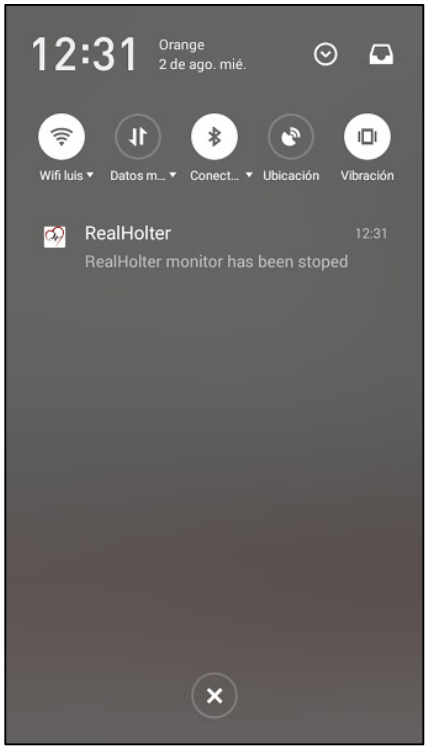


Figura 45. Notificación de servicio detenido

Además de comprobar que el servicio se ha iniciado o detenido, la aplicación ofrece información sobre la conexión con el dispositivo, de manera que prestando atención a la barra de notificaciones podremos saber si la conexión con el sensor se ha establecido y si el estado de ésta es bueno o malo. En las siguientes figuras podemos apreciar dicha información.

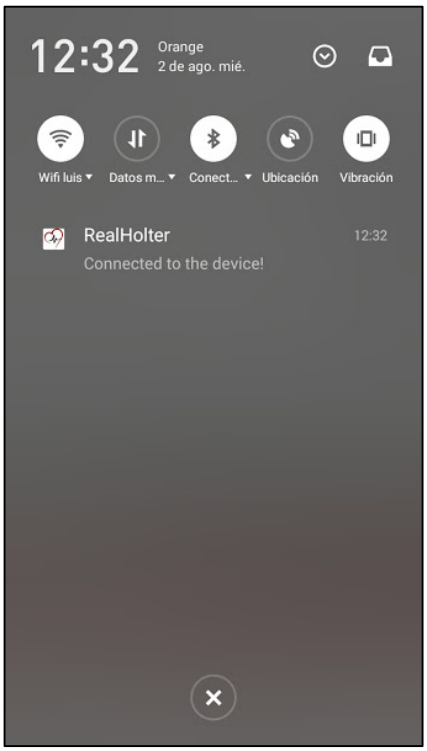


Figura 46. Notificación de conexión establecida

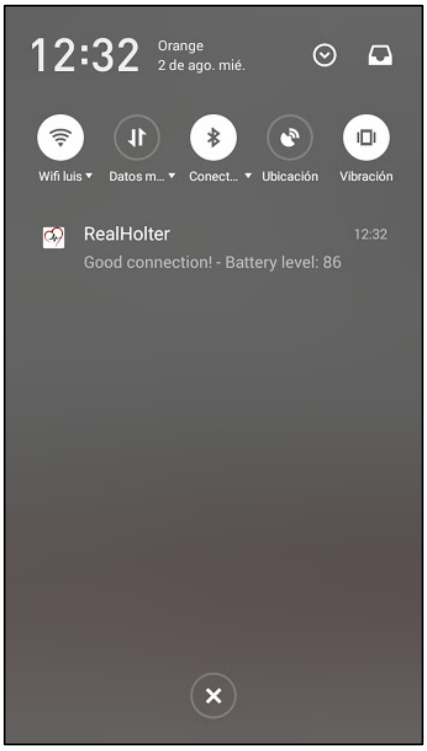


Figura 47. Notificación de buena conexión

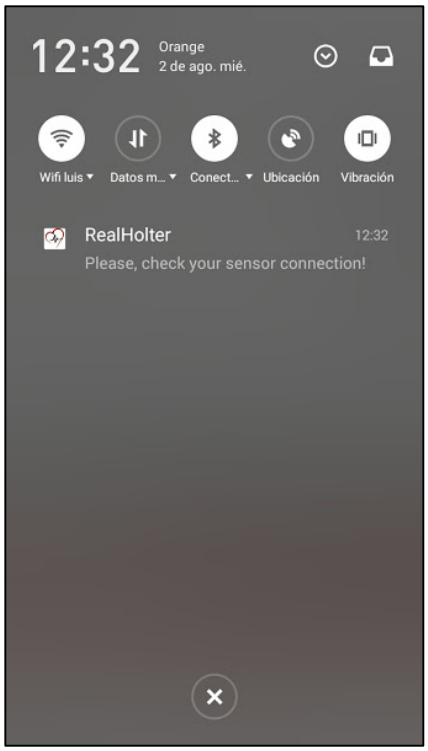


Figura 48. Notificación de mala conexión

ANEXO F: CÓDIGOS

Aplicación Android

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.3'
        classpath 'com.google.gms:google-services:3.0.0'
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Figura 49. Archivo *build.gradle* de proyecto

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile project(':cortrium-sdk')
    compile files('libs/gson-2.1.jar')
    compile files('libs/httpclient-4.2.5.wso2v1.jar')
    compile files('libs/httpcore-4.3.3.wso2v1.jar')
    compile files('libs/axiom_1.2.11.wso2v4.jar')
    compile files('libs/commons-pool-1.5.6.wso2v1.jar')
    compile files('libs/achartengine-1.2.0.jar')
    compile files('libs/GraphView-4.2.1.jar')
    compile files('libs/jxl-2.6.12.jar')
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support:design:25.3.1'
    compile 'com.mcxiaoke.volley:library:1.0.0'
    compile 'com.androidplot:androidplot-core:1.4.2'
    compile 'com.google.android.gms:play-services-auth:11.0.4'
    compile 'com.google.firebase:firebase-messaging:11.0.4'
    testCompile 'junit:junit:4.12'
}

apply plugin: 'com.google.gms.google-services'
```

Figura 50. Dependencias del archivo *build.gradle* de aplicación

```java
@Override
public void onCreate() {

    super.onCreate();

    mRequestQueue = Volley.newRequestQueue(this);
    sInstance = this;

    // Enable bluetooth connection when starts the application
    btadapter = BluetoothAdapter.getDefaultAdapter();
    if (!btadapter.isEnabled())
        btadapter.enable();

    // Trusting all server certificates (temporal)
    handleSSLHandshake();

    // Configure sign-in to request the user's ID, email address, and basic
    // profile. ID and basic profile are included in DEFAULT_SIGN_IN.
    GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestEmail()
            .build();

    // Build a GoogleApiClient with access to the Google Sign-In API and the
    // options specified by gso.
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
            .build();

}
```

Figura 51. RealHolterApplication - Método *onCreate*

```java
@Override
protected void onCreate(Bundle savedInstanceState) {

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


    preferences = getSharedPreferences(Application.class.getName(), MODE_PRIVATE);

    // Gets GoogleApiClient instance and connect to Google Play services
    mGoogleApiClient = RealHolterApplication.getInstance().get_GoogleApiClient();
    mGoogleApiClient.connect();

    // Sets the dimensions of the sign-in button and the click listener
    SignInButton signInButton = (SignInButton) findViewById(R.id.sign_in_button);
    signInButton.setSize(SignInButton.SIZE_STANDARD);
    signInButton.setOnClickListener((v) -> {
            switch (v.getId()) {
                case R.id.sign_in_button:
                    login();
                    break;
            }
    });

}
```

Figura 52. LoginActivity – Método *onCreate*

```java
/**
 * Gets the sign-in intent from Google Sign-In login.
 */
private void login() {

    Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);
    startActivityForResult(signInIntent, RC_SIGN_IN);

}


/**
 * Gets the intent result from the login request and call the function
 * {@link #handleSignInResult(GoogleSignInResult)}.
 */
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // Result returned from launching the Intent from GoogleSignInApi.getSignInIntent(...);
    if (requestCode == RC_SIGN_IN) {
        GoogleSignInResult result = Auth.GoogleSignInApi.getSignInResultFromIntent(data);
        handleSignInResult(result);
    }

}
```

Figura 53. LoginActivity – Métodos *login* y *onActivityResult*

```java
/**
 * Handles the login request result from Google Sign-In.
 *
 * @param result
 */
private void handleSignInResult(GoogleSignInResult result) {

    if (result.isSuccess()) {

        // Signed in successfully, show authenticated UI.
        GoogleSignInAccount acct = result.getSignInAccount();

        // Gets user information from the Google account
        String user_id = acct.getId().trim();
        String user_email = acct.getEmail();

        // Gets the application FCM token
        String user_token = FirebaseInstanceId.getInstance().getToken();

        // Saves the gathered information
        SharedPreferences.Editor editor = preferences.edit();
        editor.putString(Extra.USER_ID, user_id);
        editor.putString(Extra.USER_EMAIL, user_email);
        editor.putString(Extra.USER_TOKEN, user_token);
        editor.apply();

        sendLogin(user_id, user_email, user_token);

        // Shows the main page
        Class<? extends Activity> activity = MainActivity.class;
        Intent intent = new Intent(LoginActivity.this, activity);
        startActivity(intent);

    }
}
```

Figura 54. LoginActivity - Método *handleSignInResult*

Script de Python

```python
def create_edf (ecg_list, body_temp_list, rr_interval_list, respiration_rate_list, heart_rate_list):

    'Stores the sensor data into a EDF+ format file.'

    # Preparing file
    filename = "EDF_"+time.strftime("%Y%m%d_%H%M%S")+".edf"
    test_data_file = os.path.join('/home/EDF/files', filename)
    f = pyedflib.EdfWriter(test_data_file, 5,
                           file_type=pyedflib.FILETYPE_EDFPLUS)


    # Data arrays
    channel_info = []
    data_list = []

    ch_dict = {'label': 'ECG', 'dimension': 'uV', 'sample_rate': 250, 'physical_max': 600, 'physical_min': -600, 'digital_max': 32767, 'digital_min': -32768, 'transducer': '', 'prefilter':''}
    channel_info.append(ch_dict)
    r2 = np.array(ecg_list)
    data_list.append(r2)

    ch_dict = {'label': 'Temp body', 'dimension': 'degC', 'sample_rate': 41.66667, 'physical_max': 45, 'physical_min': 32, 'digital_max': 32767, 'digital_min': -32768, 'transducer': '', 'prefilter':''}
    channel_info.append(ch_dict)
    r2 = np.array(body_temp_list)
    data_list.append(r2)

    ch_dict = {'label': 'Dur RR', 'dimension': 'ms', 'sample_rate': 41.66667, 'physical_max': 600, 'physical_min': 0, 'digital_max': 32767, 'digital_min': -32768, 'transducer': '', 'prefilter':''}
    channel_info.append(ch_dict)
    r2 = np.array(rr_interval_list)
    data_list.append(r2)

    ch_dict = {'label': 'Resp body', 'dimension': '', 'sample_rate': 41.66667, 'physical_max': 300, 'physical_min': 0, 'digital_max': 32767, 'digital_min': -32768, 'transducer': '', 'prefilter':''}
    channel_info.append(ch_dict)
    r2 = np.array(respiration_rate_list)
    data_list.append(r2)

    ch_dict = {'label': 'Vel heart', 'dimension': 'bpm', 'sample_rate': 41.66667, 'physical_max': 240, 'physical_min': 0, 'digital_max': 32767, 'digital_min': -32768, 'transducer': '', 'prefilter':''}
    channel_info.append(ch_dict)
    r2 = np.array(heart_rate_list)
    data_list.append(r2)


    # Writing file
    f.setSignalHeaders(channel_info)
    f.writeSamples(data_list)
    f.close()
    del f


    return None
```

Figura 55. Script Python - Método *create_edf*

```python
def purge_data(initial_date, final_date):

    'Executes a script to purge the data stored in the DAS database.'

    # Preparing command string
    command_string = '/home/wso2das-3.1.0/bin/analytics-backup.sh -purge -table "ECGSTREAM" -tenantId -1234 -timeFrom ' + initial_date + ' -timeTo ' + final_date
    result = os.system(command_string)

    return result
```

Figura 56. Script Python - Método *purge_data*

```python
@app.route('/api/edf', methods=['POST'])
@auth.login_required
def receive_edf():

    '''
    Fetch and prepares the sensor data to be store in a
    EDF+ format file.

    The server sends a HTTPS POST message to the URL
    http://127.0.0.1:62222/api/edf that contains the
    information related to the first and the last event
    in the recording interval. Then, the script request
    the data within the interval and call a function to
    convert the data to EDF+.
    '''

    # Check if the format of the message is JSON
    if not request.json:
        abort(400)

    # Parses the data from the content
    json_parsed = json.loads(request.data)
    content = json_parsed["event"]["payloadData"]


    # Getting initial timestamp
    payload = {
    "tableName":"ECGSTREAM",
        "query":"timestamp:"+"{:.0f}".format(content['startTime']),
    "start":0,
    "count":1
    }
    url = "https://localhost:9443/analytics/search"
    headers = {'Content-Type' : 'application/json'}

    response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=('admin', 'admin'))
    initial_ts = json.loads(response.text)[0]["timestamp"]


    # Getting final timestamp
    payload = {
        "tableName":"ECGSTREAM",
        "query":"timestamp:"+"{:.0f}".format(content['endTime']),
        "start":0,
        "count":1
    }
    url = "https://localhost:9443/analytics/search"
    headers = {'Content-Type' : 'application/json'}

    # Wait for 3 seconds before requesting the data
    time.sleep(3)

    response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=('admin', 'admin'))
    final_ts = json.loads(response.text)[0]["timestamp"]


    # Fetching data from the record time interval
    url = "https://localhost:9443/analytics/tables/ECGSTREAM/"+str(initial_ts)+"/"+str(final_ts+1)

    # Extracting the data from the response
    response = requests.get(url, auth=('admin', 'admin'), verify=False)
    content = json.loads(response.text)

    # Lists to store the data
    ecg_list = []
    body_temp_list = []
    rr_interval_list = []
    respiration_rate_list = []
    heart_rate_list = []

    # Saving data into the lists
    for event in content:
    ecg_list.append(event["values"]["ecg_value"])
    body_temp_list.append(event["values"]["body_temperature"])
    rr_interval_list.append(event["values"]["rr_interval"])
        respiration_rate_list.append(event["values"]["rr_interval"])
    heart_rate_list.append(event["values"]["heart_rate"])

    # Calling EDF convertion method
    create_edf(ecg_list, body_temp_list, rr_interval_list, respiration_rate_list, heart_rate_list)

    # Preparing time interval to purge the data
    initial_date = datetime.datetime.fromtimestamp(initial_ts/1000)
    final_date = datetime.datetime.fromtimestamp(final_ts/1000)

    initial_date_formated = initial_date.strftime("%y-%m-%d %H:%M:%S")
    final_date_formated = final_date.strftime("%y-%m-%d %H:%M:%S")

    # Purge the data
    command_result = purge_data(initial_date_formated, final_date_formated)

    return jsonify(payload)
```

Figura 57. Script Python - Método *receive_edf*

```python
@app.route('/api/fever_alert', methods=['POST'])
@auth.login_required
def fever_alert():

    '''
    Alerts the user when the server detects fever in the
    user temperature.

    The server sends a HTTPS POST message to the URL
    http://127.0.0.1:62222/api/fever_alert that contains
    the information necessary to fetch the data from the
    server database.

    Notification ways:

        -   Firebase Cloud Messaging (FCM): sends a push
            notification to the user device.

        -   E-mail: sends a email to the user mail account
            that the user used when logging in the system.

    '''

    # Check if the content format is JSON
    if not request.json:
        abort(400)

    # Parses the message content
    json_parsed = json.loads(request.data)
    content = json_parsed["event"]["payloadData"]

    # Gets the user_ID and mean temperature from the mesage content
    user_id = str(content['user_id'])
    temperature = str(content['avgTemp'])

    # Requests the user's email and token to the server, using the user's ID
    payload = {
        "tableName":"LOGINSTREAM",
            "query":"user_id:" + user_id,
        "start":0,
        "count":1
    }

    url = "https://localhost:9443/analytics/search"
    headers = {'Content-Type' : 'application/json'}
    response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=('admin', 'admin'))

    # Get the requested data from the answer message
    user_email = json.loads(response.text)[0]["values"]["user_email"]
    user_token = json.loads(response.text)[0]["values"]["user_token"]


    # Prepare the FCM notification
    registration_id = user_token
    message = "RealHolter has recently detected your body temperature has increased until " + temperature + " grades centigrades. "
    message += "We strongly recommend you to go to the nearest hospital as soon as possible!"
    message_title = "RealHolter fever Alert"

    # Send push notification to the user device
    result = push_service.notify_single_device(registration_id=registration_id, message_title=message_title, message_body=message)


    # Prepating the email notification
    fromaddr = "realholternotifications@gmail.com"
    msg = MIMEMultipart()
    msg['From'] = fromaddr
    msg['To'] = user_email
    msg['Subject'] = "RealHolter fever alert"
    body = "<h1>REALHOLTER ALERT</h1><p>Recently, your RealHolter application has detected your body temperature<strong> is over 38ºC</strong>. We strongly recommend you to go to the nearest hospital, you may have <strong>fever.</strong></p>"
    msg.attach(MIMEText(body, 'html'))

    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.starttls()
    server.login(fromaddr, "pass4daspy")
    text = msg.as_string()

    # Send email
    server.sendmail(fromaddr, user_email, text)
    server.quit()
```

Figura 58. Script Python - Método *fever_alert*