Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA CON MENCIÓN EN COMPUTACIÓN

Trabajo Fin de Grado

Evaluación de actividad física y sedentarismo a partir del fenotipo digital

ESCUELA POLITECNICA SUPERIOR

Autor: Lucas García de Viedma Pérez
Tutor: José María Gutiérrez Martínez
Cotutor/es: Antonio Artés Rodríguez

Universidad de Alcalá

Agradecimientos

Si hay alguien a quien agradecer el haber llegado hasta aquí es a mis padres, por su apoyo constante e incondicional sin importar los quebraderos de cabeza que les haya dado. Obviamente, no podrían faltar tampoco mis hermanos, que me sufren a diario y hacen lo que pueden por aguantarme. Y al resto de mi amplia y genial familia que, quien viva en Alcalá, seguro que conoce a alguno.

En lo que al TFG respecta, gracias a Antonio, por darme la oportunidad de realizar un proyecto que me interesase de verdad, a todo el equipo de EB2, por los buenos (aunque cortos) momentos que he pasado con ellos y por su ayuda desinteresada, y a José María, por acoger el proyecto y por sus consejos. La verdad es que sin todos ellos no habría sido posible.

Desde que comencé este camino de cuatro años he ido conociendo a distintos compañeros pero, realmente, sólo tres han formado parte de la aventura. Incluso con las múltiples manías de alguno, formamos un grupo capaz de superar todos los retos que se nos pusiesen por delante. Otras veces, sin embargo, caíamos en la tentación del ajedrez y teníamos que remar a contracorriente para compensar las horas perdidas. Sólo puedo dar gracias a los primos Miguel, Golvin y Yanguas.

Índice general

Lista de Figuras

Resumen

El sedentarismo y los bajos niveles de actividad física son dos de los mayores factores de riesgo para numerosas enfermedades crónicas de gravedad. Sin embargo, estos conceptos son difíciles de medir de manera precisa debido al gran número de datos que se requiere para ello.

Con este proyecto se pretende resolver este problema, desarrollando un sistema informático, alojado en la nube, que utiliza la huella digital de las personas para calcular estimadores realistas de su actividad y sedentarismo.

Para poder llevar a cabo el proyecto ha sido necesario el estudio del funcionamiento de bases de datos NoSQL, de las estrategias y procesos de gestión de datos en proyectos con un gran volumen de estos, del funcionamiento de Docker y Kubernetes y, finalmente, de los conceptos médicos pertinentes.

Summary

A sedentary lifestyle and low levels of physical activity are two of the greatest risk factors for many serious chronic diseases. However, these concepts are difficult to measure accurately due to the large amount of data required to do so.

The aim of this project is to solve this problem, developing an informatic system, hosted in cloud, which uses people's digital footprint to calculate realistic estimators of their activity and sedentarism.

To carry out carry out the project, it's been necessary to study how NoSQL databases work, the data management strategies and processes in projects with a large volume of these, the operation of Docker and Kubernetes and, finally, the relevant medical concepts.

Capítulo 1

Introducción

El sedentarismo y los bajos niveles de actividad física de las personas son la mayor fuente de preocupación en el mundo de la salud. El estilo de vida actual, que implica pasar un gran número de horas sentados, tanto en el entorno laboral como en periodos de ocio, es una de las razones principales por la que más de una cuarta parte de la población adulta no llega a los niveles de actividad mínimos recomendados. Las consecuencias de esta situación no son leves; de hecho, se ha comprobado que los bajos niveles de actividad y el sedentarismo están directamente relacionados con un gran aumento en el riesgo de desarrollar obesidad, diabetes, infartos y cáncer entre otras enfermedades graves. La Organización Mundial de la Salud (OMS) ha emitido numerosas recomendaciones y alertas en relación con este tema, advirtiendo de los peligros que conllevan el sedentarismo y la falta de actividad física, pero el problema está lejos de resolverse.

Uno de los posibles factores que frena este cambio hacia un modo de vida más sano es el desconocimiento de las personas acerca de su grado de sedentarismo y de su nivel de actividad, debido a los distintos sistemas de medición que existen y la gran cantidad de datos que se necesitan para obtener resultados precisos y fiables.

Durante la carrera siempre se ha hecho gran hincapié en la utilidad de la informática a la hora de procesar o gestionar enormes cantidades de datos, automatizar tareas repetitivas, digitalizar labores manuales y optimizar procesos, lo que hizo que me plantease aplicar los conocimientos adquiridos durante estos cuatro años para intentar aportar una solución al problema superior.

El uso de datos en tiempo real se utiliza cada vez más en múltiples disciplinas médicas para obtener información más precisa sobre los pacientes. Esto ha sido posible, entre otras razones, gracias a las mejoras tecnológicas en los wearables, que registran constantemente datos de la actividad de los usuarios. Gracias a estos dispositivos es posible tener acceso a una enorme cantidad de información útil, la cual puede ser puesta a disposición de profesionales de la salud o investigadores si los usuarios lo permiten.

Esta novedosa fuente de datos abre una puerta a nuevas oportunidades en la estimación de los niveles de actividad y sedentarismo. Sin embargo, si se busca diseñar una solución a gran escala, los recursos necesarios para gestionar toda la información recopilada de las personas, asegurar la accesibilidad del sistema en cualquier momento y dar el servicio más rápido posible, derivan en la necesidad de adquirir servidores físicos que sean capaces de soportar la carga. Afortunadamente, gracias a los servicios propor-

cionados por la computación en la nube, se pueden desarrollar sistemas de este calibre utilizando únicamente un ordenador.

Por tanto, el objetivo principal de este proyecto es el desarrollo de un sistema informático, alojado en la nube, que utilice los datos recopilados de los usuarios de wearables (frecuencia cardíaca, sueño, pasos y actividad) para generar indicadores de su grado de sedentarismo y nivel de actividad diario. Estos indicadores permitirían a los usuarios, así como a los profesionales sanitarios, ser conscientes de su situación actual e intervenir para mejorar su estilo de vida.

Capítulo 2

Objetivos del proyecto

El objetivo principal del trabajo es el desarrollo de un sistema que, a partir de los datos generados por las personas mediante el uso de wearables, genere indicadores del nivel de actividad física y grado de sedentarismo diarios estos. Concretamente, los indicadores que se busca obtener son los siguientes:

- Nivel de actividad física a partir de la tasa cardíaca
- Nivel de actividad física a partir de los pasos totales
- Nivel de actividad física a partir de las actividades realizadas
- Grado de sedentarismo a partir de las actividades realizadas
- Modo o estilo de vida a partir de la tasa cardíaca
- Modo o estilo de vida a partir de los pasos totales
- Modo o estilo de vida a partir de las actividades realizadas
- Cumplimiento de las recomendaciones de la OMS acerca del gasto energético según las actividades realizadas
- Cumplimiento de las recomendaciones de la OMS acerca del tiempo de ejercicio según las actividades realizadas.

Además de la funcionalidad principal, el proyecto tiene los siguientes objetivos :

- Alojamiento del sistema en la nube y su automatización dentro de un flujo de datos constante
- Optimización temporal del proceso de obtención de los indicadores
- Optimización económica del proceso de obtención de los indicadores
- Resistencia del sistema frente a volúmenes de datos inesperados

Capítulo 3

Estado del arte

En esta sección se explicarán las nociones teóricas relacionadas con el proyecto, comenzando por aquellas relacionadas con la parte médica y finalizando por las pertenecientes al campo técnico.

3.1. Actividad física y sedentarismo

Para comprender en qué se basan las distintas operaciones de obtención de los indicadores, o por qué se han utilizado determinados datos para calcularlos, es necesaria una introducción a los siguientes conceptos del campo de la medicina.

3.1.1. Gasto energético

El cuerpo humano está constantemente consumiendo energía. Se denomina índice metabólico o metabolismo a la cantidad de energía que consume un individuo en una determinada unidad de tiempo. Esta cantidad esta compuesta por dos factores principales:

- El índice metabólico basal (BMR), que consiste en la cantidad de energía que necesita el organismo para mantener sus funciones básicas.
- El gasto adicional derivado de realizar de actividades que supongan un incremento sobre el consumo energético basal.

Para cuantificar estos conceptos y poder realizar operaciones con las mediciones energéticas que se obtengan, es necesaria una unidad de medida estándar: el MET.

El MET (metabolic equivalent of task) es la unidad de medida del índice metabólico y corresponde a $(3,5\ ml\ de\ O2)/(kg * min)$*, que es el consumo mínimo de oxígeno que el organismo necesita para mantener sus constantes vitales.*

$$1\ MET = 3,5\ \frac{ml\ de\ O2}{kg * min} = 1\frac{kcal}{kg * min} = 4{,}184\ \frac{kJ}{kg * min} = 1{,}162\ \frac{W}{kg}$$

Este estándar se emplea para medir y comparar el consumo energético de distintas actividades, ya que representa la razón entre el metabolismo de una persona durante la realización de una actividad y su metabolismo basal. Un MET también se define como el coste energético de estar sentado tranquilamente en silencio (es decir, en estado de reposo y consumiendo únicamente la energía necesaria para el BMR) durante un minuto y es equivalente a un consumo de 1 kcal/(kg*min). Una actividad cuyo coste estimado sea de 2 METs implicará un gasto del doble de la energía utilizada al estar sentados en silencio.

Cálculo de los METs diarios a partir de actividades

Gracias a este sistema de medición, y a numerosos estudios acerca del campo, se ha llegado a un consenso del número METs al que equivale la realización de cada actividad, creando así el "Compendium of Physical Activities" [1], que se va actualizando periódicamente.

Esto es de gran utilidad ya que, si se tiene conocimiento de las actividades realizadas por un individuo en un día (y de su duración), se pueden utilizar las equivalencias del "Compendium" para obtener una estimación del numero de METs que ha consumido este.

Ejemplo. *El usuario se despierta a las 7:00, se prepara para ir al trabajo hasta las 7:30 y a esa hora se va andando tranquilamente hasta llegar a su oficina (8:00). Hace 6 horas de trabajo sedentario (sentado en una silla), y a las 14:00 se desplaza al comedor para comer. Pasa allí un total de una hora, tras lo cual vuelve a su despacho y finaliza su jornada laboral (hasta las 17:00). Vuelve a casa andando y llega a las 17:30. Descansa en el sofá hasta las 19:00, hora a la que sale a correr durante 45 minutos. Se da un baño de media hora tras volver de correr y al salir, pide algo de cena a domicilio mientras ve la televisión en el sofá. Al cabo de una hora llega la cena y se la toma también sentado en el sofá (30 minutos). Al acabar, se lava los dientes y se acuesta en la cama a leer un libro durante 1 hora, tras lo cual se duerme.*

Procedimiento. *Según las actividades realizadas por el usuario podemos estimar: i) 8h y 15m de sueño - 0.9 METs/min - 445.5 METs totales, ii) 1h de almuerzo y 8h de trabajo sedentario - 1.5 METs/min - 810 METs totales, iii) 4h y 30m de ocio sedentario - 1 METs/min - 270 METs totales, iv) 1h de marcha tranquila - 3.5 METs/min - 216 METs totales, v) 45m de carrera intensa - 10 METs/min - 450 METs totales, vi) 30m preparacion previa al trabajo (lavarse, vestirse, etc.) - 2.25 METs/min - 67.5 METs totales.*

Conclusión. *A raíz de estas estimaciones podemos suponer un gasto de 2259 METs en el día.*

Cálculo de los METs diarios a partir de tasa cardíaca

Con la intención de generar nuevos métodos precisos para la estimación de la energía utilizada por las personas en el día a día, múltiples estudios han buscado distintas mediciones fisiológicas que sirvan como predictoras de los METs, como por ejemplo la tasa cardíaca.

Bragada et al. [2] demuestran en sus trabajos que la tasa cardíaca neta (NetHR =

medición actual de la tasa cardíaca - medición de la tasa cardíaca en reposo) es un estimador eficiente del gasto energético realizado, a través de la siguiente fórmula:

$$METs = 1{,}265780 + 0{,}109479 * NetHR$$

A través de esta ecuación podemos aproximar el numero de METs utilizados por un usuario en un intervalo temporal calculando, a partir de su tasa cardíaca, el número de METs que este consume en cada minuto del intervalo y sumándolos.

Ejemplo. *Se han obtenido de un usuario las siguientes mediciones de su tasa cardíaca en un intervalo de 5 minutos : 67, 75, 80, 80 y 76 latidos por minuto. A través de estudios previos se ha determinado que su tasa cardíaca en reposo es de aproximadamente 65 latidos por minuto.*

Procedimiento. *Utilizando la formula mencionada previamente podemos seguir dos aproximaciones:*

i) Calcular los METs de cada minuto y sumarlos:
*METs intervalo = [1.265780 + 0.109479 * (67 - 65)] + [1.265780 + 0.109479 * (75 - 65)] + [1.265780 + 0.109479 * (80 - 65)] + [1.265780 + 0.109479 * (80 - 65)] + [1.265780 + 0.109479 * (76 - 65)] = 12.13 METs*

ii) Calcular la tasa cardíaca media de los minutos y calcular el NetHR con ella. De esta forma, obtenemos los METs medios consumidos por minuto y, multiplicándolos por el numero de minutos utilizados, obtenemos el total:
*METs intervalo = [1.265780 + 0.109479 * ((67 + 75 + 80 + 80 + 76)/5 - 65)] * 5 = 12.13 METs*

Conclusión. *A raíz de estas estimaciones podemos suponer un gasto de 12.13 METs en el intervalo.*

Cálculo del nivel de actividad física diario

Ya se han mostrado dos formas de obtener los METs consumidos por un individuo a lo largo de un día, pero lo que realmente interesa para el proyecto es ver el nivel de actividad física que tiene este.

El PAL (physical activity level) es un valor numérico que expresa la actividad física que realiza una persona en un día y que permite estimar su gasto o consumo energético. Este valor se define como el TEE (total energy expenditure o gasto energético total) de una persona, en un periodo de 24h, dividido por su BMR.

$$PAL = \frac{TEE(24h)}{BMR(24h)}$$

Recordemos que los METs representan el gasto energético de una persona - es decir, el TEE - y que 1 MET era el valor asignado al coste energético de un individuo en reposo - es decir, el BRM -. Gracias a estas equivalencias podemos obtener el PAL de alguien dividiendo el total de METs que haya consumido en un día entre los METs que hubiese gastado si hubiese permanecido en reposo.

$$PAL = \frac{METs(24h)}{(1\ MET * 1440(minutes\ in\ 24h))}$$

En la tabla siguiente se muestran las correspondencias del PAL con distintos estilos de vida establecidos por la OMS [3], lo que permite informar a las personas acerca de su nivel de actividad física de una forma más clara que dando un valor numérico.

PAL	Ejemplo	Estilo de vida
< 1.4	Paciente con parálisis cerebral	Extremadamente inactivo
1.40 - 1.69	Trabajador de oficina que nohace ejercicio	Sedentario o levemente activo
1.70 - 1.99	Trabajador de construccion o una hora de ejercicio diario	Moderadamente activo
2.00 - 2.40	Agricultor (sin trabajo mecanizado) o dos horas de natacion diarias	Intensamente activo
> 2.40	Ciclista competitivo	Extremadamente activo

Figura 3.1: Clasificación del estilo de vida de las personas según su PAL

Ejemplo. *En el ejemplo de la obtención de los METs diarios a partir de actividades hemos obtenido que el usuario había realizado un gasto total de 2259 METs, pero ahora podemos obtener su PAL, que es lo realmente interesante*

Procedimiento. *Siguiendo la fórmula superior calculamos: PAL = 2259/1400 = 1.56*

Conclusión. *Se ha obtenido un PAL de 1.56. Si observamos los valores de la Figura 3.1 podemos ver como estos valores corresponden con un modo de vida sedentario o con actividad ligera; lo cual tiene bastante sentido si tenemos en cuenta que el individuo pasó sentado, o reclinado, aproximadamente 14 de las 16 horas que estuvo despierto.*

Cálculo del grado de sedentarismo diario

Otro de los valores que es posible obtener, a raíz de la medición del gasto energético en METs, es el grado de sedentarismo. Ya se ha hablado anteriormente del "Compendium of Physical Activities" [1], documento en el que se determina a cuantos METs equivale la realización de cada actividad y gracias al cual se ha establecido una clasificación de estas según su coste energético. Concretamente, la clasificación divide a las actividades físicas en cuatro grupos, como puede verse en la Figura 3.2.

METs	Categorías
METS ≤ 1.5	Actividades sedentarias
1.5 < METS ≤ 3.0	Actividades de esfuerzo ligero
3.0 < METS ≤ 6.0	Actividades de esfuerzo moderado
6.0 < METS	Actividades de esfuerzo intenso

Figura 3.2: Clasificación de actividades físicas según su valor de METs

Establecidas estas categorías, se denomina grado de sedentarismo (GS) al tiempo que pasa un usuario realizando actividades sedentarias con respecto al total del tiempo que pasa despierto.

$$GS = \frac{tiempo\ realizando\ actividades\ sedentarias\ en\ un\ dia}{tiempo\ despierto\ en\ ese\ dia\ concreto}$$

Al no existir un límite exacto que indique a partir de cuantas horas de actividad sedentaria un usuario es clasificado como sedentario, se ha considerado para el proyecto que si un individuo pasa más de un tercio de las horas de vigilia del día realizando actividades sedentarias se le considerará sedentario.

3.1.2. Pasos

Además de las estimaciones de actividad física y sedentarismo relacionadas con el gasto energético del día a día, otra de las técnicas principales que se utilizan para evaluar el ejercicio que hacen las personas es el conteo de sus pasos diarios.

Pese a que este procedimiento no refleja el esfuerzo realizado en actividades que no implican caminar, suele ser un buen medidor de los desplazamientos activos de las personas (considerando no activos el uso de vehículos motorizados). Además, exceptuando la bicicleta, el gimnasio y otros deportes estáticos menos frecuentes en la sociedad actual, la gran mayoría de actividades deportivas físicas tienen un factor de movimiento que incluye dar pasos que pueden contabilizarse.

Puede observarse la clasificación del modo de vida de las personas según sus pasos [4] en la Figura 3.3.

Pasos	Estilo de vida
< 5000	Sedentario
5000 - 7499	Levemente activo
7500 - 9999	Moderadamente activo
10000 - 12499	Intensamente activo
> 12500	Extremadamente activo

Figura 3.3: Clasificación del estilo de vida de las personas según sus pasos diarios

3.1.3. Contexto médico del proyecto

Tras haber definido los conceptos esenciales, es interesante dar una breve explicación, un poco más a fondo que en la introducción, del contexto en el que se desarrolla el proyecto.

Las enfermedades crónicas (EC) son enfermedades no contagiosas, generalmente de larga duración, que progresan lentamente y que habitualmente son resultado de la genética, impacto del entorno o un estilo de vida deficiente. En la actualidad, estas enfermedades causan cerca de 39 millones de muertes al año [5], equivalentes a un 72 % de las muertes totales, lo que las convierte en el máximo exponente de la mortalidad mundial.

Dos de los mayores factores de riesgo para desarrollar enfermedades crónicas son la falta de actividad física (FA) y el sedentarismo (SB). Pese a que pueden parecer sinónimos, cada uno hace referencia a conceptos distintos. Se denomina comportamiento sedentario

[6] a toda actividad en la que un sujeto despierto esté sentado, reclinado o tumbado con un gasto energético menor a 1.5 METs. Por tanto, se puede definir el sedentarismo como la proporción de tiempo del día que un individuo pasa en comportamientos sedentarios. Por otro lado, para definir la inactividad física [6] lo que se evalúa es si el gasto energético del conjunto de actividades realizadas a lo largo de un periodo de tiempo es inferior a las recomendaciones de actividad físicas establecidas. Si bien es cierto que un bajo nivel de actividad puede implicar también un elevado grado de sedentarismo, esta relación no tiene por qué darse a la inversa.

Históricamente se ha atribuido un mayor impacto a la FA como causa de enfermedades crónicas - de acuerdo con el último informe global de la OMS (2014) [7], a la falta de actividad física se le atribuyen un total de 3.2 millones de muertes y 69.3 millones de DALYs (años perdidos por enfermedad, discapacidad o muerte prematura) cada año - y se ha tratado al SB como una consecuencia derivada de esta. Pero, recientemente, distintos estudios han demostrado que el sedentarismo es un factor independiente para numerosas EC como, por ejemplo, las enfermedades pulmonares obstructivas crónicas o COPD [8].

A partir de numerosas investigaciones, se han podido establecer umbrales de actividad física a partir de los cuales disminuye enormemente el riesgo de sufrir estas enfermedades [6]. En el caso del SB estos límites son algo difusos, y se recomienda, principalmente, evitar comportamientos sedentarios siempre que se pueda. Sin embargo, en el caso de la actividad física sí que hay recomendaciones mucho más precisas: un mínimo de 150 minutos, o un gasto de 600 METs, de actividad moderada a intensa durante la semana.

Pese a estas recomendaciones, el aumento de trabajos sedentarios, el uso de medios de transporte no activos y la sustitución de actividades físicas por actividades pasivas en el ocio, entre otros factores principales, hacen que la inactividad y el sedentarismo de la población vaya en aumento llegando a cifras alarmantes. De hecho, en la actualidad, más de un 25 % de la población adulta no llega a los niveles mínimos recomendados por la OMS mencionados previamente [9].

Otro de los posibles factores que impide la concienciación de los individuos con esta situación es el desconocimiento de su nivel de actividad física y de su grado de sedentarismo. Los múltiples sistemas de medición que se han establecido, la complicación de obtener datos suficientes como para generar medidores fiables de un individuo y la variabilidad de los comportamientos de las personas hacen que determinar indicadores sea complicado.

Las estrategias de medida más habituales para cuantificar estos indicadores se basan en la utilización de cuestionarios [10][11] que recogen información auto-reportada, y que están por tanto sujetos a sesgos de recuerdo, o el uso de acelerómetros que recogen de manera fiable la actividad, pero sólo durante periodos breves de tiempo (y habitualmente en el contexto de estudios científicos). La llegada de los wearables y su reciente desarrollo tecnológico permite recoger datos precisos de los usuarios de manera pasiva, continua y completa. Pero lo más interesante es que los usuarios pueden autorizar el acceso a esta información a profesionales de la salud o investigadores, dando lugar a una nueva fuente de datos mucho más completa que los cuestionarios.

3.2. El fenotipo o huella digital

El fenotipo o huella digital es un concepto que se define como el conjunto de datos de teléfonos inteligentes y dispositivos portátiles recopilados in situ para capturar una expresión digital de los comportamientos humanos. En otras palabras, consiste en toda la información que puede recopilarse de un usuario en su interacción con dispositivos tecnológicos.

Dentro de esta información, nos interesa destacar la generada por unos dispositivos en particular: los wearables.

3.2.1. Wearables

Los wearables, tecnología vestible, tecnología corporal, ropa tecnológica, o electrónica textil, son dispositivos electrónicos inteligentes incorporados a la vestimenta o usados corporalmente como implantes o accesorios que pueden actual como extension del cuerpo o mente del usuario.

Estos dispositivos surgen por primera vez en 1500, momento en el que Peter Henlein crea pequeños relojes que eran utilizados como collares. Con paso del tiempo y el auge de la tecnología actual, los wearables han evolucionado hasta el punto de ser capaces de incluir computadores dentro de elementos tan básicos como podrían ser unos relojes o pulseras, uniéndose al resto de los dispositivos que generan una huella digital de las personas. Sin embargo, no es hasta 2009 que una compañía, Fitbit, desarrolla un wearable orientado hacia la monitorización de la salud de su portador.

Este hecho da lugar a una completa revolución, abriendo un nuevo mercado que numerosas marcas explotan y desarrollan hasta llegar a la actualidad, momento en el que los wearables son capaces de realizar mediciones de la tasa cardíaca, porcentaje de óxigeno en sangre, la calidad del sueño, etc. Todos estos datos son parte del fenotipo digital de las personas, y los usuarios pueden acceder a su información a través de las APIs.

3.2.2. APIs

Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones. Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados.

Cada marca de dispositivos wearables tiene su propia arquitectura o estructura de almacenamiento de los datos generados por sus dispositivos y, para dar acceso a los mismos, implementan las APIs. Toda esta información es privada y propiedad de los usuarios, por lo que nadie puede acceder a ella sin su consentimiento. Sin embargo, los propietarios de los datos pueden dar acceso a terceros a través de los access token.

Los access token son una cadena opaca que permite identificar a un usuario y que deja constancia de que el acceso a los datos a través de la API ha sido autorizado por este.

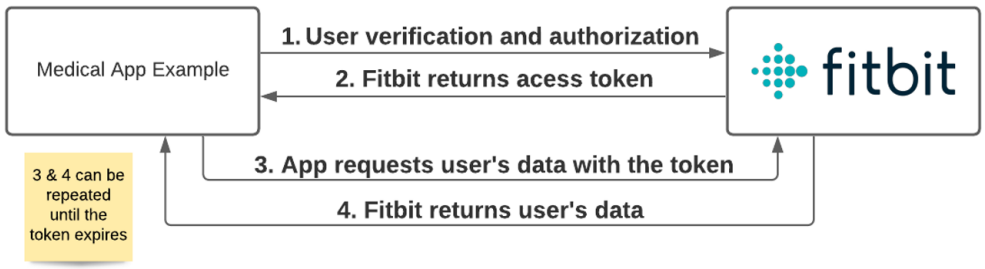

Figura 3.4: Ejemplo de la solicitud de los datos de un usuario a través de una API

De esta forma, organizaciones investigadoras o de salud pueden acceder a esta información para prestar un servicio, como se busca hacer en este proyecto.

3.3. Sistemas en la nube

A continuación, se explicarán los conceptos técnicos del proyecto relacionados con la programación en la nube y la virtualización de aplicaciones.

3.3.1. Programación en la Nube

La programación en la nube (o cloud computing) [12][13] hace referencia a la tecnología que permite dar acceso remoto a recursos informáticos a petición, desde aplicaciones hasta centros de datos, a través de Internet y con un modelo de pago según su uso. Esto presenta una serie de beneficios en comparación al uso de los recursos de ordenadores personales o servidores físicos, entre los cuales podemos encontrar: i) flexibilidad de recursos según la demanda, ii) pago únicamente de los servicios utilizados y iii) gestión ajena de los servicios adquiridos, siendo necesario únicamente preocuparse por su uso.

Existen tres tipos de nube según la accesibilidad a los servicios que incluye:

- Nube pública: todo el mundo puede acceder o adquirir sus servicios, son propiedad y están administradas por el proveedor externo y se puede acceder a ellas a través de Internet a través de un navegador web.
- Nube pública: está destinada a ser utilizada por una sola organización. Puede estar alojada en las instalaciones de su centro de datos o en las de un proveedor externa. La característica principal es que los recursos y procesos se ejecutan en una red privada y exclusiva.
- Nube pública: implementan propiedades de los tipos previos, dando las características de una nube privada para servicios confidenciales y las de una pública para servicios con una necesidad de seguridad menor.

Dentro de los servicios que ofrece el cloud computing podemos distinguir tres categorías principales: infraestructura como servicio, plataforma como servicio y software como servicio.

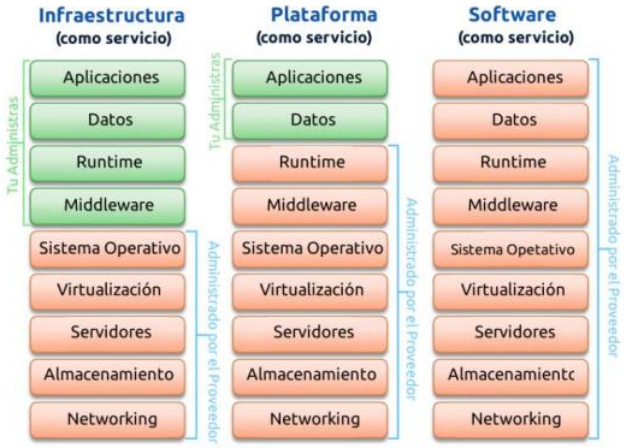

Figura 3.5: Diferencias de los servicios de cloud computing

Infraestructura como servicio

La infraestructura como servicio (IaaS) proporciona recursos informáticos, incluyendo servidores, redes, almacenamiento y espacio en centro de datos con pago en función del uso. Esto permite evitar invertir en hardware, cuyo coste puede ser inasequible para muchos proyectos, escalar los recursos si fuese necesario y seleccionar, de todos los disponibles, cuáles son los más adecuados según las necesidades. Además, no será necesario preocuparse por la disponibilidad, fiabilidad y seguridad de la infraestructura, ya que será responsabilidad del proveedor el mantenerla y asegurar estos aspectos.

Plataforma como servicio

La plataforma como servicio (PaaS) proporciona un entorno con todos los requisitos necesarios para dar soporte al ciclo de vida de creación y puesta en marcha de aplicaciones y sistemas basados en la nube, sin el coste y complejidad de comprar y gestionar el hardware, software, aprovisionamiento y alojamiento necesario. Esto habilita el desarrollo y comercialización de aplicaciones, permitiendo desplegarlas en cuestión de minutos y reduciendo la complejidad del middleware.

Es el caso del servicio que se utilizará en este proyecto, ya que utilizará los recursos de cloud para poder ejecutar el código desarrollado y gestionar la carga de información derivada del procesamiento de los datos de la huella digital de las personas.

Software como servicio

El software como servicio (SaaS) consiste en aplicaciones basadas en cloud, que se ejecutan en sistemas distantes, y que pertenecen y son administradas por otros. La característica principal de estas, es la posibilidad de conectarse a ellas a través de Internet para poder utilizar sus funcionalidades. Esto permite acceder a aplicaciones punteras sin tener problemas de compatibilidad con los equipos propios, interconectar distintas aplicaciones a través de la nube, guardar los datos en esta, etc.

3.3.2. Virtualización de aplicaciones

Generalmente, si se quiere lanzar una aplicación de manera remota, es decir, fuera de nuestro sistema operativo (SO), es necesaria una virtualización [14][15]. Este proceso consiste en la simulación, a partir del software, de algún recurso tecnológico que realmente es inexistente, haciendo creer de esta manera a la aplicación que esta operando en un sistema operativo distinto.

Este es el concepto en el que se basan las máquinas virtuales, cuyo software realiza una copia del sistema operativo que se desea emular, así como de una representación de su hardware y su plataforma. El software encargado de gestionar las máquinas virtuales y los recursos que utilizan, es el hipervisor.

Sin embargo, en la actualidad ha ganado popularidad una herramienta alternativa a estas: los contenedores [16][17]. De forma resumida, estos son un mecanismo de empaquetado lógico donde una aplicación tiene únicamente los recursos que necesita para ejecutarse. Estos requerimientos se determinan en la llamadas "imágenes de contenedor", que son los archivos que permiten crear los contenedores en cualquier momento con tiempos de ejecución de contenedores. En resumen, en vez de necesitar correr un sistema operativo entero para ejecutar una aplicación, lanzaremos esta en su contenedor, lo cual generará únicamente el gasto de recursos que necesite la aplicación.

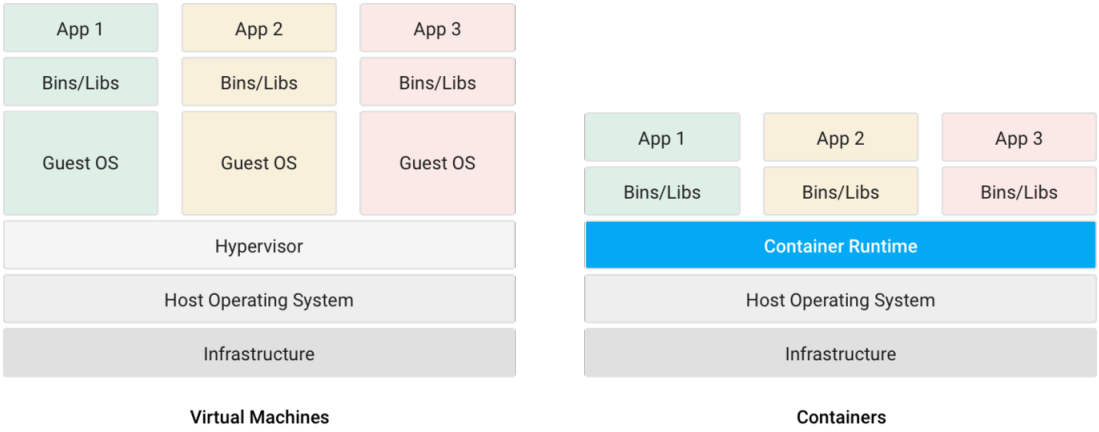

Figura 3.6: Diferencias entre las máquinas virtuales y los contenedores

Esta reducción en el gasto de recursos permitirá lanzar un numero de aplicaciones mucho mayor, cada una con su contenedor. Pero no solo esto, sino que se podrán lanzar distintos microprocesos, en sus respectivos contenedores, alojados en distintos sistemas que se comuniquen entre ellos para formar una aplicación conjunta. Gestionar tantos contenedores y su comunicación puede ser complicado, razón por la que surgen los orquestradores [18][19], herramientas que se encargan de administrar aplicaciones que están formadas por muchos contenedores, asegurándose que estos se corren en todo momento, actualizando las imágenes sin tiempo de caída de la aplicación, permitiendo la escalabilidad de la aplicación, etc.

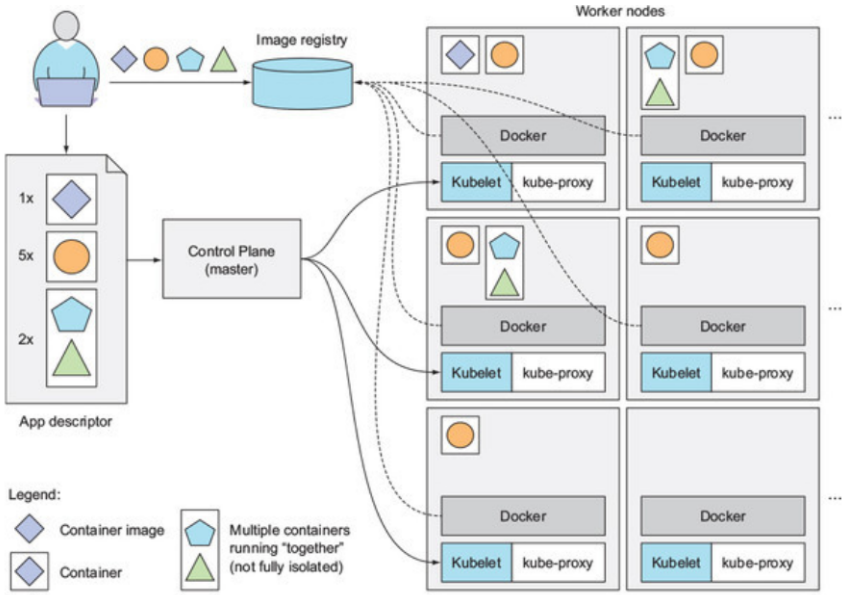

Figura 3.7: Diagrama de la orquestración de contenedores

Capítulo 4

Planteamiento del proyecto

En esta sección se especificarán las fases previas al desarrollo del sistema de obtención de indicadores, desde el origen de la idea del proyecto hasta la planificación de las fases que se llevarían a cabo para su elaboración.

4.1. Origen

Habiendo adquirido durante la carrera un gran interés por el campo de la ciencia y gestión de datos masivos, y queriendo desarrollar un TFG relacionado con este tema, surgió la oportunidad de colaborar con Evidence-Based Behavior para desarrollar un proyecto que permitiese profundizar en este ámbito.

Evidence-Based Behavior (EB2) es una empresa centrada en el estudio, análisis y cuidado de la salud mental de pacientes con determinadas condiciones psiquiátricas [20]. Para ello, utiliza técnicas de machine learning e inteligencia artificial para analizar los datos de sus pacientes, obtenidos a través de dispositivos móviles y wearables [21], y generar modelos de comportamiento que puedan ayudar a una mejor gestión de las enfermedades.

Aprovechando los grandes volúmenes de datos de usuarios de los que dispone EB2, la completitud de estos gracias al uso de wearables, sus conocimientos de herramientas de gestión de grandes volúmenes de datos, los servicios de los que disponían en la nube y la situación actual, mencionada en el contexto, se planteó desarrollar un sistema informático, alojado en cloud, que utilizase los datos obtenidos a partir de la huella digital de las personas para generar indicadores del nivel de actividad física y del sedentarismo. Este sistema permitiría, tanto a los propietarios de los datos como a profesionales de la salud e investigadores, obtener mediciones más precisas acerca del nivel de actividad y sedentarismo de las personas que las que pueden obtenerse a partir de los métodos tradicionales mencionados previamente.

4.2. Herramientas

Una vez decidido el objetivo del proyecto, era esencial determinar las herramientas que se utilizarían en su desarrollo. Estas pueden dividirse en cuatro grupos: control de versiones, almacenamiento, lenguaje de programación y alojamiento en la nube y virtualización.

4.2.1. Control de versiones

Un controlador de versiones es un sistema (o conjunto de sistemas) que permite guardar un registro del los cambios que se realizan sobre un proyecto, generalmente sobre el código fuente de este, y permite revertirlos en caso de ser necesario. Estas herramientas suelen aportan más utilidad en desarrollos colectivos, pero siempre son un beneficio frente a posibles pérdidas de información, cambios indeseados u otras eventualidades.

En este caso se decidió el uso de Git (a través de Github).

Como ya se ha mencionado, Git es un software de control de versiones que permite la gestión y el mantenimiento del código fuente de proyectos complejos. Github es una plataforma basada en la nube que aloja Git y aporta una interfaz mucho más cómoda para su uso.

4.2.2. Almacenamiento

El almacenamiento de los datos no presenta un problema cuando el volumen de estos es pequeño. Sin embargo, a medida que esta cantidad aumenta, la herramienta o sistema utilizados para su guardado es de vital importancia. Durante la carrera se ha distinguido principalmente entre dos tipos de bases de datos (BBDD): relacionales y no relacionales.

Las bases de datos relacionales se basan en la organización de los datos almacenados en tablas, las cuales se relacionan unas a otras a través de identificadores. Aportan, entre otros beneficios, atomicidad, consistencia, aislamiento y durabilidad, lo que favorece la robustez y resistencia a posibles fallos. Además, al ser el tipo de BBDD más longevas, hay un gran número de estándares y patrones accesibles para optimizar su uso y desarrollo. No obstante, cuando el volumen de datos aumenta, la dificultad y el coste de mantenimiento de estas estructuras de almacenamiento aumentan enormemente.

Por otra parte, las bases no relacionales organizan los datos en distintas estucturas según la funcionalidad a la que esten orientadas: BBDD documentales, BBDD orientadas a grafos, BBDD de clave/valor, etc. El mayor beneficio de este tipo de bases de datos es su gran versatilidad. La posibilidad de modificar la estructura de almacenamiento y la escalabilidad hacen que este tipo de BBDD sean las más utilizadas para sistemas con un gran volumen de datos.

Para el proyecto era necesaria una base capaz de soportar el almacenamiento de una gran cantidad de datos de un elevado número de usuarios, así como los resultados del sistema para cada uno de ellos, por lo que se planteó el desarrollo de una base de datos no relacional. Sin embargo, el coste monetario que se generaba por el volumen de datos

guardados no era asequible. Afortunadamente, EB2 permitió el uso de su propia base tanto para obtener directamente los datos de los usuarios, como para almacenar la salida del sistema para cada uno de estos.

Por tanto, la base utilizada consiste en una base no relacional basada en diccionarios, propiedad de EB2, que se encuentra alojada en Firebase (Cloud Firestore).

Cloud Firestore es una de las muchas herramientas de pago que implementa Firebase para el desarrollo y escalabilidad de aplicaciones. Consiste en una base NoSQL que almacena la información en documentos, los cuales están estructurados como diccionarios JSON y se organizan en colecciones, y que permite un alojamiento y disposición de los datos en tiempo real. Además, implementa un sistema de credenciales que permite el acceso a usuarios autorizados, como es este caso. Al estar almacenada en la nube, Firestore permite una completa escalabilidad, asignando más recursos a la base de datos si fuese necesario.

4.2.3. Lenguaje de desarrollo y librerías

A la hora de elegir el lenguaje de programación en el que se desarrollaría el sistema se tuvieron en cuenta los siguientes requisitos:

i) Necesidad de librerías que permitiesen realizar una conexión con Cloud Firestore de manera rápida y eficiente para evitar problemas de rendimiento por la gran cantidad de datos.

ii) Necesidad de librerías que permitiesen gestionar un gran volumen de datos y realizar operaciones sobre estos a gran velocidad para optimizar el proceso.

Tanto por ajustarse a los requisitos, como por preferencia propia, el candidato elegido para el desarrollo fue Python.

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medido, programación funcional.

Este lenguaje, además de permitir generar un código muy limpio, incluye un enorme número de librerías que satisfacen las necesidades mencionadas previamente y de las cuales se han utilizado:

- Firebase_admin

  Además de otras utilidades menos relevantes para este proyecto, habilita el establecimiento de conexiones personalizables con Firestore, permitiendo al usuario elegir entre un gran número de ajustes relativos a esta. Por ejemplo, el numero de documentos recopilados en cada conexión, la aplicación de condiciones a las consultas para seleccionar sólo los datos pertinentes, etc.

- Numpy

  Da soporte especializado para la creación de vectores y matrices multidimensionales de gran tamaño, así como una gran colección de funciones matemáticas de alto nivel para operar con ellas.

- Pandas

  Extiende numerosas funcionalidades de NumPy. Ofrece nuevas estructuras de datos (dataFrames, series y panel) y operaciones para manipular tablas numéricas y series temporales. Facilita el manejo de estructuras con grandes volúmenes de información, realizando operaciones sobre estas de manera eficiente y a gran velocidad.

Además de estas tres librerías principales, también se han utilizado otras cuya función es menos relevante y de carácter auxiliar.

El IDE elegido para trabajar con Python fue Pycharm.

4.2.4. Alojamiento en la nube y virtualización

Para alojar el sistema en la Nube, se determinó el uso de Docker y Google Kubernetes Engine.

Docker

Docker [22][23] es una tecnología de código abierto que permite automatizar la implementación de aplicaciones como contenedores portátiles autosuficientes que pueden ejecutarse en la nube o entornos locales. Estos contenedores se crean a partir de las "imagenes de conenedor" (mencionadas en el estado del arte) de Docker, llamadas "Docker Images". Y estas, a su vez, a partir de otro tipo de archivos llamados Dockerfile, que especifican el código a ejecutar y los requerimientos de este.

Esta herramienta implementa tanto una terminal a través de la cual el desarrollador puede solicitar la creación de una imagen a partir de su software, como un daemon que gestiona todos los contenedores activos.

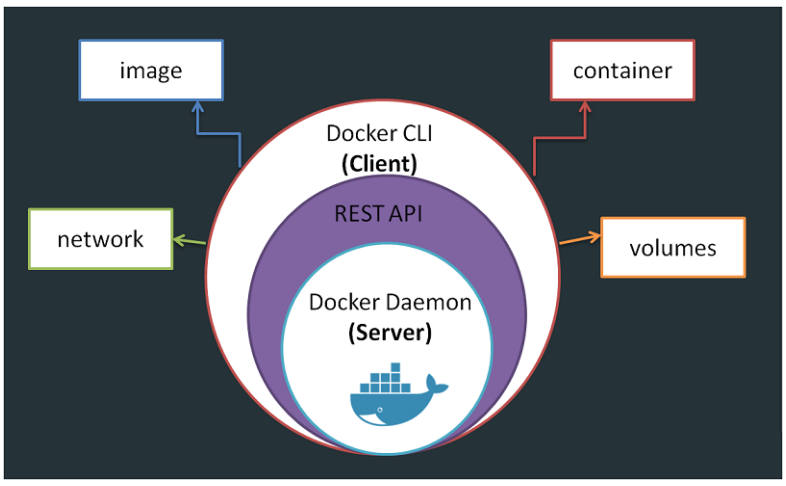

Figura 4.1: Componentes de Docker Engine

Google Kubernetes Engine

Kubernetes [24][25] es una herramienta de orquestración de contenedores que permite alojar y gestionar aplicaciones, compuestas por varios contenedores que ejecutan distintos microprocesos, en la nube.

Para ello, Kubernetes determina una serie de estructuras que utiliza para organizar y gestionar la ejecución de las aplicaciones:

- CronJob: objeto que almacena una imagen de contenedor y gestiona su ejecución.
- Pods: mínima carga de trabajo de Kubernetes. En cada pod hay un CronJob, ya que son los encargados de lanzar estos.
- Nodos: entorno en el que se lanzan los pods. Pueden ser máquinas virtuales o físicas, dependiendo del cluster.
- Cluster: plataforma de Google en la que se corre Kubernetes. Estos están compuestos por nodos. En cada cluster tiene que haber un nodo maestro, que gestiona y planifica la ejecución de los pods de todos los nodos del cluster, y nodos esclavos, que es donde se ejecutan los pods que lanzan los contenedores.
- Controlador: objetos que ejecutan ciclos de control. Hay muchos controladores distintos, dependiendo de su función concreta, pero su labor general consiste en controlar que el sistema se esté ejecutando correctamente.

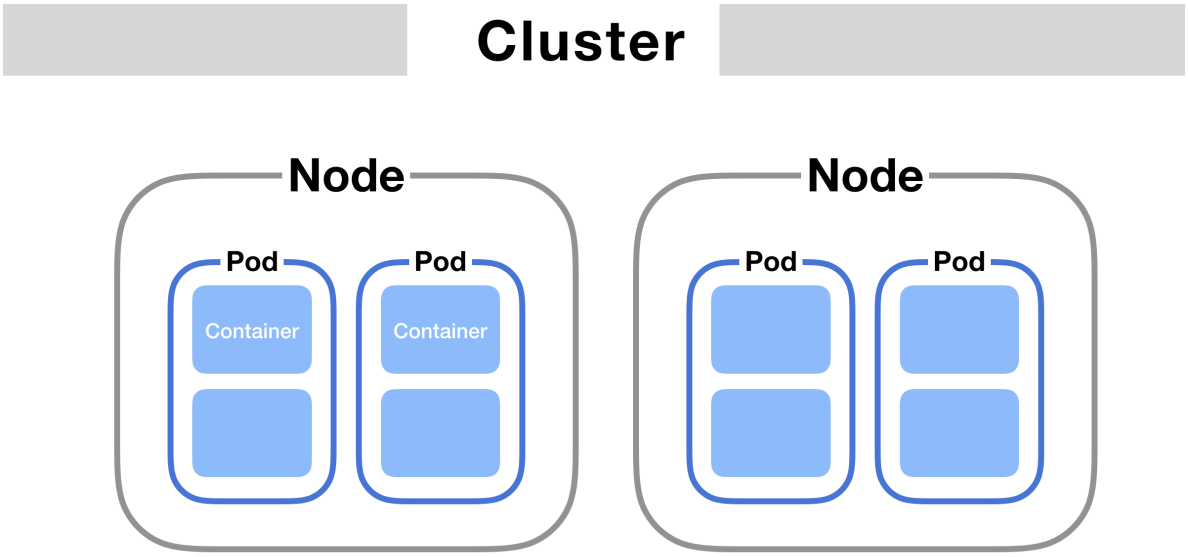

Figura 4.2: Diagrama de la estructura de Google Kubernetes Engine

Unión de ambos

Docker y Kubernetes se utilizan juntos en numerosas ocasiones ya que sus servicios se complementan a la perfección y permiten el alojamiento de aplicaciones de una manera relativamente sencilla [26][27][28]. Con Docker se generan, a partir del código de los

desarrolladores, las imágenes que conformarán los contenedores. Tras ello, se envían, desde Docker al "registro de contenedores" del proyecto de Kubernetes, las imágenes de contenedores que queramos. Se establece la estructura del proyecto de Kubernetes que se desee y se guardan las imágenes de los contenedores en los pods. Una vez hecho esto, se lanza la aplicación.

Si se quisiese actualizar el código o realizar cambios sobre este, bastará con crear una nueva imagen del código modificado y cargarla en el pod encargado de ejecutarla.

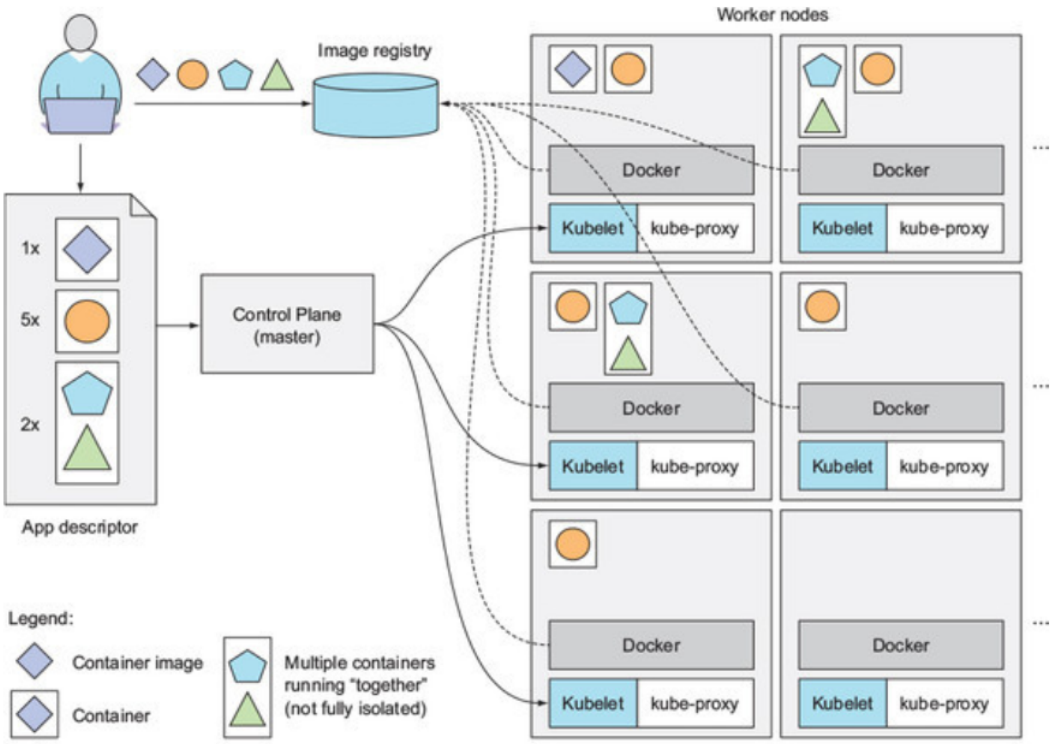

Figura 4.3: Diagrama de la conexión habitual entre Docker y Kubernetes

4.3. Planificación

Tras establecer las bases del proyecto y las herramientas a utilizar, se planificó el desarrollo del sistema dividiéndolo en cinco etapas principales:

i) Familiarización con el campo y estudio del estado del arte en la medición del nivel de actividad física y del sedentarismo de las personas.

ii) Análisis de las fuentes de datos disponibles y selección de los indicadores que pueden obtenerse a partir de estas.

iii) Diseño de la estructura del sistema, así como de las estructuras de datos necesarias para su funcionamiento.

iv) Desarrollo de las diferentes partes del sistema y la posterior unión de estas para conformar el sistema completo.

v) Prueba del sistema con datos reales para validar el funcionamiento del mismo.

Además, la metodología que se estableció para confección del sistema fue Scrum.

Scrum es un marco de trabajo para desarrollo ágil de software que se ha expandido a otras industrias. Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo y obtener el mejor resultado posible de proyectos

Capítulo 5

Descripción detallada del sistema

El sistema se ha desarrollado siguiendo una estructura que divide el proceso en cinco fases de gestión de datos - obtención, estructuración, agregación, imputación y procesamiento - y distingue entre cuatro campos de datos - actividades, tasa cardíaca, sueño y pasos -. El objetivo de esta distinción es permitir aplicar las fases de gestión que sean necesarias sobre los distintos campos de datos por separado. Por ejemplo, obtención y estructuración sobre el campo de sueño, obtención y agregación sobre el campo de pasos, etc. La Figura 5.1 muestra de manera gráfica, y mucho más clara, la estructura que sigue el sistema.

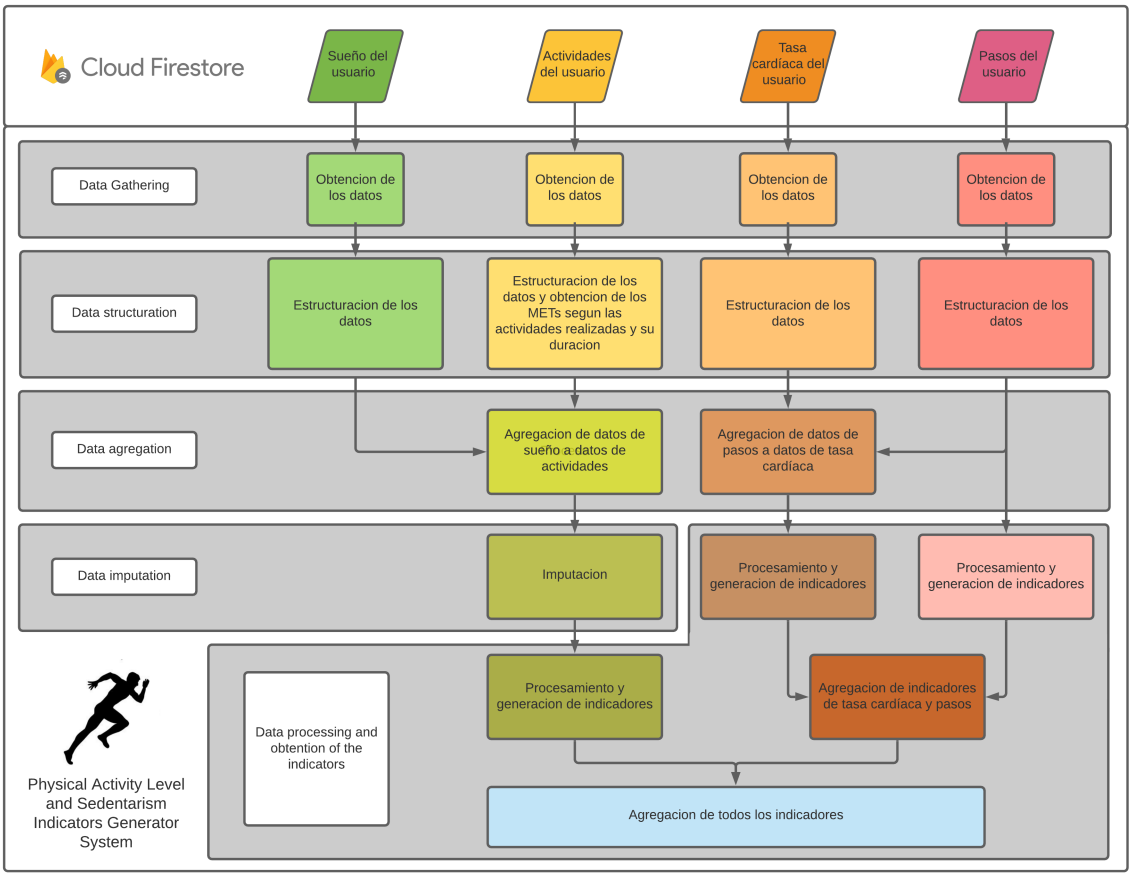

Figura 5.1: Diagrama que representa la estructura del sistema

Para la descripción del sistema se seguirá esta misma estrategia, por lo que se irán explicando las fases de gestión y, dentro de cada una de estas, se detallarán las funciones relativas a los distintos campos de datos.

5.1. Fuente de los datos

Antes de entrar en la explicación del desarrollo, es importante indicar la fuente y el formato inicial de los datos que se utilizan en el sistema.

Como se ha dicho previamente, los datos se recogen de la base de datos Cloud Firestore de EB2. Para evitar dudas en las explicaciones de las funciones que interactuan con esta, se hará una breve explicación de su funcionamiento y estructura.

Cloud Firestore [29] es una base de datos NoSQL orientada a los documentos. Esto significa que, en vez de usar tablas o filas, almacena la información en documentos que se organizan en colecciones. Cada documento contiene un conjunto de pares clave-valor (diccionario JSON) y permite incluir como valores estructuras de datos como nuevos pares clave-valor o matrices, generando documentos más complejos.

En concreto, se van a utilizar un total de cuatro colecciones de la base de EB2: sueño, actividades, tasa cardíaca y pasos. En cada una de estas colecciones se guarda, diariamente, un documento por cada usuario, que incluye la información de este relativa al concepto de la colección. Para esclarecer esta explicación, se describe a continuación el formato de los documentos que se guardan en cada colección utilizada.

5.1.1. Actividad

Esta colección almacena documentos diarios con la información relativa a las actividades del usuario. En los siguientes puntos y en la Figura 5.2 se muestra la estructura de estos documentos y se describe brevemente la información que contienen.

- User. Nombre de usuario (string)
- User_uid. Id de usuario (string)
- Date_time. Fecha de la recogida de datos (string)
- Service. Servicio del usuario (string)
- Creation_datetime. Fecha de creación de la colección de datos (timestamp)
- Last_mod_datetime. Fecha de modificación de la colección de datos (timestamp)
- Still. Matriz con el nombre de la marca del "wearables" que ha recogido los datos. Esta formada por un total de 48 "slots", que representan los intervalos de 30 minutos de un día y que almacenan el número de pasos que ha dado el usuario en cada uno.
- Walking. Matriz que almacena los segundos que el usuario ha pasado caminando. Esta formada por un total de 48 "slots", que representan los intervalos de 30 minutos de un día y que almacenan el número de segundos de esta actividad que ha realizado el usuario en cada uno.

Field name	Field_type	Subfields	Subfields_type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Creation_datetime	Timestamp		
Last_mod_datetime	Timestamp		
Still	Matrix	0	Number
		...	
		47	
Walking	Matrix	0	Number
		....	
		47	
Running	Matrix	0	Number
		...	
		47	
Biking	Matrix	0	Number
		...	
		47	
Sport	Matrix	0	Number
		..	
		47	
Vehicle	Matrix	0	Number
		...	
		47	

Figura 5.2: Estructura de los documentos de la colección de actividad

- Running. Matriz que almacena los segundos que el usuario ha pasado corriendo. Esta formada por un total de 48 "slots", que representan los intervalos de 30 minutos de un día y que almacenan el número de segundos de esta actividad que ha realizado el usuario en cada uno.

- Biking. Matriz que almacena los segundos que el usuario ha pasado montando en bicicleta. Esta formada por un total de 48 "slots", que representan los intervalos de 30 minutos de un día y que almacenan el número de segundos de esta actividad que ha realizado el usuario en cada uno.

- Sport. Matriz que almacena los segundos que el usuario ha pasado haciendo ejercicios no registrados por EB2. Esta formada por un total de 48 "slots", que representan los intervalos de 30 minutos de un día y que almacenan el número de segundos de estas actividades que ha realizado el usuario en cada uno.

- Vehicle. Matriz que almacena los segundos que el usuario ha pasado montando en coche. Esta formada por un total de 48 "slots", que representan los intervalos de 30 minutos de un día y que almacenan el número de segundos de esta actividad que ha realizado el usuario en cada uno.

- Etc. Hay más actividades posibles, pero su formato es el mismo. Se han incluido las más comunes.

5.1.2. Tasa cardíaca

Esta colección almacena documentos diarios con la información relativa a la tasa cardíaca del usuario. En los siguientes puntos y en la Figura 5.3 se muestra la estructura de estos documentos y se describe brevemente la información que contienen.

Field name	Field_type	Subfields	Subfields_type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Creation_datetime	Timestamp		
Last_mod_datetime	Timestamp		
Source_name	Matrix	value	Number
		timestamp	Timestamp

Figura 5.3: Estructura de los documentos de la colección de tasa cardíaca

- User. Nombre de usuario (string)
- User_uid. Id de usuario (string)
- Date_time. Fecha de la recogida de datos (string)
- Service. Servicio del usuario (string)
- Creation_datetime. Fecha de creación de la colección de datos (timestamp)
- Last_mod_datetime. Fecha de modificación de la colección de datos (timestamp)
- Source_name. Matriz con el nombre de la marca del "wearable" que ha recogido los datos. Esta formada por un total de N "slots", que representan cada medición de la tasa cardíaca que se ha realizado durante el día y que almacenan el valor registrado (number) y la fecha/hora a la que se ha tomado (timestamp).

5.1.3. Sueño

Esta colección almacena documentos diarios con la información relativa a los patrones de sueño del usuario. En los siguientes puntos y en la Figura 5.4 se muestra la estructura de estos documentos y se describe brevemente la información que contienen.

- User. Nombre de usuario (string)
- User_uid. Id de usuario (string)
- Date_time. Fecha de la recogida de datos (string)
- Service. Servicio del usuario (string)
- Creation_datetime. Fecha de creación de la colección de datos (timestamp)

Field name	Field_type	Subfields	Subfields_type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Creation_datetime	Timestamp		
Last_mod_datetime	Timestamp		
Source_name	Map	Start	Timestamp
		End	Timestamp
		Mid_sleep	Timestamp
		Spt	Number
		N_awakenings	Number
		waso	Number
		tst	Number
		se	Number
		tib	Number

Figura 5.4: Estructura de los documentos de la colección de sueño

- Last_mod_datetime. Fecha de modificación de la colección de datos (timestamp)
- Source_name. Diccionario con el nombre de la marca del "wearable" que ha recogido los datos con los siguientes valores:
  - Start. Fecha de inicio del intervalo de sueño (timestamp)
  - End. Fecha de fin del intervalo de sueño (timestamp)
  - Mid-sleep. Punto medio desde el inicio del sueño hasta despertar (timestamp)
  - Spt. Duración del intervalo de sueño (number)
  - N_awakenings. Número de despertares detectados (number)
  - waso. Tiempo despierto en el intervalo de sueño (number)
  - tst. Tiempo dormido en el intervalo de sueño (number)
  - se. Tiempo dormido con respecto al total (number)
  - tib. Tiempo pasado en cama en el intervalo de sueño (number)

5.1.4. Pasos

Esta colección almacena documentos diarios con la información relativa a los pasos del usuario. En los siguientes puntos y en la Figura 5.5 se muestra la estructura de estos documentos y se describe brevemente la información que contienen.

- User. Nombre de usuario (string)
- User_uid. Id de usuario (string)
- Date_time. Fecha de la recogida de datos (string)

Field name	Field_type	Subfields	Subfields_type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Creation_datetime	Timestamp		
Last_mod_datetime	Timestamp		
Source_name	Matrix	0	Number
		...	
		47	
Steps	Matrix	0	Number
		...	
		47	

Figura 5.5: Estructura de los documentos de la colección de pasos

- Service. Servicio del usuario (string)
- Creation_datetime. Fecha de creación de la colección de datos (timestamp)
- Last_mod_datetime. Fecha de modificación de la colección de datos (timestamp)
- Source_name. Matriz con el nombre de la marca del "wearable" que ha recogido los datos. Esta formada por un total de 48 "slots", que representan los intervalos de 30 minutos de un día y que almacenan el número de pasos que ha dado el usuario en cada uno.
- Steps. Matriz que agrega todos los pasos registrados por los distintos dispositivos. Esta formada por un total de 48 "slots", que representan los intervalos de 30 minutos de un día y que almacenan el número de pasos que ha dado el usuario en cada uno.

5.2. Obtención de los datos - data gathering

Las funciones pertenecientes a esta fase (Figura 5.6) son las encargadas de recopilar los documentos de las colecciones de Firestore para poder utilizar sus datos en el sistema.

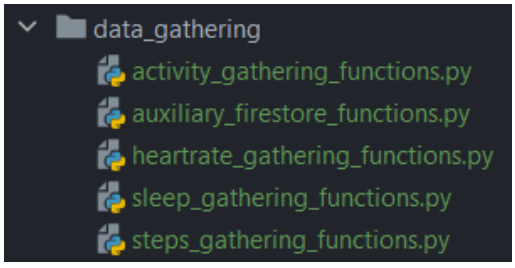

Figura 5.6: Archivos con las funciones de obtención

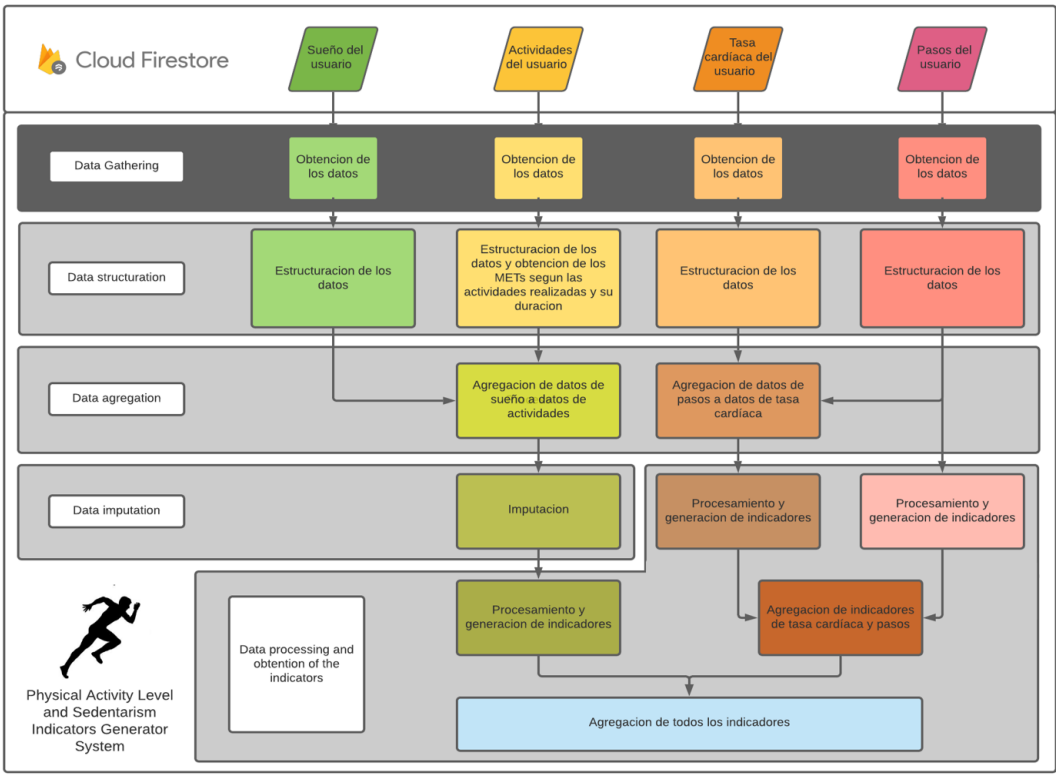

Figura 5.7: Diagrama que representa la estructura del sistema: fase de obtención

5.2.1. auxiliary_firestore_functions.py

En este archivo se encuentran las cinco funciones auxiliares que permiten intercambiar datos con la base de datos de Firestore.

Establish_firestore_credentials

- Parámetros
  - Project_name (string): identificador del proyecto de Firebase al que queremos conectarnos.
- Return
  - void

La función dota de los permisos y credenciales necesarias para establecer una conexión con un proyecto de Firebase al proceso que la lanza.

Get_data_from_collection

- Parámetros

  - Collection_name (string): nombre de la colección de Firestore de la cual queremos obtener los documentos.
  - Date_time (string): fecha de los documentos que queremos obtener.

- Return
  - Complete_data_list (lista): lista con documentos de una colección de Firestore.

La función crea, por cada usuario cuyos datos queramos utilizar posteriormente, una query de Firestore para solicitar los documentos de este de un día concreto. Tras ello, llama a la función recursive_firestore_query para ejecutarla y devuelve el resultado obtenido.

Recursive_firestore_query

- Parámetros
  - Collection_instance (firestore query): objeto que representa una query de Firestore a una colección.
  - Step (string): numero máximo de documentos que pueden leerse por ejecución de la query.
  - Cursor (string): posición del último documento de la colección que ha leído la query.

- Return
  - Complete_data_list (lista): lista con documentos de una colección de Firestore.

La función ejecuta la query recibida, tras lo cual, lee los documentos resultantes de la ejecución hasta llegar al máximo especificado por steps. Una vez llegue, los almacenará en una lista.

Si, una vez guardados estos documentos, no se ha terminado de leer todos los documentos, la función se llama a si misma con los mismos parámetros pero indicando el punto de la lectura en la que se ha quedado (cursor), para que la siguiente iteración de la función empiece desde ahí. Tras ello, añade el resultado de las ejecuciones recursivas a la lista en la que se habían guardado los documentos previos y devuelve esta como resultado. Si se se hubiesen leído todos los documentos, simplemente devolverá directamente la lista como resultado sin llamadas recursivas.

Check_document_exist_firestore

- Parámetros
  - Doc_ref: objeto de referencia a un documento de Firestore.

- Return
  - (Boolean)

La función devuelve si el documento existe o no en la colección establecida previamente.

Upload_set_firestore_new

- Parámetros
  - Doc_ref: objeto de referencia a un documento de Firestore.
  - Set_data: diccionario con los indicadores del día del usuario.
  - Upload_data: diccionario con los indicadores del día del usuario.
- Return
  - (Boolean)

La función comprueba si el documento existe o no en la colección establecida previamente. En caso de que no exista, crea el documento y añade en este la información de Set_data. Si existiese, sustituye su información por upload_data.

5.2.2. activity_gathering_functions.py

En este archivo se encuentra la función encargada de recopilar los documentos de los usuarios con información acerca de su actividad. Para ello, utiliza las funciones definidas en auxiliary_firestore_functions.py.

Gather_users_activity_data

- Parámetros
  - Date_time (string): fecha del día cuyos documentos queremos recuperar.
- Return
  - Users_activity_data_list (lista): lista con documentos de la colección Actividad (Figura 5.2) de Firestore.

La función devuelve el resultado de llamar a get_data_from_collection pasándole como parámetros el nombre de la colección de actividad y la fecha del día cuyos datos queremos.

5.2.3. heartrate_gathering_functions.py

En este archivo se encuentra la función encargada de recopilar los documentos de los usuarios con información acerca de su tasa cardíaca. Para ello, utiliza las funciones definidas en auxiliary_firestore_functions.py.

gather_users_heartrate_data

- Parámetros
  - Date_time (string): fecha del día cuyos documentos queremos recuperar.
- Return
  - Users_heartrate_data_list (lista): lista con documentos de la colección Tasa cardíaca (Figura 5.3) de Firestore.

La función devuelve el resultado de llamar a get_data_from_collection pasándole como parámetros el nombre de la colección de tasa cardíaca y la fecha del día cuyos datos queremos.

5.2.4. sleep_gathering_functions.py

En este archivo se encuentra la función encargada de recopilar los documentos de los usuarios con información acerca de sus patrones de sueño. Para ello, utiliza las funciones definidas en auxiliary_firestore_functions.py.

gather_users_sleep_data

- Parámetros
  - Date_time (string): fecha del día cuyos documentos queremos recuperar.
- Return
  - Users_sleep_data_list (lista): lista con documentos de la colección Sueño (Figura 5.4) de Firestore.

La función devuelve el resultado de llamar a get_data_from_collection pasándole como parámetros el nombre de la colección de sueño y la fecha del día cuyos datos queremos.

5.2.5. steps_gathering_functions.py

En este archivo se encuentra la función encargada de recopilar los documentos de los usuarios con información acerca de sus pasos. Para ello, utiliza las funciones definidas en auxiliary_firestore_functions.py.

gather_users_steps_data

- Parámetros
  - Date_time (string): fecha del día cuyos documentos queremos recuperar.
- Return
  - Users_steps_data_list (lista): lista con documentos de la colección pasos (Figura 5.5) de Firestore.

La función devuelve el resultado de llamar a get_data_from_collection pasándole como parámetros el nombre de la colección de pasos y la fecha del día cuyos datos queremos.

5.3. Estructuración de los datos - data structuration

Las funciones pertenecientes a esta fase (Figura 5.16) son las encargadas de seleccionar qué datos de entre todos los recopilados en la fase de obtención son útiles para el proceso y de darles el formato apropiado (dataframes) para su futuro procesamiento.

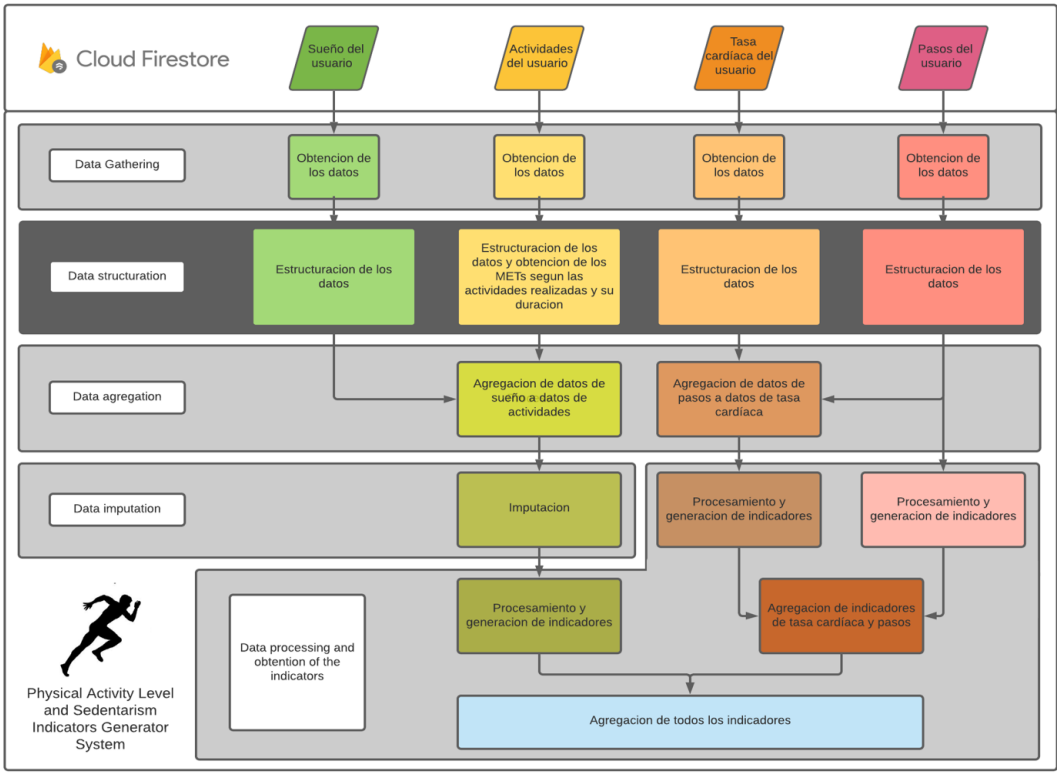

Figura 5.8: Diagrama que representa la estructura del sistema: fase de estructuración

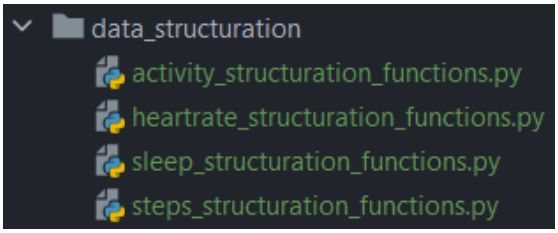

Figura 5.9: Archivos con las funciones de estructuración

5.3.1. activity_structuration_functions.py

En este archivo se encuentran las funciones encargadas de seleccionar y reestructurar los datos de actividad de los usuarios y de calcular el gasto en METs del usuario (total y en cada intervalo de 30 minutos del día).

structure_users_activity_data_and_get_METs_from_activities

- Parámetros
  - users_activity_data_list (lista): lista con documentos de la colección Actividad (Figura 5.2) de Firestore.
- Return
  - users_activity_dataframe (dataframe): dataframe con la información necesaria de la actividad de cada usuario.

La función reestructura los datos de actividad y calcula los valores de METs requeridos por el sistema. Para ello, por cada documento recibido:

Agrupa las actividades registradas en el documento según su nivel de mets - sedentarias, ligeras, moderadas e intensas - y calcula el tiempo que se ha pasado realizando actividades de cada nivel en cada intervalo de 30 minutos del día. Además, también calcula el gasto de METs total en cada intervalo de 30 minutos del día y el gasto de METs total de cada nivel de actividad en el día. Con estos datos, calcula otros valores útiles para generar los indicadores, como el tiempo y METs consumidos en actividades intensas y moderadas.

Una vez se tiene toda la información necesaria (incluyendo los valores identificadores del documento como user, uid, service y date_time), agrupa esta en una nueva lista, que será la nueva estructura de los datos del usuario (Figura 5.10).

De la misma manera, todas las listas generadas a partir de los documentos de los usuarios se agrupan en otra lista más, que se utilizará para generar el dataframe que se devolverá como resultado.

Field name	Type	Subfields	Type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Sa_duration_slots	Matrix	0 ... 47	Number
La_duration_slots	List	0 ... 47	Number
Ma_duration_slots	List	0 ... 47	Number
Ia_duration_slots	List	0 47	Number
Ma_ia_dur	Number		
Total_duration	Number		
Mets_slots	List	0 ... 47	Number
Ma_ia_mets	Number		
Total_mets	Number		

Figura 5.10: Estructura final de los datos de la actividad tras aplicar la función

get_activity_type_duration_and_met_slots

- Parámetros
  - day_data (firestore document): documento de la colección Actividad (Figura 5.2) de Firestore.
  - activity_type (string): nivel de intensidad de la actividad - sedentary, light, moderate, intense - segun su coste en METs (Figura 3.2).
  - total_activities_METs_30m_slots (lista): lista con 48 slots que alamacena los METs consumidos por las actividades realizadas en cada intervalo de 30 minutos del día.
- Return
  - current_activity_type_duration_30m_slots (lista): lista con 48 slots que alamacena cuanto tiempo se ha empleado en actividades del nivel de intensidad especificado en cada intervalo de 30 minutos del día.
  - total_activities_METs_30m_slots (lista): lista con 48 slots que alamacena los METs consumidos por las actividades realizadas en cada intervalo de 30 minutos del día.

  - current_activity_type_total_met_value (number): total de METs consumidos ese día en actividades del nivel de intensidad especificado.

La función resume la información de las actividades que pertenezcan al nivel de intensidad especificado en activity_type. Para ello genera una nueva lista en la que incluirá el tiempo total que ha pasado el usuario realizando actividades de ese nivel en cada intervalo de 30 minutos del día, y otra que almacene el gasto en METs correspondiente de cada intervalo. Finalmente, calcula cuantos METs totales se han consumido en el día en estas actividades y devuelve este valor junto a las dos listas previas.

string_to_timestamp

- Parámetros
  - date_string (string): cadena que representa una fecha en formato YYYY-MM-DD
- Return
  - date_timestamp (timestamp): valor numérico que representa una fecha

Esta función transforma una fecha en un timestamp con la librería datetime.

5.3.2. heartrate_structuration_functions.py

En este archivo se encuentran las funciones encargadas de seleccionar y reestructurar los datos de tasa cardíaca de los usuarios.

structure_users_heartrate_data

- Parámetros
  - users_heartrate_data_list (list): lista con documentos de la colección Tasa cardíaca (Figura 5.3) de Firestore.
- Return
  - users_heartrate_dataframe (dataframe): dataframe con la información necesaria de la tasa cardíaca de cada usuario

La función reestructura los datos de tasa cardíaca. Para ello, por cada documento recibido:

Se obtienen las mediciones de tasa cardíaca de la fuente de datos prioritaria (la prioridad de las fuentes de datos está establecida en el código) del documento y se guardan en una lista.

Una vez se tiene toda la información necesaria (incluyendo los valores identificadores del documento como user, uid, service y date_time), agrupa esta en una nueva lista, que será la nueva estructura de los datos del usuario (Figura 5.11).

De la misma manera, todas las listas generadas a partir de los documentos de los usuarios se agrupan en otra lista más, que se utilizará para generar el dataframe que se devolverá como resultado.

Field name	Field_type	Subfields	Subfields_type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Hr_slots	List	value	Number
		timestamp	Timestamp

Figura 5.11: Estructura final de los datos de la tasa cardíaca tras aplicar la función

string_to_timestamp

- Parámetros
  - date_string (string): cadena que representa una fecha en formato YYYY-MM-DD
- Return
  - date_timestamp (timestamp): valor numérico que representa una fecha

Esta función transforma una fecha en un timestamp con la librería datetime.

5.3.3. sleep_structuration_functions.py

En este archivo se encuentran las funciones encargadas de seleccionar y reestructurar los datos de sueño de los usuarios.

structure_users_sleep_data

- Parámetros
  - users_sleep_data_list (lista): lista con documentos de la colección Sueño (Figura 5.4) de Firestore.
- Return
  - users_sleep_dataframe (dataframe): dataframe con la información necesaria del sueño de cada usuario

La función reestructura los datos de sueño. Para ello, por cada documento recibido:

Se obtienen, de la fuente de datos prioritaria (la prioridad de las fuentes de datos está estbalecida en el código) del documento, los segundos que ha pasado dormido el usuario y las fechas de inicio y fin del intervalo de sueño. Tras ello, se guardan en una lista.

Una vez se tiene toda la información necesaria (incluyendo los valores identificadores del documento como user, uid, service y date_time), agrupa esta en una lista que será la nueva estructura de los datos del usuario (Figura 5.12).

De la misma manera, todas las listas generadas a partir de los documentos de los usuarios se agrupan en otra lista más, que se utilizará para generar el dataframe que se devolverá como resultado.

Field name	Field_type	Subfields	Subfields_type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Sleep_summary	List	start	Timestamp
		end	Timestamp
		tst	Number

Figura 5.12: Estructura final de los datos del sueño tras aplicar la función

string_to_timestamp

- Parámetros
  - date_string (string): cadena que representa una fecha en formato YYYY-MM-DD
- Return
  - date_timestamp (timestamp): valor numérico que representa una fecha

Esta función transforma una fecha en un timestamp con la librería datetime.

5.3.4. steps_structuration_functions.py

En este archivo se encuentran las funciones encargadas de seleccionar y reestructurar los datos de los pasos de los usuarios.

structure_users_steps_data

- Parámetros
  - users_steps_data_list (lista): lista con documentos de la colección Pasos (Figura 5.5) de Firestore.

- Return
  - users_steps_dataframe (dataframe): dataframe con la información necesaria de los pasos de cada usuario

La función reestructura los datos de pasos. Para ello, por cada documento recibido:

Se obtiene la matriz que resume los pasos de cada intervalo de 30 minutos del documento y se almacena su contenido en una lista.

Una vez se tiene toda la información necesaria (incluyendo los valores identificadores del documento como user, uid, service y date_time), agrupa esta en una lista que será la nueva estructura de los datos del usuario (Figura 5.13).

De la misma manera, todas las listas generadas a partir de los documentos de los usuarios se agrupan en otra lista más, que se utilizará para generar el dataframe que se devolverá como resultado.

Field name	Field_type	Subfields	Subfields_type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Steps	List	0	Number
		...	
		47	
Total_steps	Number		

Figura 5.13: Estructura final de los datos de los pasos tras aplicar la función

string_to_timestamp

- Parámetros
  - date_string (string): cadena que representa una fecha en formato YYYY-MM-DD
- Return
  - date_timestamp (timestamp): valor numérico que representa una fecha

Esta función transforma una fecha en un timestamp con la librería datetime.

5.4. Imputación de los datos - data imputation

Las funciones pertenecientes a esta fase (Figura 5.16) son las encargadas de completar los posibles "missings" de las mediciones con datos de otros campos o con información previa que se tenga.

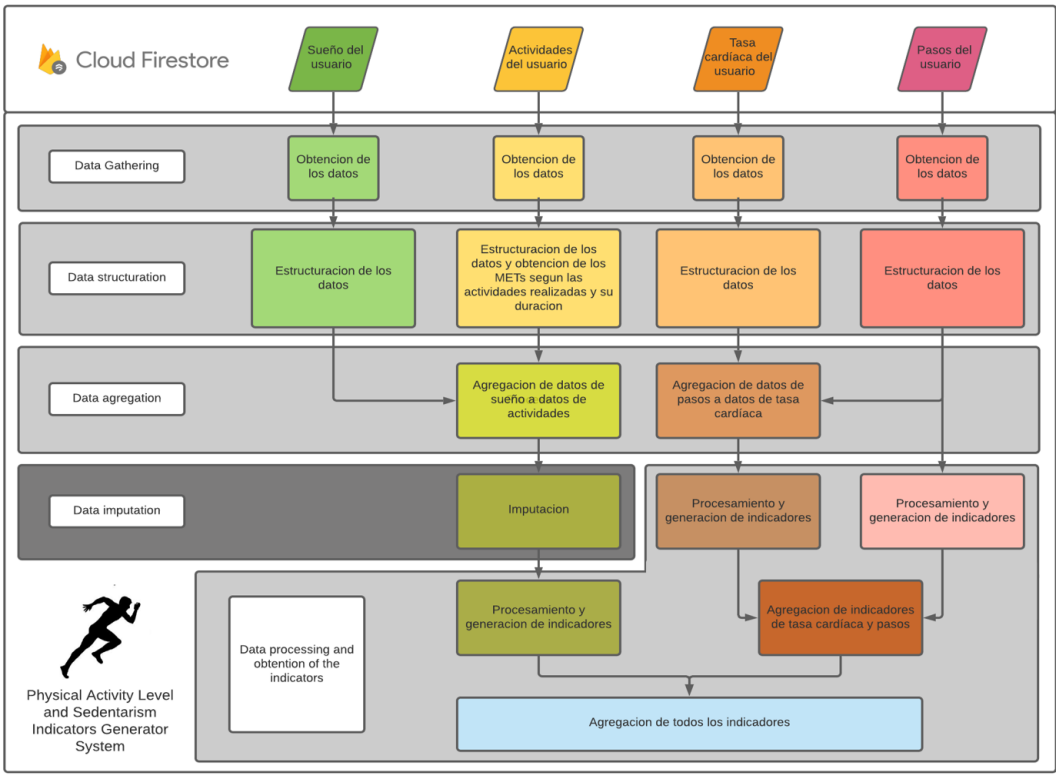

Figura 5.14: Diagrama que representa la estructura del sistema: fase de imputación

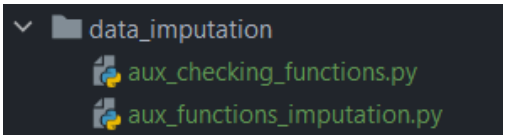

Figura 5.15: Archivos con las funciones de imputación

5.4.1. auxiliary_imputation_functions.py

Los "wearablesregistran la información de las actividades, pero suelen tener más dificultades determinando que actividad realiza el usuario cuando no se mueve. En ocasiones, esto hace que se registren los periodos de "actividad sedentaria por sueño" como periodos de "actividad sedentaria por defecto en vigilia" (despierto), lo cual tiene un valor de METs distinto. Además, algunos dispositivos ni siguiera indican las actividades sedentarias, por lo que es interesante completar esta información con el sueño todo lo que se pueda.

En este archivo se encuentran las cuatro funciones auxiliares que permiten imputar datos de actividad a partir del sueño.

add_sleep_seconds

- Parámetros
  - current_30m_slot_total_SA_dur (number): valor numérico que representa el tiem-

po total empleado en actividades sedentarias en un intervalo de 30 minutos.

- current_30m_slot_total_dur (number): valor numérico que representa el tiempo total empleado en todas las actividades realizadas en ese intervalo de 30 minutos.
- current_30m_slot_MET_value (number): valor numérico que representa cuantos METs se han consumido en ese intervalo de 30 minutos.
- remaining_sleep_seconds(number): valor numérico que representa los segundos restantes de sueño en el instante inicial de ese intervalo de 30 minutos.

- Return

  - current_30m_slot_total_SA_dur (number): valor numérico que representa el tiempo total empleado en actividades sedentarias en un intervalo de 30 minutos.
  - current_30m_slot_MET_value (number): valor numérico que representa cuantos METs se han consumido en ese intervalo de 30 minutos.
  - remaining_sleep_seconds (number): valor numérico que representa los segundos restantes de sueño en el instante final de ese intervalo de 30 minutos.

La función calcula el número de segundos que se ha pasado realizando actividades sedentarias en el intervalo y cuantos segundos sin mediciones de actividad se han registrado en este. Tras ello, mientras queden segundos de sueño en remaining_sleep_seconds los utilizará para: i) sustituir los segundos de actividad sedentaria que se pueda y ii) en caso de que queden, completar los segundos sin mediciones del intervalo con segundos de sueño. Una vez hecho esto, devolverá el gasto del intervalo en METs actualizado, el tiempo que se ha pasado realizando actividades sedentarias actualizado, y el número de segundos de sueño restantes.

imputate_sleep_on_activity

- Parámetros

  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad y sueño del día de un usuario.

- Return

  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad y sueño del día de un usuario.

La función calcula los segundos de sueño del usuario en el día y el intervalo de 30 minutos en el que el usuario comenzó a dormir. Tras ello, mientras queden segundos de sueño disponibles, llamará a add_sleep_seconds para completar los datos de actividad con los de sueño. Una vez utilizados todos los segundos de sueño, actualizará los datos de actividad del usuario con los nuevos valores.

imputation_still_on_missing

- Parámetros
  - current_30m_slot_total_dur (number):valor numérico que representa el tiempo total empleado en todas las actividades realizadas en ese intervalo de 30 minutos.
  - current_30m_slot_MET_value (number): valor numérico que representa cuantos METs se han consumido en ese intervalo de 30 minutos.
- Return
  - current_30m_slot_total_value (number): valor numérico que representa cuantos METs se han consumido en ese intervalo de 30 minutos.
  - slot_missing_seconds (number): valor numérico que indica cuantos segundos se han imputado en la función.

La función calcula cuantos segundos sin mediciones de actividad se han registrado en el intervalo actual y añade esa misma cantidad de segundos de actividad sedentaria al intervalo. Tras ello, devuelve el nuevo valor de METs del intervalo recalculado y el numero de segundos de actividad sedentaria añadidos a este.

impute_missing_on_activity

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad y sueño del día de un usuario.
- Return
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad y sueño del día de un usuario.

Por cada intervalo de 30 minutos del día, la función llama a imputation_still_on_missing para completar la información que sea posible. Tras ello, actualiza los datos de actividad del usuario con los nuevos valores obtenidos.

5.5. Procesamiento de los datos - data processing

Las funciones pertenecientes a esta fase (Figura 5.16) son las encargadas de procesar los datos actuales y generar los indicadores.

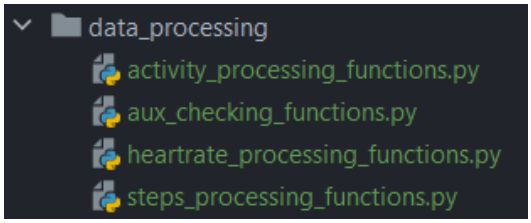

Figura 5.16: Archivos con las funciones de procesamiento

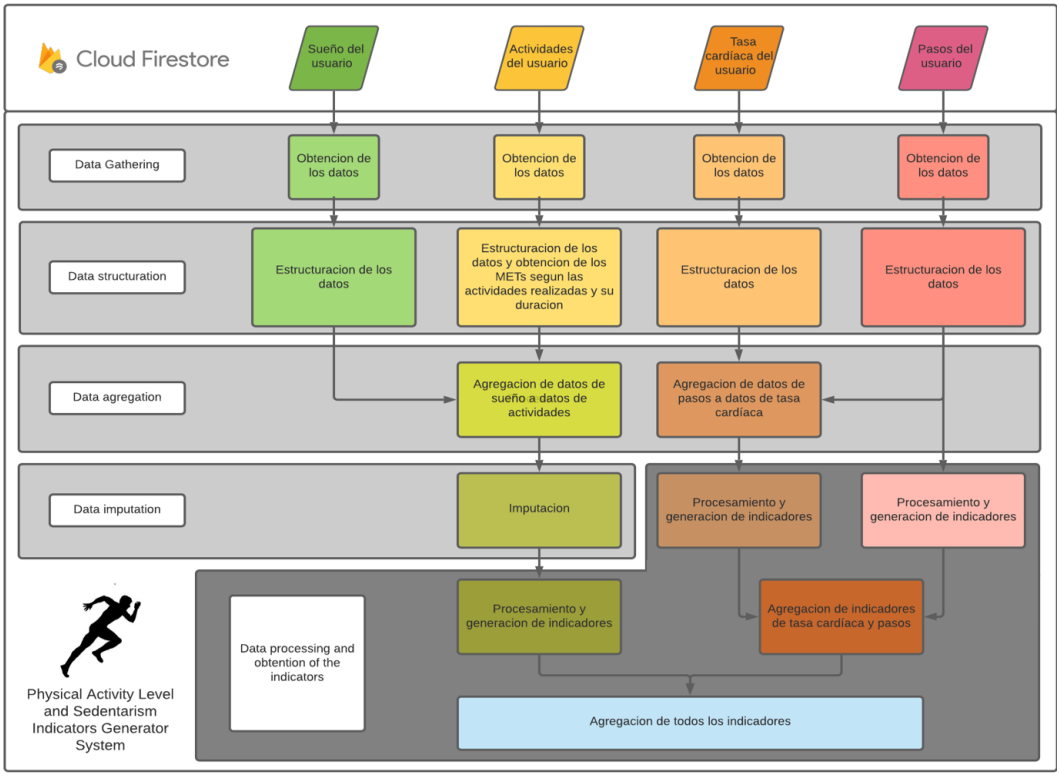

Figura 5.17: Diagrama que representa la estructura del sistema: fase de procesamiento

5.5.1. activity_processing_functions.py

En este archivo se encuentran las funciones encargadas de procesar los datos de actividad y generar los indicadores de este campo.

meets_OMS_daily_min_recommendation_METs

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad del día de un usuario.
- Return
  - (boolean)

La función devuelve si el usuario ha gastado el mínimo de METs, establecido por la OMS, realizando actividades moderadas o intensas ese día.

meets_OMS_daily_min_recommendation_dur

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad del día de un usuario.
- Return
  - (boolean)

La función devuelve si el usuario ha gastado el mínimo de METs, establecido por la OMS, realizando actividades moderadas o intensas ese día.

get_METs_daily_ratio

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad del día de un usuario.
- Return
  - (number)

La función devuelve el nivel de actividad física del usuario. La fórmula utilizada se explica en la base teórica.

get_PAL_daily_level

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad del día de un usuario.
- Return
  - (string)

La función devuelve el modo de vida correspondiente al nivel de actividad física calculado previamente. La tabla de correspondencias se encuentra en la base teórica.

get_sedentarism_daily_ratio

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad del día de un usuario.
- Return
  - (number)

La función devuelve el grado de sedentarismo del usuario. Para ello, calcula los segundos que pasa el usuario en actividades sedentarias y no sedentarias, resta al tiempo de actividad sedentaria el tiempo de sueño (si no se tiene información de sueño del usuario se restan 8h en segundos), y se calcula el ratio de sedentarismo con la formula explicada en la base teórica.

is_sedentary

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de actividad del día de un usuario.
- Return
  - (boolean)

La función devuelve si el usuario se considera sedentario de acuerdo a lo establecido en la base teórica.

5.5.2. heartrate_processing_functions.py

En este archivo se encuentran las funciones encargadas de procesar los datos de tasa cardíaca y generar los indicadores de este campo.

calculate_average_hr_slots

- Parámetros
  - heart_rate_measures_list (lista): lista con las mediciones de tasa cardíaca del usuario y sus respectivas horas de toma.
  - slot_index_list (lista): lista con los indices de los intervalos de 30 minutos del día a utilizar.
- Return

  - hr_baseline_list (lista): lista con las tasas cardíacas medias de los intervalos indicados.

La función obtiene la hora de inicio y fin de los intervalos indicados en la lista y calcula la media de todas las mediciones de tasa cardíaca que se hayan realizado en cada uno. Tras ello, añade estos valores a hr_baseline_list y la devuelve como resultado.

calculate_total_average_hr

- Parámetros
  - average_hr_slots_list (lista): lista con tasas cardíacas medias de intervalos de 30 minutos del día.
- Return
  - (number)

La función devuelve la media de los valores recibidos.

get_bmr_with_aux_steps

- Parámetros
  - heart_rate_measures_list (lista): lista con las mediciones de tasa cardíaca de un usuario y sus respectivas horas de toma.
  - steps_measures_list (lista): lista con 48 "slots" que almacena cuantos pasos ha dado un usuario en cada intervalo de 30 minutos del día.
- Return
  - (number)

La función obtiene el índice de aquellos intervalos en los que el usuario haya dado más de 0 y menos de 210 pasos para asegurar que está en estado de reposo. Una vez hecho esto, llama a la función calculate_average_hr_slots para obtener la tasa cardíaca media del usuario en estos intervalos y pasa estos valores a calculate_total_average_hr para calcular la tasa cardíaca media en reposo (BMR), la cual devuelve como .

get_bmr_from_sources

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la tasa cardíaca del día de un usuario.
- Return

  - (number)

La función llama a get_bmr_with_aux_steps, pasando como parámetro la lista con las mediciones de tasa cardíaca del usuario y la matriz con los pasos que ha dado el usuario en cada intervalo de 30 minutos del día, y devuelve el resultado de la llamada.

get_slots_average_hr_from_sources

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la tasa cardíaca del día de un usuario.
- Return
  - (lista)

La función llama a calculate_average_hr_slots para obtener la tasa cardíaca media de cada intervalo de 30 minutos del día, y devuelve el resultado de la llamada.

approximate_METs_with_hr_data

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la tasa cardíaca del día de un usuario.
- Return
  - total_30m_METs_slots (lista): lista con 48 "slots" que alamacena los METs consumidos, calculados a partir de la tasa cardíaca. La formula utilizada se encuentra en la base teórica.

La función calcula los METs estimados de cada intervalo de 30 minutos del día con la tasa cardíaca media correspondiente a cada uno y el BMR del día. Una vez se han calculado los METs de todos los intervalos, se devuelve una lista con estos como resultado.

approximate_total_METs

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la tasa cardíaca del día de un usuario.
- Return
  - total_mets (number): valor numérico que representa el total de METs consumidos en el día.

La función calcula los METs totales gastados en el día a partir de los calculados con la tasa cardíaca.

get_METs_daily_ratio

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la tasa cardíaca del día de un usuario.
- Return
  - (number)

La función devuelve el nivel de actividad física del usuario. La fórmula utilizada se explica en la base teórica.

get_PAL_daily_level

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la tasa cardíaca del día de un usuario.
- Return
  - (number)

La función devuelve el modo de vida correspondiente al nivel de actividad física calculado previamente. La tabla de correspondencias se encuentra en la base teórica.

5.5.3. steps_processing_functions.py

En este archivo se encuentran las funciones encargadas de procesar los datos de tasa cardíaca y generar los indicadores de este campo.

get_PAL_daily_level

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de los pasos del día de un usuario.
- Return
  - (string)

La función devuelve el modo de vida correspondiente al numero de pasos dados en el día. La tabla de correspondencias se encuentra en la base teórica.

5.5.4. aux_checking_functions.py

En ocasiones puede haber usuarios sin datos de algún tipo o con mediciones erróneas que puedan generar excepciones. Para evitarlas, las funciones de procesamiento llaman a otras auxiliares que realizan comprobaciones sobre los datos e indican si se puede realizar el procesamiento. Estas funciones auxiliares se encuentran en este archivo.

available_activity_data

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la actividad del día de un usuario.
- Return
  - (boolean)

La función devuelve si el usuario tiene datos de actividad.

available_heartrate_data

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la tasa cardíaca del día de un usuario.
- Return
  - (boolean)

La función devuelve si el usuario tiene datos de tasa cardíaca.

available_steps_data

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de los pasos del día de un usuario.
- Return
  - (boolean)

La función devuelve si el usuario tiene datos de pasos.

available_sleep_data

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información del sueño del día de un usuario.
- Return
  - (boolean)

La función devuelve si el usuario tiene datos de sueño.

valid_activity_MET_meassurements

- Parámetros
  - user_day (pandas.series): fila de un dataframe que contiene, al menos, la información de la actividad del día de un usuario.
- Return
  - (boolean)

La función devuelve si los METs registrados del usuario están dentro de los límites establecidos.

5.6. Funciones generadoras

Las funciones pertenecientes a esta categoría no forman parte de las cinco etapas de gestión de datos, sino que son las encargadas de llamar a todas las funciones que ya se han explicado para generar los indicadores de actividad y sedentarismo.

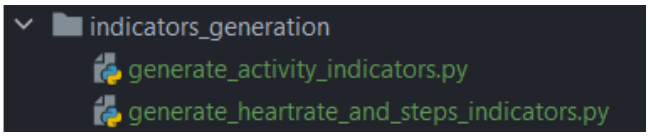

Figura 5.18: Archivos con las funciones generadores

5.6.1. generate_activty_indicators.py

En este archivo se encuentra la función encargada de llamar a las funciones necesarias para generar los indicadores a partir de los datos de actividad.

generate_indicators_from_activity

- Parámetros
  - date_time (string): cadena que representa la fecha de cuyo día queremos obtener los datos.
- Return
  - users_steps_dataframe (dataframe): dataframe con los indicadores de actividad de cada usuario

La función recoge los datos de actividad y sueño del usuario y los reestructura para adaptarlos al formato adecuado. Realiza un LEFT JOIN de los datos de actividad con los de sueño, obteniendo así todos los usuarios con datos de actividad y añadiendo a estos los datos de sueño que tengan. Utilizando este nuevo campo de datos conjunto, realiza las imputaciones que sean posibles y, con la información que posea, genera los indicadores de cada usuario. Una vez generados, los devuelve en forma de dataframe.

5.6.2. generate_heartrate_and_steps_indicators.py

En este archivo se encuentra la función encargada de llamar a las funciones necesarias para generar los indicadores a partir de los datos de pasos y tasa cardíaca.

generate_indicators_from_heartrate_and_steps

- Parámetros
  - date_time (string): cadena que representa la fecha de cuyo día queremos obtener los datos.
- Return
  - users_steps_dataframe (dataframe): dataframe con los indicadores de pasos y tasa cardíaca de cada usuario

La función recoge los datos de tasa cardíaca y pasos del usuario y los reestructura para adaptarlos al formato adecuado. Realiza un INNER JOIN de los datos de tasa cardíaca con los de pasos, obteniendo así sólo los usuarios con datos de actividad y datos de pasos. Tras ello, con la información recogida de los pasos, genera los indicadores de cada usuario relacionados con los pasos y, con la información resultante del JOIN, los indicadores relacionados con la tasa cardíaca. Una vez generados, los devuelve en forma de dataframe.

5.7. Función ejecutable (main)

Finalmente, queda la función principal del sistema, que se ejecuta para lanzar el sistema y generar los indicadores.

5.7.1. pal_and_sl_indicators_generator.py

Al ser simplemente la encargada de recopilar los indicadores, su labor es bastante simple. En primer lugar, establece las credenciales necesarias para establecer la conexión con la base de datos. Tras ello, determina la fecha del día cuyos datos queremos obtener, llama a las dos funciones generadoras para obtener los indicadores y une estos mediante un OUTTER JOIN. Una vez tiene los indicadores de cada usuario agrupados, por cada usuario genera un documento con sus indicadores y lo añade colección de los indicadores del nivel de actividad y sedentarismo de la base de datos de Firestore.

Field name	Field_type	Subfields	Subfields_type
User	String		
User_uid	String		
Date_time	String		
Service	String		
Creation_datetime	Timestamp		
Last_mod_datetime	Timestamp		
Steps	Map / Dictionary	PAL	String
		N_steps	Number
Heart_rate	Map / Dictionary	METs_ratio	Number
		PAL	String
Activty	Map / Dictionary	OMS_METs	Boolean
		OMS_duration	Boolean
		METs_ratio	Number
		PAL	String
		sedentary_ratio	Number
		Is_sedentary	Boolean

Figura 5.19: Estructura de los documentos de la colección de indicadores del nivel de actividad y sedentarismo

- User. Nombre de usuario (string)
- User_uid. Id de usuario (string)
- Date_time. Fecha de la recogida de datos (string)
- Service. Servicio del usuario (string)
- Creation_datetime. Fecha de creación de la colección de datos (timestamp)
- Last_mod_datetime. Fecha de modificación de la colección de datos (timestamp)
- Steps. Diccionario con los indicadores del nivel de actividad y sedentarismo a partir de los datos de los pasos:
  - PAL. Modo de vida del usuario según sus pasos del día (string)
  - N_steps. Pasos totales del día (number)
- Heartrate. Diccionario con los indicadores del nivel de actividad y sedentarismo a partir de los datos de la tasa cardíaca:

  - PAL. Modo de vida del usuario según su nivel de actividad (string)
  - METs_ratio. Nivel de actividad del usuario en METs (number)

- Activity. Diccionario con los indicadores del nivel de actividad y sedentarismo a partir de los datos de actividad:

  - OMS_METs. Valor que indica si el usuario cumple los niveles mínimos establecidos por la OMS en cuanto al gasto de METs en actividades moderadas e intensas (boolean)
  - OMS_duration. Valor que indica si el usuario cumple los niveles mínimos establecidos por la OMS en cuanto al tiempo empleado en actividades moderadas e intensas (boolean)
  - METs_ratio. Nivel de actividad del usuario en METs (number)
  - PAL. Modo de vida del usuario según su nivel de actividad (string)
  - sedentary_ratio. Ratio del tiempo que el usuario emplea en actividades sedentarias sobre no sedentarias mientras está despierto (number)
  - is_sedentary. Valor que indica si el usuario pasa más de 3/4 del tiempo de vigilia de un día en actividades sedentarias (boolean)

Capítulo 6

Optimización del sistema

El sistema desarrollado no sigue la estructura establecida, ni cuenta con las funciones que cuenta por casualidad. Al buscar ser alojado en cloud y estar continuamente operando con nuevos datos, ha sido necesario un proceso de optimización para disminuir tanto el coste temporal, como el coste de recursos, los cuales derivan directamente en un coste económico.

6.1. Obtención de los datos

La facturación de Firestore [30] puede ser muy elevada cuando se manejan un gran numero de colecciones, documentos y por ende, datos. En concreto, hay una parte de la facturación que se centra únicamente en el acceso a los datos. Es decir, las operaciones de lectura.

Previamente se han explicado ya las funciones encargadas de recopilar los datos de las colecciones de Firestore. En este caso, nos interesa concretamente recursive_firestore_query.

En un inicio, en vez de utilizar esta función, se realizaba una única lectura desde get_data_from_collection que obtenía el total de los documentos resultantes de ejecutar una query sobre la base de datos. Esto puede parecer lógico ya que, realmente, la función recursiva termina leyendo todos los documentos de igual manera. Sin embargo, surgía un problema si, mientras se estaban leyendo todos los documentos, cambiaba el contenido de alguno. Firestore, para evitar problemas de integridad, detecta si algún documento que ya se haya leído se modifica mientras la lectura no haya terminado, y si es así, hace que vuelva a leerse antes de finalizar el proceso. En una base de datos relativamente pequeña o estática esto no es un problema, pero si estamos hablando de una base de datos de grandes dimensiones, con actualizaciones de datos de los usuarios en tiempo real, las lecturas extra pueden incrementar el coste enormemente.

Fue por ello que se decidió desarrollar la función recursiva actual, que realiza una lectura de N documentos de entre todos los devueltos por la consulta, almacena la información, y realiza una lectura distinta a partir del punto en el que haya finalizado la anterior. Realizar varias consultas no presenta un problema, ya que Firestore cobra únicamente por la lectura de cada documento de manera individual, por lo que costaría lo mismo una lectura de diez documentos una vez, que diez lecturas de un único documento. Sin embargo, si que causa que la lectura total sea más lenta, lo cual tampoco es deseable.

Que recursive_firestore_query permitiese la lectura de un numero arbitrario de documentos, permitió valorar qué volumen de documentos minimizaba el número de lecturas repetidas, sin derivar en un aumento demasiado elevado del tiempo que tomaba obtener todos los documentos.

Para ello, se realizó una prueba en la que se le asignaron tres posibles valores al numero máximo de documentos leídos por iteración recursiva de la función: N = 500, N = 1000 y N = 1500. Para cada valor se leían cien veces, de manera repetida, los documentos del día (de las colecciones utilizada por el sistema) de todos los usuarios de test de EB2. De esta forma, se podía obtener, para cada N: i) el tiempo medio (de 100 ejecuciones) de lectura de todos los documentos necesarios para la ejecución del sistema y ii) el número de documentos leídos totales tras las 100 ejecuciones.

Tiempo medio de lectura de todos los datos requeridos por el sistema	Numero máximo de documentos leídos por iteración recursiva (N)	Documentos totales leídos
3,33782839775085	500	3219
2,87932397842407	1000	3354
2,87599468231201	1500	3327

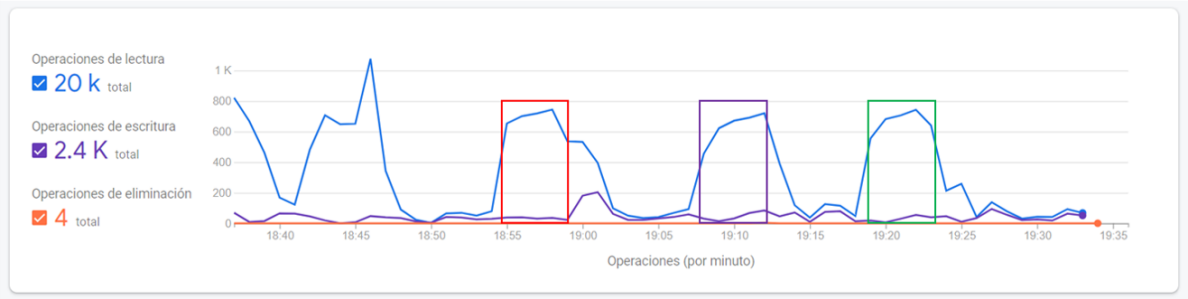

Figura 6.1: Resultados de las pruebas de rendimiento de obtención de datos

En los resultados (Figura 6.1) puede verse como hay un incremento en las lecturas totales de N = 1000 y N = 1500 en comparación con las de N = 500, derivadas de las lecturas repetidas de documentos. Al estar usando menos usuarios (únicamente los de test) la diferencia de lecturas es leve pero, si se somete al sistema a un mayor numero de usuarios, esta aumentará. Valorando estos datos con EB2, ya que se estaban utilizando sus recursos, se llegó a la conclusión de que la diferencia, en el tiempo de lectura media de todos los datos, entre N = 500 y las otras dos asignaciones, era lo suficientemente baja como para priorizar una lectura de documentos totales más baja antes que el tiempo de lectura total. Es por esto que se utiliza finalmente el valor 500 en el parámetro step de recursive_firestore_query.

6.2. Gestión de los datos

Al igual que en el caso previo, la estructura de los datos y del sistema ha ido variando a lo largo del proceso, buscando optimizar el tiempo y los recursos necesarios para generar los indicadores.

En un inicio, buscando únicamente el correcto funcionamiento del sistema y la generación de indicadores de manera adecuada, la estructura del sistema giraba alrededor de

la creación de un macroconjunto de datos de los usuarios sobre el que aplicar todas las funciones de imputación y procesamiento.

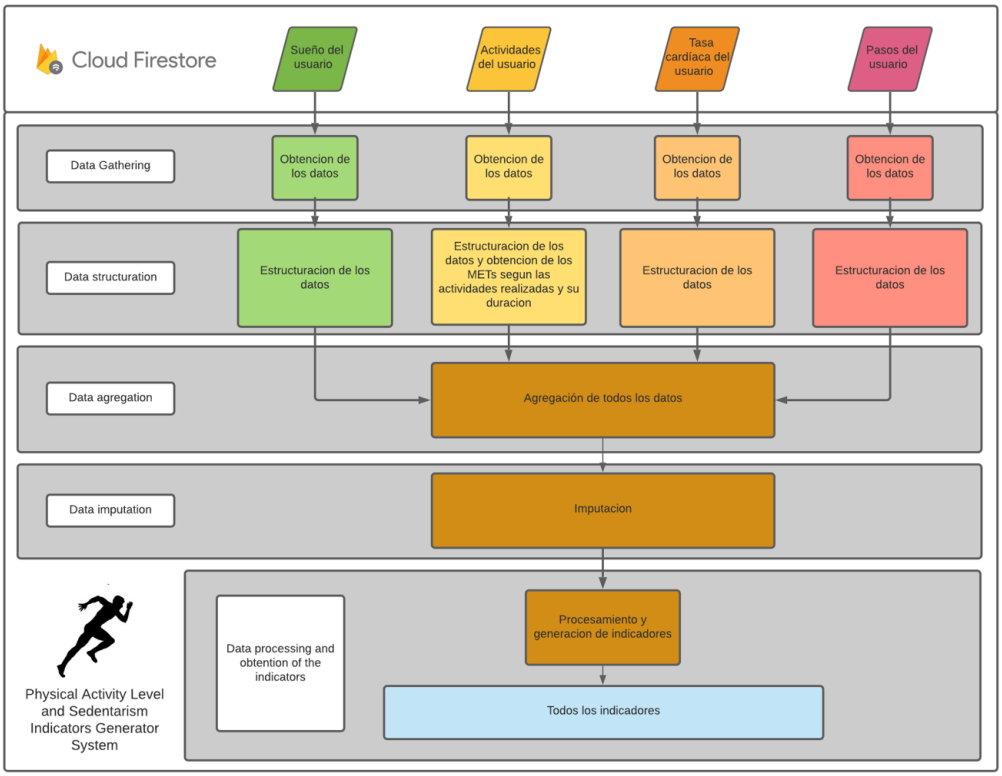

Figura 6.2: Primera estructura del sistema, previa a la optimización

Sin embargo, esta estrategia presentaba varios problemas que se fueron observando a medida que se buscaba optimizar el proceso:

a) Gran coste derivado de realizar un join de todos los datos de todas las colecciones: la operación JOIN, realizada en el sistema con la función merge, es una herramienta costosa tanto a nivel temporal, como a nivel de recursos, a la hora de agupar grandes conjuntos de datos.

b) Aplicación de procesos innecesarios sobre el total de los usuarios: como es natural, las fuentes de información de los usuarios no tienen porqué ser las mismas, lo que deriva en que haya usuarios con datos en algunas colecciones de las utilizadas y otros usuarios con datos en otras distintas. Al agrupar todos los datos después de la estructuración, se obligaba al sistema a intentar aplicar funciones de imputación en usuarios sin datos de actividad, funciones de procesamiento de pasos a usuarios sin datos de pasos, etc.

c) Aplicación de todos los procesos sobre el total de los datos: al trabajar con un único macroconjunto de datos, se aplicaban todas las funciones a todos los usuarios y se forzaba a estas a buscar los datos que necesitaban de entre todos los que contenía cada usuario.

d) Uso de estructuras o funciones no óptimas a la hora de realizar cálculos sobre matrices o conjuntos de datos: principalmente el uso de bucles, para realizar acciones

repetidas u operaciones aritméticas sobre grandes conjuntos de datos, y las referencias innecesarias a datos de los usuarios en vez de almacenar estos en variables locales.

Para mejorar el rendimiento se llevó a cabo un proceso de optimización que se organizó de la siguiente manera: i) optimización del código - buscando resolver el apartado "d" -, ii) aplicación de las funciones únicamente sobre los datos de los usuarios que estas necesitan - para solventar el apartado "c" - y iii) cambio de la estructura del sistema para buscar una alternativa al uso del macroconjunto de datos - para solucionar los puntos "a" y "b" -.

Para dar una idea de la mejoría del sistema a medida que se fueron aplicando las distintas optimizaciones, se incluirán dos mediciones del tiempo de ejecución realizadas en cada una de estas etapas. La primera, analiza 1000 ejecuciones del sistema con una cantidad limitada de datos reales y calcula el tiempo medio de estas. Por otro lado, la segunda busca comprobar el rendimiento del sistema con una sobrecarga de datos (250000 usuarios), muy superior a la carga a la estaría expuesto en un principio. A continuación se muestran ambas estadísticas para el estado inicial del sistema:

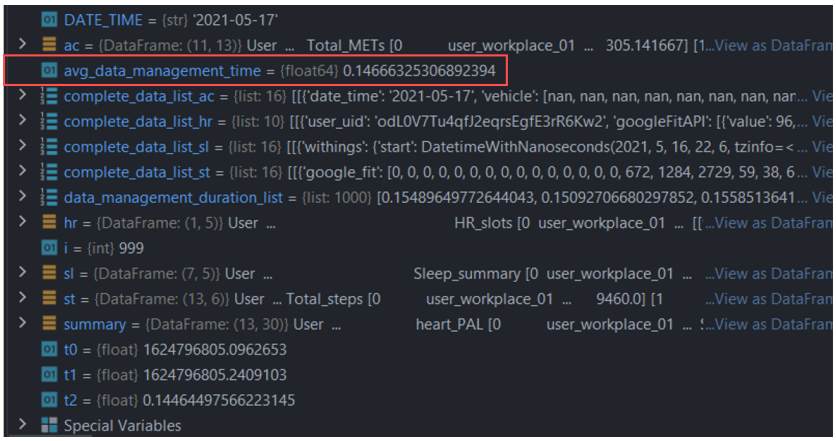

Figura 6.3: Tiempo medio en obtener los indicadores en 1000 ejecuciones del sistema con un volumen de datos habitual: 0.14666 segundos

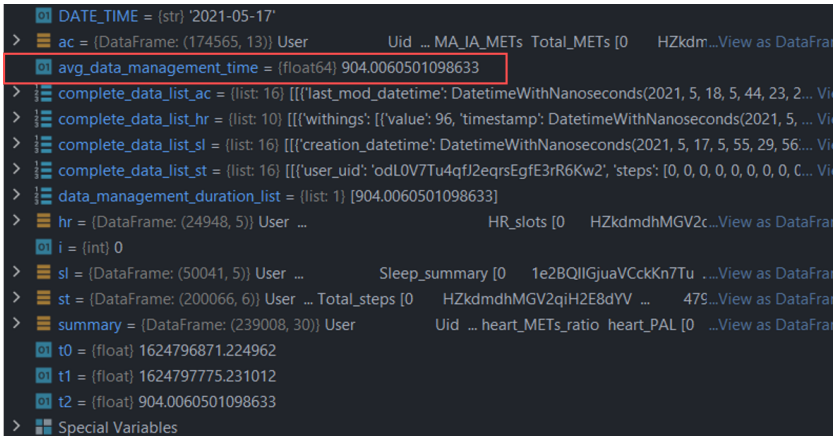

Figura 6.4: Tiempo medio en obtener los indicadores en una ejecución del sistema con un datos de 250000 usuarios: 904.00605 segundos

6.2.1. Optimización del código

Como se ha mencionado en la lista anterior, en un inicio se utilizaban bucles para realizar operaciones aritméticas sobre los distintos elementos de un conjunto de datos. Para mejorar este proceso, los "for" que se utilizaban para operaciones aritméticas fueron sustituidos por funciones que implementa la librería numpy (nanmean, nansum, etc.) destinadas para esta labor concreta [31].

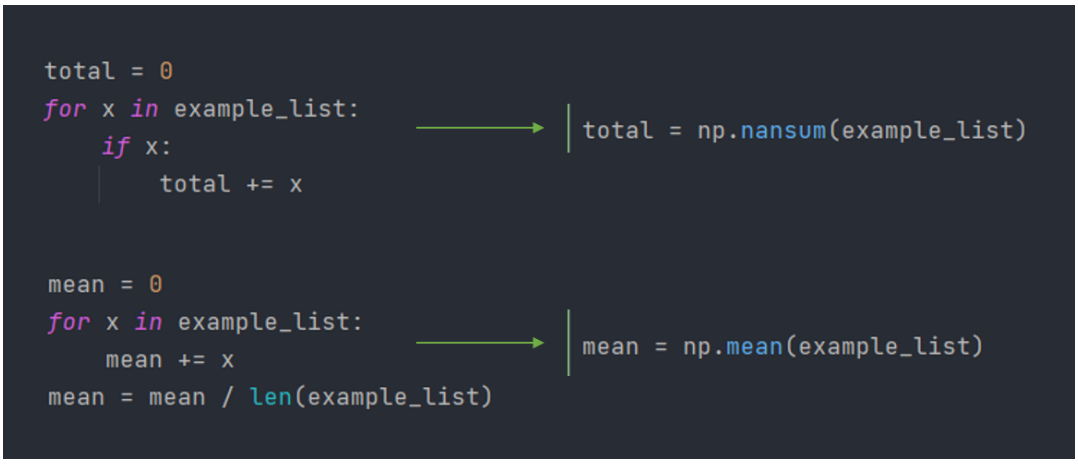

Figura 6.5: Ejemplo básico del cambio de un bucle a operación de orden superior de Numpy

Además, se procuró sustituir todos los bucles encargados de crear nuevas listas por las estructuras comúnmente denominadas "list comprehension" de python [32], que permiten acelerar estos procesos y obtener un mejor rendimiento.

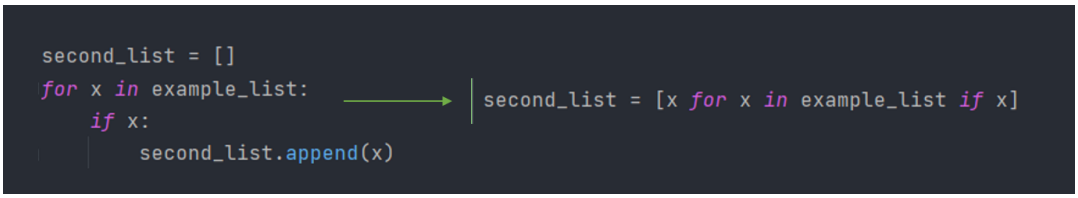

Figura 6.6: Ejemplo básico del cambio de un bucle a estructura "list comprehension" de Python

Una vez hecho esto, se eliminó el uso de referencias múltiples a información que hubiese que buscar en conjuntos de datos, para eliminar los costes de esta búsqueda repetida. En su lugar, se almacenó la información necesitada en variables locales para dar lugar a un coste menor.

Figura 6.7: Ejemplo de cambio de referencias múltiples a uso de variables locales

El rendimiento resultante del sistema, para ambas estadísticas, fue el siguiente:

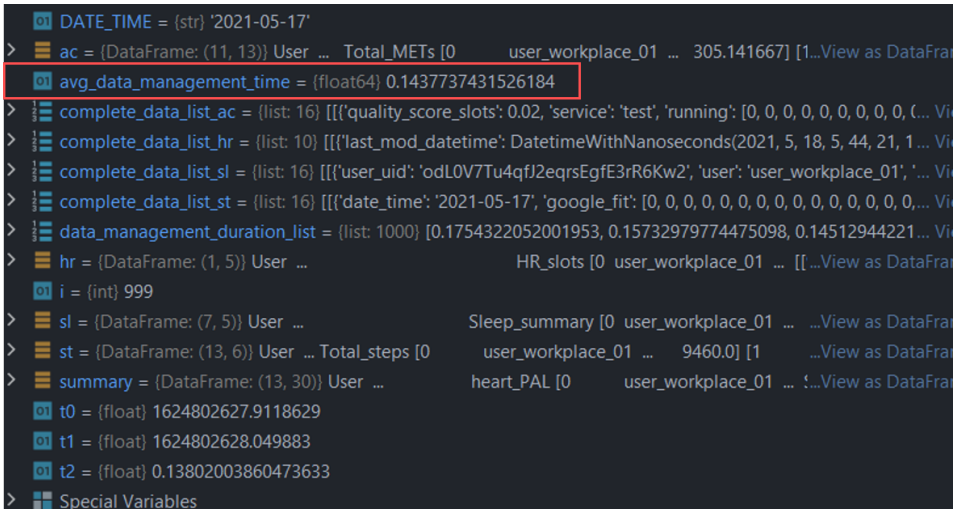

Figura 6.8: Tiempo medio en obtener los indicadores en 1000 ejecuciones del sistema con un volumen de datos habitual: 0.14377 segundos

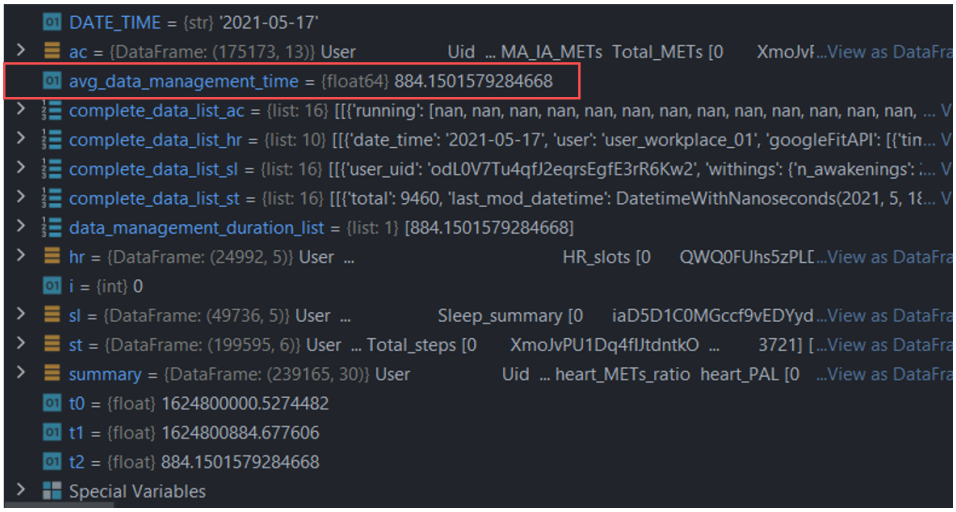

Figura 6.9: Tiempo medio en obtener los indicadores en una ejecución del sistema con un datos de 250000 usuarios: 884.15015 segundos

6.2.2. Aplicación de las funciones únicamente sobre los datos de los usuarios que estas necesitan

Al contrario que para los cambios previos, en este caso no fue necesaria una importante investigación sobre buenas prácticas de gestión de grandes volúmenes de datos, sino que fue suficiente con aprovechar una de las utilidades de la librería pandas.

Teniendo la información ya estructurada en dataframes, pandas incorpora una sencilla manera para obtener un sub-dataframe a partir de columnas o filas del original. Esto permitió solucionar el problema simplemente aplicando la función deseada, en vez de sobre el conjunto total, sobre el subconjunto que se eligiese.

```
example['new_field1']           = example.apply(example_function1, axis=1)
example['new_field2']           = example.apply(example_function2, axis=1)

example['new_field1']           = example[['field1', 'field2', 'field3']].apply(example_function1, axis=1)
example['new_field2']           = example[['field1', 'field6', 'field7']].apply(example_function2, axis=1)
```

Figura 6.10: Ejemplo de uso de subconjuntos para aplicar sobre estos las funciones

Sin embargo, para mi sorpresa, el rendimiento tras este cambio empeoró claramente:

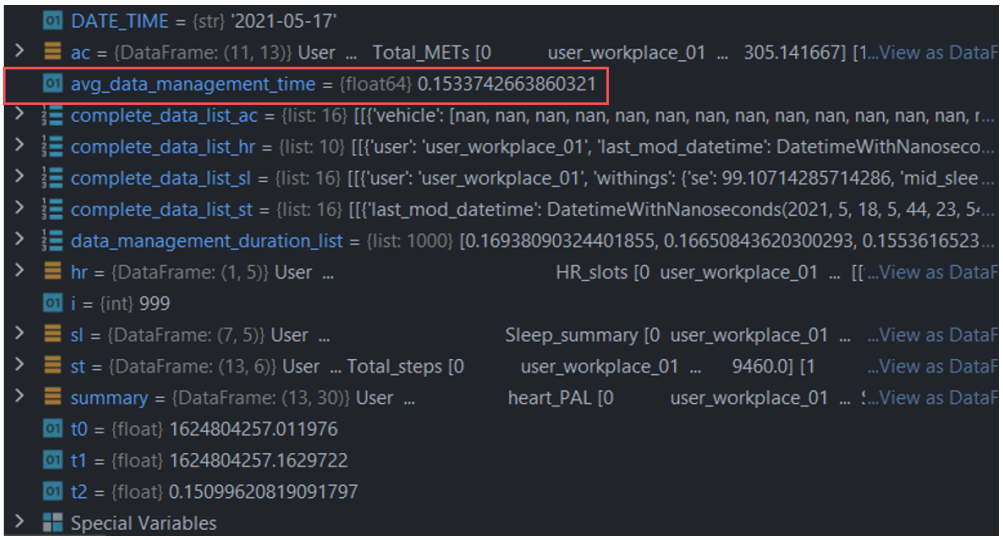

Figura 6.11: Tiempo medio en obtener los indicadores en 1000 ejecuciones del sistema con un volumen de datos habitual: 0.15337 segundos

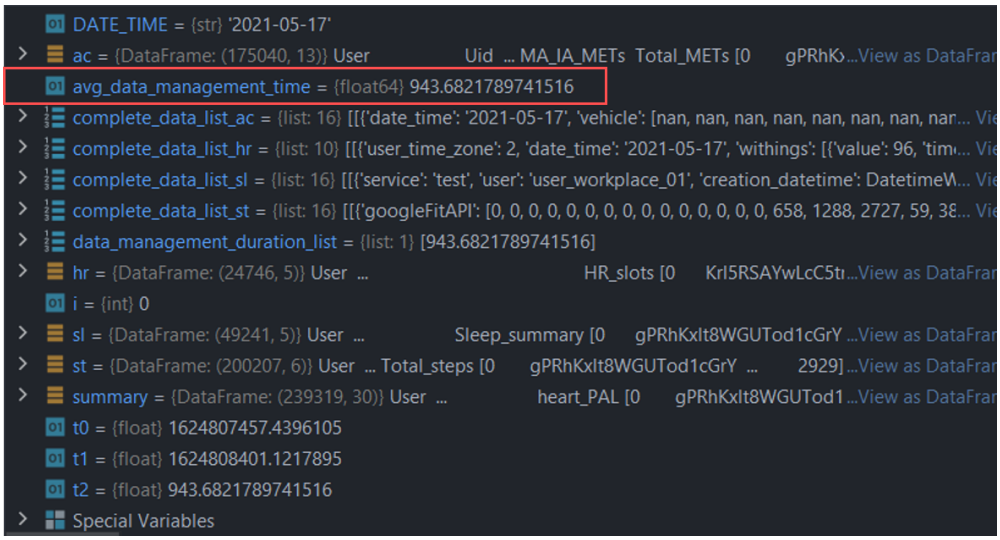

Figura 6.12: Tiempo medio en obtener los indicadores en una ejecución del sistema con un datos de 250000 usuarios: 943.68217 segundos

La razón era obvia y es que, pese a estar reduciendo la carga que soportaban las funciones, el sistema tenia que generar tantos subconjuntos de datos como funciones

quisiese aplicar, lo cual aumentaba el esfuerzo total. En resumen, estábamos haciendo al inicio de la ejecución un merge, que luego era beneficioso deshacer para mejorar el rendimiento, lo cual es bastante absurdo. Esta conclusión llevó a la última fase de la optimización.

6.2.3. Cambio de la estructura del sistema para buscar una alternativa al uso del macroconjunto de datos

Teniendo en mente los resultados del intento de optimización previo, el coste de realizar un JOIN con un número elevado de datos, y que no todos los usuarios requerían de todas las funciones de procesamiento, se llegó a la conclusión de que era necesaria plantear una nueva estructura.

Durante la carrera, en las asignaturas relacionadas con las bases de datos, siempre se ha hecho gran hincapié en la importancia de minimizar el tamaño de los conjuntos sobre los que se realizaría algún tipo de JOIN. Para ello, en muchas ocasiones se realizaba sobre estos subconjuntos las operaciones pertinentes y se realizaba el JOIN con los elementos resultantes.

Siguiendo esta enseñanza, se estableció una nueva estructura (la actual, mostrada en la Figura 5.1) que separase completamente el procesamiento de cada colección mientras fuese posible, y sólo uniese los campos de datos en caso de que fuese necesario. De esta forma, resolvíamos los puntos "b" y "c" mencionados previamente, ya que sólo se realizarían operaciones relacionadas con la actividad sobre los usuarios con información de este campo, y encima estos no tendrían datos de otras colecciones no relevantes.

Además, al realizar las distintas operaciones, se descartan aquellos datos que no sean necesarios y los usuarios con mediciones erróneas, lo que deriva en subconjuntos de datos más pequeños que al inicio. Esto implicaría un coste menor al realizar el JOIN con los resultados finales de operar con cada colección o grupo de colecciones (solventando así el punto "a").

Los resultados del cambio fueron positivos y pueden verse a continuación:

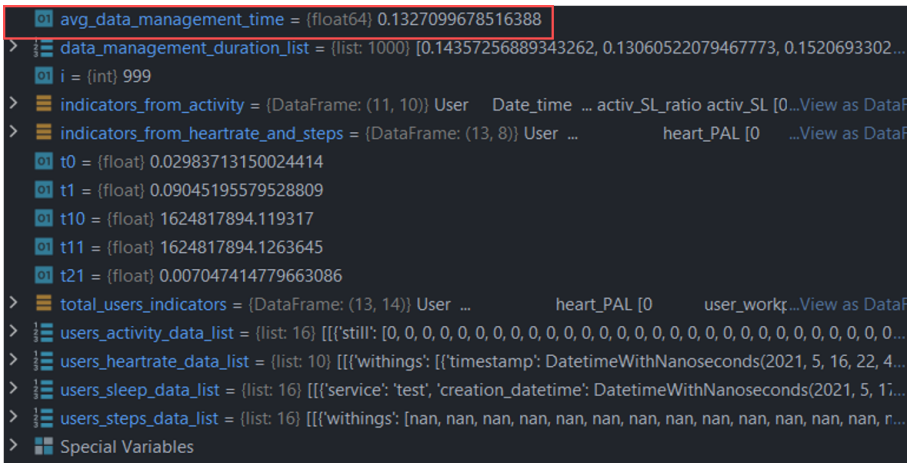

Figura 6.13: Tiempo medio en obtener los indicadores en 1000 ejecuciones del sistema con un volumen de datos habitual: 0.13270 segundos

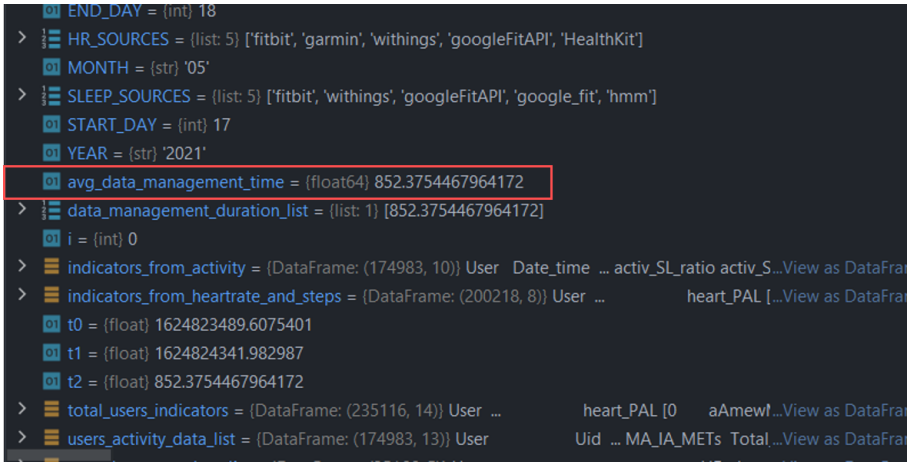

Figura 6.14: Tiempo medio en obtener los indicadores en una ejecución del sistema con un datos de 250000 usuarios: 852.37544 segundos

Capítulo 7

Funcionamiento del sistema

Si bien en la sección anterior se han descrito el sistema y su optimización, es esencial mostrar su funcionamiento a lo largo de las distintas fases. Esto permitirá, tanto demostrar la correcta operatividad del sistema, como entender completamente el flujo de la información y los cambios que sufre esta durante el sistema.

En la demostración del funcionamiento se utilizarán datos reales, del día 17 de mayo de 2021, de los usuarios de test de EB2, por lo que no se mostrarán posibles datos que puedan ser sensibles.

7.1. Generación de indicadores a partir de los datos de actividad

Habiendo especificado la fecha de los datos que queremos obtener, el sistema obtendrá primero los provenientes de las colecciones de actividad y sueño. Como se ha explicado en el sub-apartado de las fuentes de datos, estos se obtendrán como una lista de documentos con formato de diccionarios JSON, por lo que la información a simple vista puede ser algo confusa si no esta organizada.

Datos de la colección de actividad recopilados

[['running': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, 0, 0, 0, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, nan, nan, nan, nan, nan, nan, nan, 0, nan, 0, 0, 0, 0, nan, nan, nan, 0, nan], 'user': 'user_workplace_01', 'user_uid': 'odL0V7Tu4qfJ2eqrsEgfE3rR6Kw2', 'still': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, 0, 0, 0, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, nan, nan, nan, nan, nan, nan, nan, 0, nan, 0, 0, 0, 0, nan, nan, nan, 0, nan], 'date_time': '2021-05-17', 'service': 'test', 'quality_slots': 0.23, 'walking': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 646, 779, 1440, 60, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 250, nan, nan, nan, nan, nan, nan, nan, 372, nan, 240, 660, 480, 180, nan, nan, nan, 124, nan], 'sport': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, 0, 0, 0, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, nan, nan, nan, nan, nan, nan, nan, 0, nan, 0, 0, 0, 0, nan, nan, nan, 0, nan], 'vehicle': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, 0, 0, 0, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, nan, nan, nan, nan, nan, nan, nan, 0, nan, 0, 0, 0, 0, nan, nan, nan, 0, nan], 'last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 18, 5, 44, 23, 249000, tzinfo=¡UTC¿), 'biking': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, 0, 0, 0, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 0, nan, nan, nan, nan, nan, nan, nan, 0, nan, 0, 0, 0, 0, nan, nan, nan, 0, nan], 'quality_score_slots': 0.02, 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 17, 6, 0, 0, 604000, tzinfo=¡UTC¿)], ['user': 'mindcare_test_AAR', 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 16, 22, 8, 50, 719000, tzinfo=¡UTC¿), 'last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 17, 22, 3, 48, 143000, tzinfo=¡UTC¿), 'service': 'test', 'quality_score_slots': 0.02, 'quality_slots': 1, 'walking': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 397, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'sport': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'still': [1800, 1800, 1800, 1800, 1800, 1800, 1800, 1800, 1800, 1800, 1800, 1800, 1800, 1800, 1800, 1780, 1799, 1223, 703, 1800, 1800, 1800, 1800, 1800, 1199, 1800, 1800, 1800, 1069, 893, 1799, 1800, 1799, 1798, 1801, 1800, 1800, 1800, 1797, 1801, 757, 646, 1500, 1782, 1785, 1800, 1800, 1799], 'vehicle': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 540, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 209, 635, 0, 0, 0, 0, 0, 0], 'user_uid': 'EWnmc5uZHTVtC0mRZI8JswIlTlt1', 'biking': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'date_time': '2021-05-17', 'running': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],.......

Datos de la colección de sueño recopilados

[['last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 17, 23, 30, 6, 140227, tzinfo=¡UTC¿), 'withings': 'mid_sleep': DatetimeWith-

Nanoseconds(2021, 5, 17, 1, 50, tzinfo=¡UTC¿), 'end': DatetimeWithNanoseconds(2021, 5, 17, 5, 34, tzinfo=¡UTC¿), 'start': DatetimeWithNanoseconds(2021, 5, 16, 22, 6, tzinfo=¡UTC¿), 'tib': 448.0, 'spt': 448.0, 'tst': 444.0, 'n_awakenings': 2, 'se': 99.10714285714286, 'quality': None, 'waso': 4.0, 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 17, 5, 55, 29, 563474, tzinfo=¡UTC¿), 'service': 'test', 'user_uid': 'odL0V7Tu4qfJ2eqrsEgfE3rR6Kw2', 'date_time': '2021-05-17', 'hmm': 'start': DatetimeWithNanoseconds(2021, 5, 16, 21, 40, 58, 513584, tzinfo=¡UTC¿), 'end': DatetimeWithNanoseconds(2021, 5, 17, 5, 30, 58, 513584, tzinfo=¡UTC¿), 'n_awakenings': 0, 'quality': None, 'se': 100.0, 'mid_sleep': DatetimeWithNanoseconds(2021, 5, 17, 1, 35, 58, 513584, tzinfo=¡UTC¿), 'tst': 470.0, 'waso': 0, 'tib': None, 'spt': 470.0, 'user': 'user_workplace_01', 'user_time_zone': 2], ['user': 'mindcare_test_AAR', 'service': 'test', 'date_time': '2021-05-17', 'HealthKit': 'spt': 390.0, 'n_awake-nings': None, 'tib': None, 'tst': None, 'start': DatetimeWithNanoseconds(2021, 5, 16, 22, 30, tzinfo=¡UTC¿), 'quality': None, 'se': None, 'end': DatetimeWithNanoseconds(2021, 5, 17, 5, 0, tzinfo=¡UTC¿), 'mid_sleep': DatetimeWithNanoseconds(2021, 5, 17, 1, 45, tzinfo=¡UTC¿), 'waso': None, 'user_uid': 'EWnmc5uZHTVtC0mRZI8JswIlTlt1', 'last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 17, 14, 32, 27, 253560, tzinfo=¡UTC¿), 'user_time_zone': 2, 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 17, 7, 4, 38, 130820, tzinfo=¡UTC¿), 'hmm': 'n_awakenings': 1, 'spt': 460.0, 'start': DatetimeWithNanoseconds(2021, 5, 16, 21, 34, 38, 130820, tzinfo=¡UTC¿), 'tib': None, 'quality': None, 'se': 97.82608695652173, 'waso': 10, 'end': DatetimeWithNanoseconds(2021, 5, 17, 5, 14, 38, 130820, tzinfo=¡UTC¿), 'mid_sleep': DatetimeWithNanoseconds(2021, 5, 17, 1, 24, 38, 130820, tzinfo=¡UTC¿), 'tst': 450.0], [], [], ['date_time': '2021-05-17', 'user': 'mindcare_test_4', 'user_uid': 'N5PdkLJZxQZuLbrlQH8QuStsky-r2', 'service': 'test', 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 17, 9, 29, 40, 324487, tzinfo=¡UTC¿), 'hmm': 'spt': 430.0, 'tib': None, 'start': DatetimeWithNanoseconds(2021, 5, 16, 23, 19, 40, 324487, tzinfo=¡UTC¿), 'waso': 50, 'quality': None, 'end': DatetimeWithNanoseconds(2021, 5, 17, 6, 29, 40, 324487, tzinfo=¡UTC¿), 'n_awakenings': 2, 'tst': 380.0, 'se': 88.37209302325581, 'mid_sleep': DatetimeWithNanoseconds(2021, 5, 17, 2, 54, 40, 324487, tzinfo=¡UTC¿), 'user_time_zone': 2, 'last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 17, 9, 29, 40, 324487, tzinfo=¡UTC¿)], [], [], [], [], ['creation_datetime': DatetimeWithNanoseconds(2021, 5, 17, 19, 0, 26, 360464, tzinfo=¡UTC¿), 'hmm': 'se': 95.1219512195122, 'tst': 390.0, 'mid_sleep': DatetimeWithNanoseconds(2021, 5, 14, 3, 45, 26, 360464, tzinfo=¡UTC¿), 'end': DatetimeWithNanoseconds(2021, 5, 14, 7, 10, 26, 360464, tzinfo=¡UTC¿), 'spt': 410.0, 'tib': None, 'start': DatetimeWithNanoseconds(2021, 5, 14, 0, 20, 26, 360464, tzinfo=¡UTC¿), 'waso': 20, 'quality': None, 'n_awakenings': 1, ...

Esta estructura complica mucho tanto el entendimiento de la información, como el trabajar con ella. Además, hay un gran numero de datos que no son necesarios para el sistema o que pueden resumirse. Para solventar este problema, el sistema estructurará los datos, dándoles un nuevo formato y seleccionando únicamente la información necesaria para obtener los futuros indicadores.

Datos de la actividad con la nueva estructura

User	Uid	Serv...	Dat...	SA_duration_slots	LA_duration_slots	MA_duration_slots
user_wor...	odL0V7T...	test	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare...	EWnmc5...	test	16212024...	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare...	EzNZ4ma...	uc3m_wo...	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare...	N5PdkLJ...	test	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare...	XVqLk49...	test	16212024...	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare...	XiS8SFC9...	test	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
test-cond...	zqaAWK...	test	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare...	oZyp7e9...	test	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
user_well...	pSpf0tMe...	wellness_...	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
pruebas7	rGRbkicl...	test	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare...	Ut36kG3...	test	16212024...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

IA_duration_slots	MA_IA_dur	Total_duration	METs_slots	MA_IA_METs	Total_METs
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	5231	5231	[0.0, 0.0, 0.0, 0.0, 0.0,	305.14167	305.14167
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	397	80327	[40.5, 40.5, 40.5, 40..	23.15833	1821.58333
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	1955	1955	[0.0, 0.0, 0.0, 0.0, 0.0,	325.83333	325.83333
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	174	174	[0.0, 0.0, 0.0, 0.0, 0.0,	10.15000	10.15000
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9	81201	[40.5, 40.5, 40.5, 40..	0.52500	1827.34500
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	2661	2661	[0.0, 0.0, 0.0, 0.0, 0.0,	255.95833	255.95833
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9	9	[0.0, 0.0, 0.0, 0.0, 0.0,	0.52500	0.52500
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	3661	3661	[0.0, 0.0, 0.0, 0.0, 0.0,	213.55833	213.55833
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9920	9920	[0.0, 0.0, 0.0, 0.0, 0.0,	1078.00000	1078.00000
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	7792	7792	[0.0, 0.0, 0.0, 0.0, 0.0,	454.53333	454.53333
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	1023	1023	[0.0, 0.0, 0.0, 0.0, 0.0,	59.67500	59.67500

Figura 7.1: Estructura actual de los datos de actividad (1/2 y 2/2)

Datos del sueño con la nueva estructura

User	Uid	Serv...	Date_time	Sleep_summary
user_workp...	odL0V7Tu...	test	1621202400.00000	[1621202760.0, 1621229640.0, 444.0]
mindcare_t...	EWnmc5u...	test	1621202400.00000	[1621200878.13082, 1621228478.13082, 450.0]
mindcare_t...	N5PdkLJZ...	test	1621202400.00000	[1621207180.324487, 1621232980.324487, 380.0]
mindcare_t...	lqUHq9R...	test	1621202400.00000	[1620951626.360464, 1620976226.360464, 390.0]
mindcare_t...	oZyp7e9PJ...	test	1621202400.00000	[1621203330.0, 1621226010.0, 323.0]
user_wellne...	pSpf0tMe...	wellness_...	1621202400.00000	[1621199070.0, 1621222920.0, 378.5]
pruebas7	rGRbkiclg...	test	1621202400.00000	[1621197354.711806, 1621229754.711806, 520.0]

Figura 7.2: Estructura actual de los datos de sueño

Una vez se ha dado un formato homogeneo a la informacion y se han seleccionado los datos útiles, el sistema intentará completar la información de actividad del usuario. La razón de ello es que los wearables únicamente dan información de las actividades que se han realizado, y es esta información la que se ha utilizado, en la estructuración de datos de actividad, para calcular los METs de los usuarios. Sin embargo, tanto el tiempo que esta una persdona durmiendo como en reposo tienen un consumo de METs asociado, por lo que hay que corregir estos valores. Para ello, lo primero que hace el sistema es realizar un LEFT JOIN con los datos de actividad y sueño, creando así una nueva estructura con los datos de actividad de los usuarios y los datos de sueño de aquellos que tengan.

Datos de la actividad y del sueño tras el join

User	Uid	Service	Date_time	SA_duration_slots	LA_duration_slots	MA_duration_slots
user_workp...	odL0V7Tu...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare_t...	EWnmc5u...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare_t...	EzNZ4ma...	uc3m_workpla...	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare_t...	N5PdkLJZ...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare_t...	XVqLk49L...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare_t...	XiS8SFC9S...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
test-condis	zqaAWK6...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare_t...	oZyp7e9PJ...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
user_wellne...	pSpf0tMe...	wellness_beta	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
pruebas7	rGRbkiclg...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
mindcare_t...	Ut36kG3d...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

IA_duration_slots	MA_IA_dur	Total_duration	METs_slots	MA_IA_METs	Total_METs	Sleep_summary
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	5231	5231	[0.0, 0.0, 0.0, 0.0, 0.0,	305.14167	305.14167	[1621202760.0, 1621229
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	397	80327	[40.5, 40.5, 40.5, 40..	23.15833	1821.58333	[1621200878.13082, 162
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	1955	1955	[0.0, 0.0, 0.0, 0.0, 0.0,	325.83333	325.83333	-1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	174	174	[0.0, 0.0, 0.0, 0.0, 0.0,	10.15000	10.15000	[1621207180.324487, 16
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9	81201	[40.5, 40.5, 40.5, 40..	0.52500	1827.34500	-1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	2661	2661	[0.0, 0.0, 0.0, 0.0, 0.0,	255.95833	255.95833	-1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9	9	[0.0, 0.0, 0.0, 0.0, 0.0,	0.52500	0.52500	-1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	3661	3661	[0.0, 0.0, 0.0, 0.0, 0.0,	213.55833	213.55833	[1621203330.0, 1621226
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9920	9920	[0.0, 0.0, 0.0, 0.0, 0.0,	1078.00000	1078.00000	[1621199070.0, 1621222
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	7792	7792	[0.0, 0.0, 0.0, 0.0, 0.0,	454.53333	454.53333	[1621197354.711806, 16
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	1023	1023	[0.0, 0.0, 0.0, 0.0, 0.0,	59.67500	59.67500	-1

Figura 7.3: Estructura actual de la actividad y sueño tras el join (1/2 y 2/2)

El siguiente paso para la corrección de los errores que se han mencionado previamente, es la llamada a las funciones de imputación, que serán las encargadas de alterar los valores erroneos y de actualizar el resto de datos con la nueva información.

Datos de la actividad y del sueño tras la imputación

User	Uid	Service	Date_time	SA_duration_slots	LA_duration_slots	MA_duration_slots
user_workp...	odL0V7Tu...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
mindcare_t...	EWnmc5u...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
mindcare_t...	EzNZ4ma...	uc3m_workpla...	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
mindcare_t...	N5PdkLJZ...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
mindcare_t...	XVqLk49L...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
mindcare_t...	XiS8SFC9S...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
test-condis	zqaAWK6...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
mindcare_t...	oZyp7e9PJ...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
user_wellne...	pSpf0tMe...	wellness_beta	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
pruebas7	rGRbkiclg...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
mindcare_t...	Ut36kG3d...	test	1621202400.00000	[1800, 1800, 1800, 1800, 1..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

IA_duration_slots	MA_IA_dur	Total_duration	METs_slots	MA_IA_METs	Total_METs	Sleep_summary
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	5231	86400.00000	[27.0, 27.0, 27.0, 27..	305.14167	1982.94417	[1621202760.0, 1621229
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	397	86402.00000	[27.0, 27.0, 27.0, 27..	23.15833	1821.18485	[1621200878.13082, 162
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	1955	86400.00000	[40.5, 40.5, 40.5, 40..	325.83333	2225.84583	-1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	174	86400.00000	[27.0, 27.0, 27.0, 27..	10.15000	1797.38257	[1621207180.324487, 16
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9	86400.00000	[40.5, 40.5, 40.5, 40..	0.52500	1944.32250	-1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	2661	86400.00000	[40.5, 40.5, 40.5, 40..	255.95833	2140.08583	-1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9	86400.00000	[40.5, 40.5, 40.5, 40..	0.52500	1944.32250	-1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	3661	86400.00000	[27.0, 27.0, 27.0, 27..	213.55833	1976.86083	[1621203330.0, 1621226
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	9920	10000086718.00000	[27.0, 27.0, 27.0, 27..	1078.00000	225002714.60500	[1621199070.0, 1621222
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	7792	86400.00000	[27.0, 27.0, 27.0, 27..	454.53333	2081.05299	[1621197354.711806, 16
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	1023	86400.00000	[40.5, 40.5, 40.5, 40..	59.67500	1980.65750	-1

Figura 7.4: Estructura actual de la actividad y sueño tras la imputación (1/2 y 2/2)

Con la información ya completada, el sistema generará los distintos indicadores del nivel de actividad física y sedentarismo a partir de esta.

Datos imputados de actividad y sueño junto a los indicadores

User	...	Serv...	Dat...	SA_...	LA_...	MA...	IA_...	MA_IA_dur	Total_d...	METs_slots	MA_IA_METs
user_wor...	odL0V7T...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	5231	86400.00000	[27.0, 27.0, 27.0, 27..	305.14167
mindcare...	EWnmc5...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	397	86402.00000	[27.0, 27.0, 27.0, 27..	23.15833
mindcare...	EzNZ4ma...	uc3m_wo...	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	1955	86400.00000	[40.5, 40.5, 40.5, 40..	325.83333
mindcare...	N5PdkLJ...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	174	86400.00000	[27.0, 27.0, 27.0, 27..	10.15000
mindcare...	XVqLk49...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	9	86400.00000	[40.5, 40.5, 40.5, 40..	0.52500
mindcare...	XiS8SFC9...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	2661	86400.00000	[40.5, 40.5, 40.5, 40..	255.95833
test-cond...	zqaAWK...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	9	86400.00000	[40.5, 40.5, 40.5, 40..	0.52500
mindcare...	oZyp7e9...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	3661	86400.00000	[27.0, 27.0, 27.0, 27..	213.55833
user_well...	pSpf0tMe...	wellness_...	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	9920	10000086718...	[27.0, 27.0, 27.0, 27..	1078.00000
pruebas7	rGRbkicl...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	7792	86400.00000	[27.0, 27.0, 27.0, 27..	454.53333
mindcare...	Ut36kG3...	test	16212024...	[1800, 18...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	[0, 0, 0, 0, ...	1023	86400.00000	[40.5, 40.5, 40.5, 40..	59.67500

Total_METs	Sleep_summary	activ_OMS_METs	activ_OMS_dur	activ_METs_ratio	activ_PAL	activ_SL_ratio	activ_SL
1982.94417	[1621202760.0, 1621229.	True	True	1.37704	Inactive (practic...	15.43204	True
1821.18485	[1621200878.13082, 162.	False	False	1.26471	Inactive (practic...	215.50378	True
2225.84583	-1	True	True	1.54573	Sedentary or ligh...	28.46292	True
1797.38257	[1621207180.324487, 16.	False	False	1.24818	Inactive (practic...	493.36782	True
1944.32250	-1	False	False	1.35022	Inactive (practic...	6399.00000	True
2140.08583	-1	True	True	1.48617	Sedentary or ligh...	20.64600	True
1944.32250	-1	False	False	1.35022	Inactive (practic...	6399.00000	True
1976.86083	[1621203330.0, 1621226.	True	True	1.37282	Inactive (practic...	22.51188	True
225002714.60500	[1621199070.0, 1621222.	-1	-1	-1.00000	-1	-1.00000	-1
2081.05299	[1621197354.711806, 16.	True	True	1.44518	Sedentary or ligh...	10.02156	True
1980.65750	-1	False	False	1.37546	Inactive (practic...	55.30499	True

Figura 7.5: Estructura actual de la actividad y sueño con los indicadores (1/2 y 2/2)

Finalmente, para no trabajar con información adicional, devuelve únicamente los indicadores con los identificadores de cada usuario a la función principal.

Indicadores de nivel de actividad y sedentarismo a partir de los datos de actividad

User	Date_time	Service	Uid	activ_OMS_METs	activ_OMS_dur	activ_METs_ratio	activ_PAL	activ_SL_ratio	activ_SL
user_workp...	1621202400.00000	test	odL0V7Tu...	True	True	1.37704	Inactive (practic...	15.43204	True
mindcare_t...	1621202400.00000	test	EWnmc5u...	False	False	1.26471	Inactive (practic...	215.50378	True
mindcare_t...	1621202400.00000	uc3m_workpla...	EzNZ4ma...	True	True	1.54573	Sedentary or ligh...	28.46292	True
mindcare_t...	1621202400.00000	test	N5PdkLJZ...	False	False	1.24818	Inactive (practic...	493.36782	True
mindcare_t...	1621202400.00000	test	XVqLk49L...	False	False	1.35022	Inactive (practic...	6399.00000	True
mindcare_t...	1621202400.00000	test	XiS8SFC9S...	True	True	1.48617	Sedentary or ligh...	20.64600	True
test-condis	1621202400.00000	test	zqaAWK6...	False	False	1.35022	Inactive (practic...	6399.00000	True
mindcare_t...	1621202400.00000	test	oZyp7e9PJ...	True	True	1.37282	Inactive (practic...	22.51188	True
user_wellne...	1621202400.00000	wellness_beta	pSpf0tMe...	-1	-1	-1.00000	-1	-1.00000	-1
pruebas7	1621202400.00000	test	rGRbkiclg...	True	True	1.44518	Sedentary or ligh...	10.02156	True
mindcare_t...	1621202400.00000	test	Ut36kG3d...	False	False	1.37546	Inactive (practic...	55.30499	True

Figura 7.6: Estructura de los indicadores al devolverlos la función

7.2. Generación de indicadores a partir de la tasa cardíaca y pasos

Una vez obtenidos los primeros indicadores, a partir de la actividad del usuario, el sistema obtendrá, de la misma manera que para el caso previo, los documentos con los datos de tasa cardíaca y pasos.

Datos de la colección detasa cardíaca recopilados

[['user_time_zone': 2, 'service': 'test', 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 17, 5, 55, 26, 729000, tzinfo=¡UTC¿), 'last_mod_ DatetimeWithNanoseconds(2021, 5, 18, 5, 44, 21, 115000, tzinfo=¡UTC¿), 'googleFitAPI': ['value': 96, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 22, 6, 12, tzinfo=¡UTC¿), 'value': 97, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 22, 16, 41, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 22, 24, 57, tzinfo=¡UTC¿), 'value': 50, 'value': 45, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 22, 35, 12, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 22, 45, 8, tzinfo=¡UTC¿), 'value': 48, 'value': 49, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 22, 54, 56, tzinfo=¡UTC¿), 'value': 49, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 23, 4, 58, tzinfo=¡UTC¿), 'value': 48, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 23, 15, 8, tzinfo=¡UTC¿), 'value': 51, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 23, 26, 8, tzinfo=¡UTC¿), 'value': 55, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 23, 34, 58, tzinfo=¡UTC¿), 'value': 149, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 23, 46, 28, tzinfo=¡UTC¿), 'value': 48, 'timestamp': DatetimeWithNanoseconds(2021, 5, 16, 23, 55, 10, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 0, 4, 54, tzinfo=¡UTC¿), 'value': 50, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 0, 14, 54, tzinfo=¡UTC¿), 'value': 47, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 0, 25, 4, tzinfo=¡UTC¿), 'value': 48, 'value': 49, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 0, 34, 57, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 0, 45, tzinfo=¡UTC¿), 'value': 49, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 0, 55, tzinfo=¡UTC¿), 'value': 47, 'value': 48, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 1, 5, 7, tzinfo=¡UTC¿), 'value': 50, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 1, 15, 6, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 1, 24, 56, tzinfo=¡UTC¿), 'value': 51, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 1, 35, 4, tzinfo=¡UTC¿), 'value': 49, 'value': 49, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 1, 45, 10, tzinfo=¡UTC¿), 'value': 47, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 1, 55, 12, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 2, 5, 12, tzinfo=¡UTC¿), 'value': 45, 'value': 47, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 2, 15, 13, tzinfo=¡UTC¿), 'value': 48, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 2, 25, 37, tzinfo=¡UTC¿), 'value': 50, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 2, 35, 10, tzinfo=¡UTC¿), 'value': 48, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 2, 45, 10, tzinfo=¡UTC¿), 'value': 48, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 2, 55, 8, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 3, 5, 10, tzinfo=¡UTC¿), 'value': 48, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 3, 15, 8, tzinfo=¡UTC¿), 'value': 49, 'value': 52, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 3, 24, 58, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021,

5, 17, 3, 35, 6, tzinfo=¡UTC¿), 'value': 47, 'value': 44, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 3, 45, 8, tzinfo=¡UTC¿), 'value': 47, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 3, 55, 5, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 4, 5, 21, tzinfo=¡UTC¿), 'value': 48, 'value': 53, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 4, 34, 56, tzinfo=¡UTC¿), 'value': 52, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 4, 46, 6, tzinfo=¡UTC¿), 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 4, 55, 42, tzinfo=¡UTC¿), 'value': 55, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 5, 5, 8, tzinfo=¡UTC¿), 'value': 47, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 5, 14, 58, tzinfo=¡UTC¿), 'value': 50, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 5, 24, 58, tzinfo=¡UTC¿), 'value': 50, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 5, 35, 8, tzinfo=¡UTC¿), 'value': 60, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 5, 46, 52, tzinfo=¡UTC¿), 'value': 63, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 5, 55, 18, tzinfo=¡UTC¿), 'value': 78, 'value': 68, 'timestamp': DatetimeWithNanoseconds(2021, 5, 17, 6, 5, 22, tzinfo=¡UTC¿), ...

Datos de la colección de pasos recopilados

[['total': 9460, 'last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 18, 5, 44, 23, 54000, tzinfo=¡UTC¿), 'withings': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 657, 1305, 2670, 40, 53, 69, 143, 16, 95, 12, nan, nan, 17, 31, 115, 87, 19, 6, 159, nan, nan, 48, 93, 15, 75, 1514, 1387, 620, 72, 28, 41, 73, nan], 'quality_score_slots': 1, 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 16, 22, 4, 45, 895000, tzinfo=¡UTC¿), 'date_time': '2021-05-17', 'service': 'test', 'steps': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 657, 1305, 2670, 40, 53, 69, 143, 16, 95, 12, 0, 0, 17, 31, 115, 87, 19, 6, 159, 0, 0, 48, 93, 15, 75, 1514, 1387, 620, 72, 28, 41, 73, 0], 'google_fit': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 672, 1284, 2729, 59, 38, 65, 143, 16, 95, 12, 0, 0, 17, 31, 112, 87, 19, 6, 159, 0, 0, 60, 100, 15, 75, 1507, 1397, 627, 63, 28, 0, 55, 0], 'user': 'user_workplace_01', 'user_uid': 'odL0V7Tu4qfJ2eqrsEgfE3rR6Kw2', 'googleFitAPI': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 658, 1288, 2727, 59, 38, 65, 143, 16, 95, 12, 0, 0, 17, 31, 112, 87, 19, 6, 159, 0, 0, 60, 100, 15, 75, 1507, 1400, 627, 63, 28, 0, 55, 0]], ['quality_score_slots': 1, 'service': 'test', 'app': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 251, 290, 0, 0, 0, 0, 0, 397, 0, 0, 0, 513, 404, 0, 0, 0, 0, 0, 0, 0, 0, 0, 195, 174, 0, 13, 17, 0, 0, 0], 'user': 'mindcare_test_AAR', 'last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 18, 6, 39, 56, 54000, tzinfo=¡UTC¿), 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 16, 22, 3, 31, 567000, tzinfo=¡UTC¿), 'user_uid': 'EWnmc5uZHTVtC0mRZI8JswIlTlt1', 'HealthKit': [74, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 40, 17, nan, 583, 578, 55, 10, 10, 232, 40, 897, nan, nan, nan, 1042, 753, nan, 444, 8, nan, nan, 26, 6, 11, 13, nan, 446, 431, nan, 23, 88, 40, nan, nan], 'date_time': '2021-05-17', 'steps': [74, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 40, 17, 0, 583, 578, 55, 10, 10, 232, 40, 900, 0, 0, 0, 1046, 753, 0, 444, 8, 0, 0, 26, 6, 11, 13, 0, 446, 432, 0, 23, 88, 40, 0, 0], 'total': 5875], ['service': 'uc3m_workplace', 'garmin': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 5482, nan, nan, nan, nan, nan, nan, nan], 'steps': [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, 5482, nan, nan, nan, nan, nan, nan, nan], 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 17, 18, 36, 28, 565000, tzinfo=¡UTC¿), 'user_uid': 'EzNZ4maadwdDi3ASedDpGAusOd02', 'user': 'mindcare_test_26', 'quality_score_slots': 0.02, 'date_time': '2021-05-17', 'last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 17, 18, 36, 28, 565000, tzinfo=¡UTC¿)], [], ['service': 'test', 'creation_datetime': DatetimeWithNanoseconds(2021, 5, 17, 9, 29, 33, 488000, tzinfo=¡UTC¿), 'google_fit': [55, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 222, 108, 31, 0, 0, 0, 0, 0, 0, 36, 12, 0, 0, 0, 0, 0, 0, 0, 14, 49, 0, 11, 62, 26, 0, 42, 174, 44, 10, 32, 0, 11, 80], 'date_time': '2021-05-17', 'last_mod_datetime': DatetimeWithNanoseconds(2021, 5, 18, 6, 47, 4, 922000, tzinfo=¡UTC¿), 'quality_score_slots': 1, 'user': 'mindcare_test_4', 'steps': [55, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 222, 108, 31, 0, 0, 0, 0, 0, 0, 36, 12, 0, 0, 0, 0, 0, 0, 0, 14, 49, 0, 11, 62, 26, 0, 42, 174, 44, 10, 32, 0, 11, 80], 'user_uid': 'N5PdkLJZxQZuLbrlQH8QuStskyr2', 'total': 1019], ...

Y estructurará la información con las funciones respectivas para poder trabajar con los datos.

Datos de la tasa cardiaca con la nueva estructura

User	Uid	Service	Date_time	HR_slots
user_workplace_01	odL0V7Tu...	test	1621202400.00000	[[96, '22:04:57'], [..

Figura 7.7: Estructura actual de los datos de tasa cardíaca

Datos de los pasos con la nueva estructura

User	Uid	Service	Date_time	Steps_slots	Total_steps
user_workp...	odL0V7Tu...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	9460.00000
mindcare_t...	EWnmc5u...	test	1621202400.00000	[74, 0, 0, 0, 0, 0, 0, 0,.	5875.00000
mindcare_t...	EzNZ4ma...	uc3m_workpla...	1621202400.00000	[nan, nan, nan, nan,..	5482.00000
mindcare_t...	N5PdkLJZ...	test	1621202400.00000	[55, 0, 0, 0, 0, 0, 0, 0,.	1019.00000
mindcare_t...	OSnT3MC...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	112.00000
mindcare_t...	XVqLk49L...	test	1621202400.00000	[26, 0, 0, 0, 0, 0, 0, 0,.	492.00000
mindcare_t...	XiS8SFC9S...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	1963.00000
mindcare_t...	IqUHq9R...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	304.00000
test-condis	zqaAWK6...	test	1621202400.00000	[26, 0, 0, 0, 0, 0, 0, 0,.	492.00000
mindcare_t...	oZyp7e9PJ...	test	1621202400.00000	[27, 9, 0, 0, 0, 0, 0, 0,.	8836.00000
user_wellne...	pSpf0tMe...	wellness_beta	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	9819.00000
pruebas7	rGRbkiclg...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	9471.00000
mindcare_t...	Ut36kG3d...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	1382.00000

Figura 7.8: Estructura actual de los datos de pasos

Una vez estructurados, para generar los indicadores de tasa cardíaca, es necesario realizar un left join de tasa cardíaca con los pasos, ya que estos se utilizan en el calculo del BMR de los usuarios (el cual se usa a su vez para el calculo de los indicadores). Sin embargo, a diferencia de los datos del sueño, que no se utilizan más que para la imputación, los pasos sí que tendrán uso en la siguiente parte del proceso.

Datos de la tasa cardíaca (con los pasos) tras el join

User	Uid	Service	Date_time	HR_slots	Steps_slots
user_workp...	odL0V7Tu...	test	1621202400.00000	[[96, '22:04:57'], [..	[0, 0, 0, 0, 0, 0, 0, 0, ..

Figura 7.9: Estructura actual de los datos de tasa cardíaca tras el join

El sistema tiene ya toda la información que necesita y con la estructura adecuada, por lo que calculará por una parte, los indicadores con los datos de tasa cardíaca y pasos, y por otra, los indicadores a partir de únicamente los pasos de los usuarios. Esto dará lugar a las siguientes estructuras:

Datos de la tasa cardíaca (con los pasos) junto a sus indicadores

User	Uid	Service	Date_time	HR_slots	Steps_slots	heart_BMR
user_workp...	odL0V7Tu...	test	1621202400.00000	[[96, '22:04:57'], [..	[0, 0, 0, 0, 0, 0, 0, 0, ..	68.53000

heart_AVG_slots	heart_MET_slots	heart_total_METs	heart_METs_ratio	heart_PAL
[48.33, 48.33, 49.67, 48.3..	[37.9734, 37.9734, 37.97..	2340.60266	1.62542	Sedentary or ligh...

Figura 7.10: Estructura actual de la tasa cardíaca con los indicadores (1/2 y 2/2)

Datos de los pasos y sus indicadores

User	Uid	Service	Date_time	Steps_slots	Total_steps	steps_PAL
user_workp...	odL0V7Tu...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	9460.00000	Active
mindcare_t...	EWnmc5u...	test	1621202400.00000	[74, 0, 0, 0, 0, 0, 0, 0,	5875.00000	Little active
mindcare_t...	EzNZ4ma...	uc3m_workpla...	1621202400.00000	[nan, nan, nan, nan,..	5482.00000	Little active
mindcare_t...	N5PdkLJZ...	test	1621202400.00000	[55, 0, 0, 0, 0, 0, 0, 0,	1019.00000	Sedentary
mindcare_t...	OSnT3MC...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	112.00000	Inactive (practica...
mindcare_t...	XVqLk49L...	test	1621202400.00000	[26, 0, 0, 0, 0, 0, 0, 0,	492.00000	Inactive (practica...
mindcare_t...	XiS8SFC9S...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	1963.00000	Sedentary
mindcare_t...	lqUHq9R...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	304.00000	Inactive (practica...
test-condis	zqaAWK6...	test	1621202400.00000	[26, 0, 0, 0, 0, 0, 0, 0,	492.00000	Inactive (practica...
mindcare_t...	oZyp7e9PJ...	test	1621202400.00000	[27, 9, 0, 0, 0, 0, 0, 0,	8836.00000	Active
user_wellne...	pSpf0tMe...	wellness_beta	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	9819.00000	Active
pruebas7	rGRbkiclg...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	9471.00000	Active
mindcare_t...	Ut36kG3d...	test	1621202400.00000	[0, 0, 0, 0, 0, 0, 0, 0, ..	1382.00000	Sedentary

Figura 7.11: Estructura actual de los pasos con los indicadores (1/2 y 2/2)

Finalmente, para no trabajar con información adicional, devuelve únicamente los indicadores con los identificadores de cada usuario a la función principal.

Indicadores de nivel de actividad y sedentarismo a partir de los datos de la tasa cardíaca y los pasos

User	Uid	Service	Date_time	Total_steps	steps_PAL	heart_METs_ratio	heart_PAL
user_workp...	odL0V7Tu...	test	1621202400.00000	9460.00000	Active	1.62542	Sedentary or ligh...
mindcare_t...	EWnmc5u...	test	1621202400.00000	5875.00000	Little active	-1.00000	-1
mindcare_t...	EzNZ4ma...	uc3m_workpla...	1621202400.00000	5482.00000	Little active	-1.00000	-1
mindcare_t...	N5PdkLJZ...	test	1621202400.00000	1019.00000	Sedentary	-1.00000	-1
mindcare_t...	OSnT3MC...	test	1621202400.00000	112.00000	Inactive (practica...	-1.00000	-1
mindcare_t...	XVqLk49L...	test	1621202400.00000	492.00000	Inactive (practica...	-1.00000	-1
mindcare_t...	XiS8SFC9S...	test	1621202400.00000	1963.00000	Sedentary	-1.00000	-1
mindcare_t...	lqUHq9R...	test	1621202400.00000	304.00000	Inactive (practica...	-1.00000	-1
test-condis	zqaAWK6...	test	1621202400.00000	492.00000	Inactive (practica...	-1.00000	-1
mindcare_t...	oZyp7e9PJ...	test	1621202400.00000	8836.00000	Active	-1.00000	-1
user_wellne...	pSpf0tMe...	wellness_beta	1621202400.00000	9819.00000	Active	-1.00000	-1
pruebas7	rGRbkiclg...	test	1621202400.00000	9471.00000	Active	-1.00000	-1
mindcare_t...	Ut36kG3d...	test	1621202400.00000	1382.00000	Sedentary	-1.00000	-1

Figura 7.12: Estructura de los indicadores al devolverlos la función

7.3. Agrupación de indicadores

Tras haber obtenido los indicadores de todas las fuentes de datos, el sistema los unifica, dando lugar a una única estructura con todos ellos.

User	Date_time	Service	Uid	activ_OMS_METs	activ_OMS_dur	activ_METs_ratio
user_workp...	1621202400.00000	test	odL0V7Tu...	True	True	1.37704
mindcare_t...	1621202400.00000	test	EWnmc5u...	False	False	1.26471
mindcare_t...	1621202400.00000	uc3m_workpla...	EzNZ4ma...	True	True	1.54573
mindcare_t...	1621202400.00000	test	N5PdkLJZ...	False	False	1.24818
mindcare_t...	1621202400.00000	test	XVqLk49L...	False	False	1.35022
mindcare_t...	1621202400.00000	test	XiS8SFC9S...	True	True	1.48617
test-condis	1621202400.00000	test	zqaAWK6...	False	False	1.35022
mindcare_t...	1621202400.00000	test	oZyp7e9PJ...	True	True	1.37282
user_wellne...	1621202400.00000	wellness_beta	pSpf0tMe...	-1	-1	-1.00000
pruebas7	1621202400.00000	test	rGRbkiclg...	True	True	1.44518
mindcare_t...	1621202400.00000	test	Ut36kG3d...	False	False	1.37546
mindcare_t...	1621202400.00000	test	OSnT3MC...	-1	-1	-1.00000
mindcare_t...	1621202400.00000	test	lqUHq9R...	-1	-1	-1.00000

activ_PAL	activ_SL_ratio	activ_SL	Total_steps	steps_PAL	heart_METs_ratio	heart_PAL
Inactive (practically im...	15.43204	True	9460.00000	Active	1.62542	Sedentary or light active
Inactive (practically im...	215.50378	True	5875.00000	Little active	-1.00000	-1
Sedentary or light activ...	28.46292	True	5482.00000	Little active	-1.00000	-1
Inactive (practically im...	493.36782	True	1019.00000	Sedentary	-1.00000	-1
Inactive (practically im...	6399.00000	True	492.00000	Inactive (practica...	-1.00000	-1
Sedentary or light activ...	20.64600	True	1963.00000	Sedentary	-1.00000	-1
Inactive (practically im...	6399.00000	True	492.00000	Inactive (practica...	-1.00000	-1
Inactive (practically im...	22.51188	True	8836.00000	Active	-1.00000	-1
-1	-1.00000	-1	9819.00000	Active	-1.00000	-1
Sedentary or light activ...	10.02156	True	9471.00000	Active	-1.00000	-1
Inactive (practically im...	55.30499	True	1382.00000	Sedentary	-1.00000	-1
-1	-1.00000	-1	112.00000	Inactive (practica...	-1.00000	-1
-1	-1.00000	-1	304.00000	Inactive (practica...	-1.00000	-1

Figura 7.13: Estructura de todos los indicadores agrupados

El sistema ya ha cumplido su función, por lo que transformará los datos de cada usuario (que se encuentran ya agrupados en la nueva estructura) en un documento con

el formato adecuado para almacenarse en Firestore y los guardará en la colección asignada para los indicadores de actividad física y sedentarismo. A continuación, se muestra un ejemplo de como se estructura la información al final del sistema para generar el documento de Firestore.

{'user': 'user_workplace_01',
'date_time': '2021-06-28',
'creation_datetime': datetime.datetime(2021, 6, 29, 15, 40, 8, 224382),
'last_mod_datetime': datetime.datetime(2021, 6, 29, 15, 40, 8, 224382),
'user_uid': 'odL0V7Tu4qfJ2eqrsEgfE3rR6Kw2',
'service': 'test',
'steps': 'PAL': 'Sedentary', 'n_steps': 2325,
'activity': 'OMS_METs': True, 'OMS_duration': True, 'METs_ratio': 1.3988726851851854,
'PAL': 'Inactive (practically immobile)', 'sedentary_ratio': 28.327902240325866, 'is_seden -tary': True, 'heart_rate': 'METs_ratio': 1.455087437500001, 'PAL': 'Sedentary or light active'}

Capítulo 8

Alojamiento del sistema en la nube

Una vez desarrollado el código, ya funcional y optimizado, era necesaria la subida a la nube para comenzar a trabajar, de manera automática e ininterrumpida, sobre un flujo de datos constante.

En esta sección se explicará de manera abstracta el proceso que se ha seguido para el alojamiento, buscando una explicación genérica que pueda ser de utilidad para comprender los pasos necesarios para virtualizar un proyecto de software mediante Docker y Kubernetes.

El objetivo final del todo el proceso es lanzar una Docker Image en un pod del proyecto de Kubernetes de EB2, dando lugar a un contenedor con nuestro sistema ejecutándose. Sin embargo, para ello es necesario crear la propia Docker Image, para lo cual hace falta un Dockerfile.

Creación del dockerfile

En este archivo, escrito a mano, hay que especificar todos los elementos necesarios que necesitaría el sistema para ser ejecutado en el contenedor. La version de Python o el ejecutable del proyecto son algunos de los aspectos esenciales que hay que incluir. Habitualmente se incluyen todos los requerimientos de librerías utilizadas por el proyecto en un documento aparte, llamado Requirements.txt, al cual también se hace referencia en el Dockerfile.

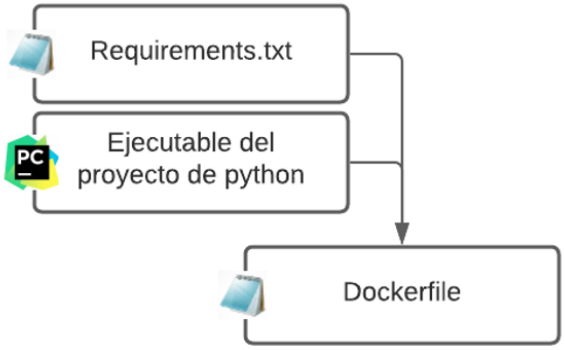

Figura 8.1: Diagrama general de la creación de un Dockerfile

Creación de la Docker Image

Una vez terminado el Dockerfile, podremos utilizarlo para ordenar a Docker, desde el directorio en el que se encuentre nuestro proyecto, la creación de la Docker Image. Para ello, Docker utilizará el propio proyecto de software, el documento con los requerimientos especificados en el Dockerfile y el Dockerfile, dando lugar a la esperada Docker Image.

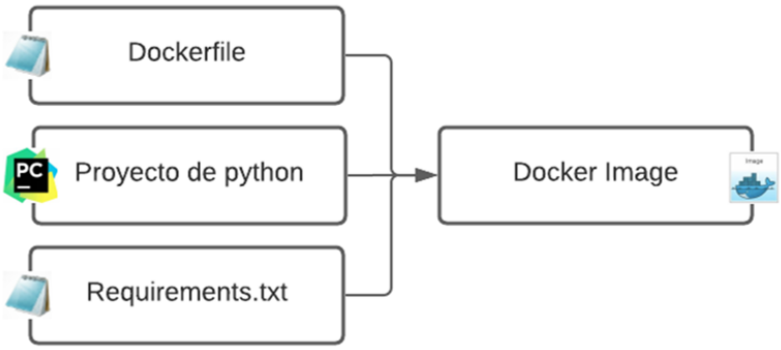

Figura 8.2: Diagrama general de la creación de una Docker Image

Comunicacion con Kubernetes

La Docker Image ya está lista para ser ejecutada, sin embargo, Kubernetes no tiene acceso a esta. Para resolver este problema, será necesario enviarla desde Docker hasta el Registro de Contenedores del proyecto de Kubernetes. Antes de enviarla, es importante asignar una etiqueta a la imagen, ya que el contenedor tiene una gran cantidad de ellas. Normalmente, al realizar este proceso cada vez que se desarrolla una nueva version de un software que ya está en ejecución y se quiere actualizar, se suele utilizar un sistema de etiquetas que permita identificar tanto la función del código como su version.

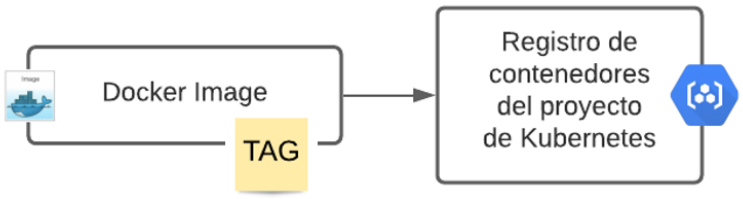

Figura 8.3: Diagrama general de la comunicación entre Docker y Kubernetes

Creación del CronJob

Habiendo recibido ya la Docker Image, para poder ejecutarla y crear contenedores, Kubernetes utiliza los CronJobs. Estos, son objetos que incluyen en sus parámetros la Docker Image a utilizar, que ahora sí posee Kubernetes, y permiten especificar otras particularidades de su lanzamiento como cuándo se ejecuta, cuánto puede tardar en iniciarse, etc.

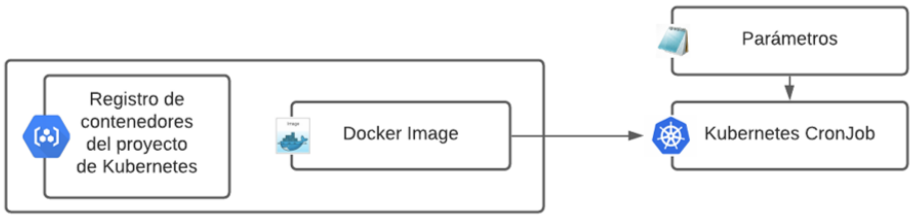

Figura 8.4: Diagrama general de la creación de un CronJob

Lanzamiento del CronJob y creación del contenedor con el sistema

Para lanzar el CronJob que se haya creado con la Docker Image y crear el contenedor pertinente, es necesario ejecutarlo en el cluster, nodo y pod del proyecto de Kubernetes que nos interese. Para ello, podemos navegar hasta estos a través de la consola de Google Cloud del proyecto y, una vez estemos en el lugar deseado, lanzaremos el CronJob que, siguiendo los parámetros establecidos en su creación, ejecutará su imagen.

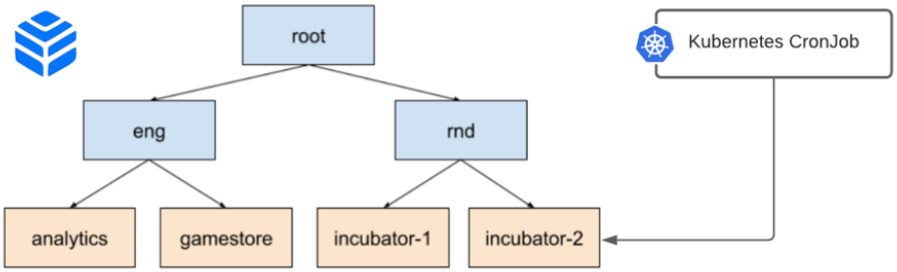

Figura 8.5: Diagrama general de la ejecución de un CronJob

En este punto, nuestro sistema ya se encontrará alojado en la Nube y únicamente será necesario monitorizar su funcionamiento.

Capítulo 9

Coste del proyecto

En esta sección se pretende estimar el desembolso económico que sería necesario para desarrollar el proyecto. Para ello, se valorarán los costes materiales y el coste por tiempo de trabajo de la persona encargada del desarrollo del proyecto. Sin embargo, no se valorarán los costes posteriores derivados del uso del sistema, ni el coste de almacenamiento de los datos, ya que ambos valores dependerán de la integración (en servidores propios, cloud, etc.) que se quiera hacer después del desarrollo.

Coste de equipos o material

Que incluye los gastos derivados de los equipos utilizados para el desarrollo del proyecto y elementos relacionados con estos.

A lo largo del proyecto se ha utilizado un ordenador portátil de gama media-alta con conexión a Internet. Se estima que para la realización del proyecto han sido necesarias un total de 480 horas, lo que equivale a 60 días laborales de 8 horas y a aproximadamente 3 meses de trabajo.

Equipo	Precio	Vida útil	Uso	Total
Ordenador de gama media - alta	950 €	5 años	3 meses	47'5 €
Conexión a internet	35 €	-	3 meses	105 €
	Coste total derivado de equipos			**152'5 €**

Figura 9.1: Coste de equipos o material

Coste por tiempo de trabajo

Que incluye el desembolso derivado de las horas de trabajo de la persona responsable del desarrollo del proyecto.

Utilizando el número de horas mencionadas en el apartado previo, y los salarios indicados en páginas especializadas (Jobted y Glassdoor), se ha obtenido la siguiente estimación.

Profesión	€ / hora	Nº horas	Total
Ingeniero Informático	16 €	480	7680 €
	Coste total derivado del tiempo de trabajo		**7680 €**

Figura 9.2: Coste por tiempo de trabajo

Coste total de ejecución material

Consistente en la suma del coste de material y del coste de tiempo de trabajo.

Categoría	Total
Coste total derivado de equipos	152'5 €
Coste total derivado del tiempo de trabajo	7680 €
Coste total de ejecución material	**7832'5 €**

Figura 9.3: Coste total de ejecución material

Gastos generales y beneficio industrial

Que incluyen los gastos necesarios que cubren las instalaciones en las que se desempeña el trabajo, junto a otros gastos adicionales. Este es equivalente a un 20 % del coste total de ejecución material.

Categoría	Total
Coste total de ejecución material	7832'5 €
Gastos generales y beneficio industrial	**1566'5 €**

Figura 9.4: Gastos generales y beneficio industrial

Presupuesto de ejecución por contrata

Consistente en la suma de los gastos generales y beneficio industrial y el coste total de ejecución material.

Categoría	Total
Coste total de ejecución material	7832'5 €
Gastos generales y beneficio industrial	1566'5 €
Presupuesto de ejecución por contrata	**9399 €**

Figura 9.5: Presupuesto de ejecución por contrata

Importe total

Consistente en la suma del presupuesto de ejecución por contrata más el impuesto del 21 % de I.V.A.

Categoría	Total
Presupuesto de ejecución por contrata	9399 €
Importe total (incluido 21% I.V.A)	**11372'79 €**

Figura 9.6: Importe total

Es importante volver a recalcar que los costes derivados del uso del sistema no han sido incluidos debido a las múltiples posibilidades que pueden darse en su implantación.

Capítulo 10

Resumen, conclusiones y trabajos futuros

En esta sección se incluirá un resumen de las tareas realizadas, las conclusiones obtenidas y posibles mejorías o trabajos futuros relacionados con el proyecto.

10.1. Resumen

Al inicio del proyecto, fue necesaria una amplia investigación del estado del arte en la medición de la actividad física, del sedentarismo y de su impacto en la salud de las personas. Para ello, fue necesario también profundizar en los distintos conceptos médicos que permitieron establecer las relaciones entre los distintos sistemas de medición.

Una vez obtenidos los conocimientos del campo necesarios, se investigaron las posibles mediciones que se podían obtener a través de los wearables y la recopilación de estas a través de APIs, lo que permitió analizar a qué datos del fenotipo digital de las personas se tenía acceso y cuáles se podían utilizar para generar los indicadores. Además, se examinó la conexión de las bases de datos de Cloud Firestore de EB2 (que se utilizarían como fuente de datos del sistema a desarrollar) con las APIs que se pretendía utilizar, para valorar si se recogían los datos pertinentes y en el formato adecuado para su futuro uso.

Teniendo en cuenta las necesidades del proyecto a la hora de gestionar los volúmenes de datos generados por los wearables de un gran número de personas, se estableció la necesidad de alojar el sistema en la nube, eligiendo para ello el uso de Docker y Kubernetes. Para el desarrollo del código fue elegido el lenguaje de programación Python (con el IDE Pycharm), por la limpieza de su código y el gran numero de librerías que incluye.

Comprendidas las nociones médicas, establecidas las fuentes de datos y elegidas las herramientas a utilizar, se comenzó con el proceso de desarrollo del sistema generador de indicadores. Para ello, se investigó el funcionamiento de las bases NoSQL de Firestore y de las librerías de Python que permitirían establecer la conexión entre el sistema y las bases de datos. Por otro lado, también se exploraron las librerías principales de Python para la gestión de grandes volúmenes de datos.

El desarrollo en sí consistió en un sistema informático que, a partir de los datos recopilados de la huella digital de las personas, permite generar una estimación o indicador

del nivel de actividad y sedentarismo de esta (junto a otros indicadores auxiliares), demostrando así una nueva forma de medir estos valores de manera precisa y automática.

Sin embargo, se buscaba que esta funcionalidad fuese también escalable, por lo que, tras un estudio del funcionamiento de Docker y Kubernetes, se alojó el sistema en un contenedor en la nube (en la plataforma de EB2) que ejecutaba este de manera automática y le aportaba los recursos que necesitase dependiendo del volumen de datos que tratase. Para mejorar el rendimiento del sistema, se realizó también un proceso de optimización de este, tanto a nivel de código como estructural.

Una vez hecho esto, el desarrollo e implementación del sistema generador de indicadores de actividad física y sedentarismo, a partir del fenotipo digital, como servicio en la nube, se dio por finalizado.

10.2. Conclusiones

Durante la carrera se nos ha dado la oportunidad de desarrollar múltiples aplicaciones en las que se simulaba una interacción con el mundo real, pero siempre ha sido ese el límite al que se ha llegado, una simulación. Sin embargo, este proyecto me ha dado la oportunidad de aplicar los conocimientos adquiridos para aportar una ayuda a un problema real de gravedad, con las dificultades y motivación que ello conlleva.

El proyecto en sí presentaba varias dificultades principales, entre las cuales se pueden destacar:

- El desconocimiento previo del campo sobre el que se aplicaría el sistema y la importancia de la integridad y precision de los resultados.
- La falta de experiencia previa con bases de datos no relacionales, sistemas de contenedores, orquestradores y alojamiento en la nube.
- La dificultad de desarrollar un sistema que funcionase, en su mayor medida, como una caja negra para permitir su uso en distintas situaciones modificando únicamente la entrada y salida de los datos.
- La complicación de optimizar el funcionamiento lo máximo posible (para minimizar los gastos derivados del uso del sistema) derivada de la poca experiencia con librerías de gestión de datos de Python y, una vez más, de los factores mencionados en el segundo punto.

Sin embargo, gracias a la abundante, aunque a veces recóndita, documentación que hay disponible en Internet, se han conseguido superar estos problemas, adquiriendo en el proceso un gran número de conocimientos de gran utilidad. Además de las dificultades previas, me gustaría recalcar la importancia de establecer los requisitos y funcionalidades del sistema de manera clara y concisa antes de iniciar el desarrollo y, en caso de querer ampliar estas, plantear una nueva estrategia que permita incorporarlas de manera óptima al plan inicial. Las pocas ocasiones en las que se consideró que el añadido de una funcionalidad era tan mínimo que no influiría y no era necesario el planificarlo, estas derivaron en perdidas de tiempo posteriores y aparición de errores que fue necesario solventar.

Por otra parte, los conocimientos adquiridos durante los últimos cuatro años relativos a metodologías de trabajo, buenas prácticas en el desarrollo de código, la correcta implementación de flujos de información, etc. han sido de gran ayuda para llevar a cabo este proyecto, el cual no habría sido posible sin haberlos recibido. Esto, sumado al aprendizaje del funcionamiento de Cloud Firestore, Docker, Kubernetes , las APIs y las herramientas que pone a disposición Python, constituye una formación muy completa para llevar a cabo proyectos reales como este.

En conclusión, me gustaría recalcar la importancia que tiene el desarrollo de proyectos que permitan aumentar el nivel de conocimiento acerca de la salud de las personas. Se ha demostrado la posibilidad de desarrollar un sistema que utiliza una nueva fuente de datos en auge para determinar indicadores de la actividad física de las personas, pero esto es sólo una de las puertas que se abren con el uso de estos datos. Si se consiguen explotar de manera adecuada, se podría llegar a alcanzar un nuevo hito en la salud a nivel global, ya que se contaría con más datos de las personas de los que nunca se han tenido.

10.3. Trabajos futuros

Como futuros trabajos relacionados con el proyecto se pueden distinguir tres posibles vías de trabajo: el aumento de funcionalidades, la facilitación de la implementación y el uso del sistema con una API.

10.3.1. Aumento de funcionalidades

Dentro de esta vía se consideraría el uso de los datos de los wearables para obtener indicadores o análisis de otros campos de la salud. Esto presenta tantas posibilidades que se darán únicamente tres breves ejemplos, relacionadas con la actividad física, que permitan dar una idea del alcance que podría tener el proyecto:

- Generación de modelos de la tasa cardíaca para: valorar alteraciones que puedan ser peligrosas para el usuario o analizar el impacto de la actividad en el ritmo cardíaco.
- Generación de modelos del sueño del usuario para: valorar la calidad del sueño de este o el impacto de la actividad sobre el mismo.
- Clasificación de usuarios según niveles de actividad y sedentarismo para plantear estudios según las categorías generadas.

Pese a haber dado únicamente ejemplos relacionados con la actividad física, el abanico de posibles mediciones o análisis es prácticamente ilimitado.

10.3.2. Facilitación de implementación

El sistema se ha desarrollado como una caja negra que pueda ser implementada en cualquier arquitectura o proyecto. Sin embargo, en la actualidad únicamente actuaría como una caja negra *pura* en proyectos que utilicen Cloud Firestore, ya que se realiza la

conexión con este tipo de bases de datos. Esta vía se centraría en ampliar los posibles formatos de los datos de entrada y salida del sistema para permitir adaptarlo a otros tipos de bases de datos alternativas. De esta forma, se daría a elegir al encargado de implementar el sistema en un proyecto, que formato desea utilizar.

10.3.3. Uso del sistema con una API

Otra posibilidad sería el desarrollo de una API que permitiese el acceso público a las funcionalidades del sistema, y diese la posibilidad, a todo el que estuviese interesado, de obtener sus indicadores de nivel de actividad física y sedentarismo a partir de los datos que hayan generado sus dispositivos.

Bibliografía

[1] B. E. Ainsworth, W. L. Haskell, S. D. Herrmann, N. Meckes, D. R. Bassett, C. Tudor-Locke, J. L. Greer, J. Vezina, M. C. Whitt-Glover, and A. S. Leon, "2011 compendium of physical activities: a second update of codes and MET values," vol. 43, no. 8, pp. 1575–1581.

[2] J. A. Bragada, P. M. Pedro, C. S. Vasques, M. B. Tiago, and P. L. Vítor, "Net heart rate to prescribe physical activity in middle-aged to older active adults," vol. 8, no. 4, pp. 616–621.

[3] Human energy requirements. [Online]. Available: http://www.fao.org/3/y5686e/y5686e00.htm

[4] C. Tudor-Locke and D. Bassett, "How many steps/day are enough? preliminary pedometer indices for public health," vol. 34, pp. 1–8.

[5] M. Naghavi *et al.*, "Global, regional, and national age-sex specific mortality for 264 causes of death, 1980–2016: a systematic analysis for the global burden of disease study 2016," vol. 390, no. 10100, pp. 1151–1210, publisher: Elsevier. [Online]. Available: https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(17)32152-9/abstract

[6] F. C. Bull, S. S. Al-Ansari, S. Biddle, K. Borodulin, M. P. Buman, G. Cardon, C. Carty, J.-P. Chaput, S. Chastin, R. Chou, P. C. Dempsey, L. DiPietro, U. Ekelund, J. Firth, C. M. Friedenreich, L. Garcia, M. Gichu, R. Jago, P. T. Katzmarzyk, E. Lambert, M. Leitzmann, K. Milton, F. B. Ortega, C. Ranasinghe, E. Stamatakis, A. Tiedemann, R. P. Troiano, H. P. van der Ploeg, V. Wari, and J. F. Willumsen, "World health organization 2020 guidelines on physical activity and sedentary behaviour," vol. 54, no. 24, pp. 1451–1462.

[7] World Health Organization, *Global status report on noncommunicable diseases 2014.*, OCLC: 905955517.

[8] K. C. Furlanetto, L. Donária, L. P. Schneider, J. R. Lopes, M. Ribeiro, K. B. Fernandes, N. A. Hernandes, and F. Pitta, "Sedentary behavior is an independent predictor of mortality in subjects with COPD," vol. 62, no. 5, pp. 579–587.

[9] R. Guthold, G. A. Stevens, L. M. Riley, and F. C. Bull, "Worldwide trends in insufficient physical activity from 2001 to 2016: a pooled analysis of 358 population-based surveys with 1·9 million participants," vol. 6, no. 10, pp. e1077–e1086, publisher: Elsevier. [Online]. Available: https://www.thelancet.com/journals/langlo/article/PIIS2214-109X(18)30357-7/abstract

[10] WHO. Cuestionario mundial sobre actividad física. [Online]. Available: https://www.who.int/ncds/surveillance/steps/GPAQ_ES.pdf

[11] ——. Global physical activity questionnaire (analysis guide). [Online]. Available: https://www.who.int/ncds/surveillance/steps/resources/GPAQ_Analysis_Guide.pdf

[12] S. Ranger. What is cloud computing? everything you need to know about the cloud explained. [Online]. Available: https://www.zdnet.com/article/what-is-cloud-computing-everything-you-need-to-know-about-the-cloud/

[13] ¿qué es el cloud computing? [Online]. Available: https://www.redhat.com/es/topics/cloud

[14] What is virtualization and how does it work? [Online]. Available: https://searchservervirtualization.techtarget.com/definition/virtualization

[15] Virtualization. Page Version ID: 1023577731. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Virtualization&oldid=1023577731

[16] Cloud containers: What are they? why do they work? [Online]. Available: https://www.ukfast.co.uk/blog/2020/01/13/what-are-and-why-cloud-containers/

[17] Contenedores: qué son y cuáles son sus ventajas. [Online]. Available: https://cloud.google.com/containers?hl=es

[18] Orquestación de contenedores. [Online]. Available: https://blog.techdata.com/ts/latam/orquestaci%C3%B3n-de-contenedores

[19] Heidilohr. Información general sobre la orquestación de contenedores de windows. [Online]. Available: https://docs.microsoft.com/es-es/virtualization/windowscontainers/about/overview-container-orchestrators

[20] P. Carretero, J. J. Campana-Montes, and A. Artes-Rodriguez, "Ecological momentary assessment for monitoring risk of suicide behavior," vol. 46, pp. 229–245.

[21] A. Porras-Segovia, R. M. Molina-Madueño, S. Berrouiguet, J. López-Castroman, M. L. Barrigón, M. S. Pérez-Rodríguez, J. H. Marco, I. Díaz-Oliván, S. de León, P. Courtet, A. Artés-Rodríguez, and E. Baca-García, "Smartphone-based ecological momentary assessment (EMA) in psychiatric patients and student controls: A real-world feasibility study," vol. 274, pp. 733–741. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165032720304407

[22] Docker explained – an introductory guide to docker - DZone cloud. [Online]. Available: https://dzone.com/articles/docker-explained-an-introductory-guide-to-docker

[23] Y. Mulonda. What is docker? "in simple english". [Online]. Available: https://blog.usejournal.com/what-is-docker-in-simple-english-a24e8136b90b

[24] ¿qué es kubernetes? | microsoft azure. [Online]. Available: https://azure.microsoft.com/es-es/topic/what-is-kubernetes/

[25] ¿qué es kubernetes? Section: docs. [Online]. Available: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

[26] T. Rodríguez. De docker a kubernetes: entendiendo qué son los contenedores y por qué es una de las mayores revoluciones de la industria del desarrollo. [Online]. Available: https://www.xataka.com/otros/docker-a-kubernetes-entendiendo-que-contenedores-que-mayores-revoluciones-industria-desarrollo

[27] Comparación entre kubernetes y docker | microsoft azure. [Online]. Available: https://azure.microsoft.com/es-es/topic/kubernetes-vs-docker/

[28] Deploy on kubernetes. [Online]. Available: https://docs.docker.com/desktop/kubernetes/

[29] Cloud firestore. [Online]. Available: https://firebase.google.com/docs/firestore?hl=es

[30] ¿cómo es la facturación de cloud firestore? | firebase. [Online]. Available: https://firebase.google.com/docs/firestore/pricing?hl=es

[31] R. Python. Look ma, no for-loops: Array programming with NumPy – real python. [Online]. Available: https://realpython.com/numpy-array-programming/

[32] Python list comprehension (with examples). [Online]. Available: https://www.programiz.com/python-programming/list-comprehension