UNIVERSIDAD DE GRANADA

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Grado en Ingeniería Informática

TRABAJO DE FIN DE GRADO

Análisis de datos de Dispositivos de Salud Personal

Presentado por:
Pablo Huertas Arroyo

Curso académico 2024-2025

Análisis de datos de Dispositivos de Salud Personal

Pablo Huertas Arroyo

Pablo Huertas Arroyo *Análisis de datos de Dispositivos de Salud Personal* .
Trabajo de fin de Grado. Curso académico 2024-2025.

Responsable de tutorización

Roberto Morcillo Jiménez
Departamento de Ciencias de la Computación e Inteligencia Artificial

María José Martín Bautista
Departamento de Ciencias de la Computación e Inteligencia Artificial

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Universidad de Granada

Declaración de originalidad

D./Dña. Pablo Huertas Arroyo

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2024-2025, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 17 de noviembre de 2024

Fdo: Pablo Huertas Arroyo

Índice general

Summary

In the digital age, technological advances have radically transformed the way we interact with our environment, manage our time, and, crucially, monitor our well-being. Among the many technological innovations that have emerged in recent years, health watches, commonly known as smartwatches, have gained prominence as portable devices that go far beyond the simple functions of a conventional watch. These devices not only offer an elegant and versatile design but also integrate powerful sensors and advanced algorithms that allow users to monitor key aspects of their health in real time.

The impact of health watches on daily life is undeniable. These devices have revolutionized the way we relate to our own health, providing millions of users with a constant and accurate window into their physical and mental state. Thanks to their physical activity tracking features, heart rate monitoring, sleep quality measurement, and stress level detection, among others, health watches not only collect valuable data but also promote healthier lifestyle habits. This proactive approach allows users to make adjustments to their daily routines, contributing to the prevention of chronic diseases, improved overall well-being, and the adoption of more balanced lifestyles.

Moreover, the value of these devices extends beyond passive monitoring, as many health watches are equipped with alerts and features that can be crucial in emergency situations. From fall detection and warnings in case of irregular heart rates to the ability to request help immediately, these devices provide an additional layer of security that can make a significant difference in the lives of many people, especially those with pre-existing medical conditions.

In this context, health watches have ceased to be merely a technological trend or a fashion accessory. They have become essential tools in the field of preventive care and self-management of health. Their ability to collect, analyze, and present relevant data empowers users, providing them with the knowledge necessary to make more informed decisions about their well-being. Additionally, with the growing integration of artificial intelligence and machine learning, health watches are rapidly evolving into even more precise, personalized, and proactive monitoring systems.

It is within this framework of increasing relevance of health watches and their impact on personal wellness management that my Final Degree Project (TFG) is situated. My research has focused on the study of different current smartwatches and the analysis of data generated by these devices, with the aim of exploring the information they provide.

In conclusion, my work aims to contribute to the emerging field of digital health, offering a clearer view of the value of the data generated by health watches and proposing ways in which they can be applied to improve individual and collective health. In the following sections, I will detail the data collection process and the analysis carried out, as well as the subsequent creation of software to process this data.

Objective of the final degree project: *This Final Degree Project proposes to design and implement an advanced health monitoring system using wearables, aimed at various user groups. The research will focus on the creation and integration of SOA web services with wearable devices from different brands to collect health data and store it in an information system. This system will include a user interface to visualize the data captured by the wearables.*

This was my initial objective for the TFG, but it was later necessary to discard the possibility of collecting data directly from these devices due to their scarcity and the inability to extract such diverse data from different people, activities, and dates (such diverse data in summary). Therefore, it was decided to carry out the research, extraction, and cleaning of data from different datasets, which will be detailed in the TFG along with the process of how they were processed.

During the development of this work, several of the proposed objectives were achieved. Although it was not possible to collect data directly from wearable devices due to the aforementioned limitations, a detailed analysis of data extracted from various public datasets was carried out. This allowed for the implementation of an information system to store and analyze health data. Additionally, an intuitive user interface was developed to efficiently visualize this data.

The project also succeeded in designing and implementing web services based on a service-oriented architecture (SOA), which enabled modularity and scalability in data integration. Although the goal of obtaining real data from wearables was not achieved, the adaptation of the project to the available secondary data provided an enriching perspective on health data management and the implementation of web technologies for its analysis.

Introducción

En la era digital, los avances tecnológicos han transformado radicalmente la manera en que interactuamos con nuestro entorno, gestionamos nuestro tiempo y, de forma crucial, monitoreamos nuestro bienestar. Entre las múltiples innovaciones tecnológicas que han emergido en los últimos años, los relojes de salud, comúnmente conocidos como smartwatches o relojes inteligentes, han ganado protagonismo como dispositivos portátiles que van mucho más allá de las simples funciones de un reloj convencional. Estos dispositivos no solo ofrecen un diseño elegante y versátil, sino que integran potentes sensores y algoritmos avanzados que permiten a los usuarios monitorear aspectos clave de su salud en tiempo real.

El impacto de los relojes de salud en la vida cotidiana es indudable. Estos dispositivos han revolucionado la forma en que nos relacionamos con nuestra propia salud, proporcionando a millones de usuarios una ventana constante y precisa hacia su estado físico y mental. Gracias a sus funciones de seguimiento de la actividad física, el monitoreo del ritmo cardíaco, la medición de la calidad del sueño y la detección de niveles de estrés, entre otras, los relojes de salud no solo recopilan datos valiosos, sino que también fomentan hábitos de vida más saludables. Este enfoque proactivo permite a los usuarios hacer ajustes en sus rutinas diarias, lo que contribuye a la prevención de enfermedades crónicas, la mejora del bienestar general y la adopción de estilos de vida más equilibrados.

Además, el valor de estos dispositivos se extiende más allá del monitoreo pasivo, ya que muchos relojes de salud están equipados con alertas y funciones que pueden ser cruciales en situaciones de emergencia. Desde la detección de caídas y avisos en caso de irregularidades cardíacas, hasta la posibilidad de solicitar ayuda de forma inmediata, estos dispositivos proporcionan una capa adicional de seguridad que puede marcar una diferencia significativa en la vida de muchas personas, especialmente aquellas con afecciones médicas previas.

En este contexto, los relojes de salud han dejado de ser una simple tendencia tecnológica o un accesorio de moda. Se han convertido en herramientas esenciales dentro del campo del cuidado preventivo y la autogestión de la salud. Su capacidad para recopilar, analizar y presentar datos relevantes empodera a los usuarios, proporcionándoles el conocimiento necesario para tomar decisiones más informadas sobre su bienestar. Además, con la creciente integración de la inteligencia artificial y el aprendizaje automático, los relojes de salud están evolucionando rápidamente hacia sistemas de monitoreo aún más precisos, personalizados y proactivos.

Es en este marco de creciente relevancia de los relojes de salud y su impacto en la gestión personal del bienestar, donde se sitúa mi Trabajo de Fin de Grado (TFG). Mi investigación se ha centrado en el estudio de los distintos relojes inteligentes de la actualidad, el análisis de datos generados por estos dispositivos, con el objetivo de explorar cómo es la información proporcionada por estos.

En conclusión, mi trabajo busca aportar al campo emergente de la salud digital, ofreciendo una visión más clara sobre el valor de los datos generados por los relojes de salud y

proponiendo maneras en que estos pueden ser aplicados para mejorar la salud individual y colectiva. En las siguientes secciones, detallaré el proceso de recolección de datos y el análisis realizado, como la posterior creación de un software para tratar dichos datos.

Objetivo del trabajo de fin de grado: *Este Trabajo de Fin de Grado propone diseñar e implementar un sistema avanzado de control de salud mediante wereables, dirigido a diversos grupos de usuarios. La investigación se centrará en la creación e integración de servicios web SOA con dispositivos wearables de diferentes marcas para recopilar datos de salud y almacenarlos en un sistema de información. Dicho sistema llevará una interfaz de usuario para poder visualizar los datos captados por los wearables.*

Este era mi objetivo inicial del TFG, pero posteriormente se tuvo que descartar la posibilidad de recopilar los datos directamente de dichos dispositivos, debido a la escasez de estos y la imposibilidad de poder extraer tantos datos distintos de diferentes personas y en diferentes fechas, actividades, etc. (datos tan diversos en resumen). Por lo que, posteriormente, se decidió realizar la investigación, extracción y limpieza de datos de distintos datasets, que se detallarán en el TFG junto con el proceso de cómo fueron procesados.

En el desarrollo de este trabajo se lograron varios de los objetivos propuestos. Aunque no fue posible recopilar los datos directamente de los dispositivos wearables debido a las limitaciones mencionadas, se realizó un análisis detallado de datos extraídos de distintos datasets públicos. Esto permitió implementar un sistema de información para almacenar y analizar los datos de salud. Además, se desarrolló una interfaz de usuario intuitiva que permite visualizar estos datos de manera eficiente.

El proyecto también logró diseñar e implementar servicios web basados en una arquitectura orientada a servicios (SOA), lo que permitió modularidad y escalabilidad en la integración de los datos. Aunque no se alcanzó el objetivo de obtener datos reales de los wearables, la adaptación del proyecto a los datos secundarios disponibles ofreció una perspectiva enriquecedora sobre el manejo de datos de salud y la implementación de tecnologías web para su análisis.

Parte I.

Capítulo 1- Estado del arte

En la actualidad, los relojes inteligentes (*smartwatches*) han ganado relevancia como herramientas de monitoreo de la salud. No solo permiten hacer un seguimiento de la actividad física diaria, sino que también recopilan información detallada sobre diversos parámetros biométricos, como la frecuencia cardíaca, la calidad del sueño y los niveles de oxígeno en sangre, entre muchos otros. Entre los principales fabricantes de estos dispositivos se encuentran Apple, Garmin y Fitbit, cada uno ofreciendo soluciones con diferentes características y tipos de datos que permiten a los usuarios llevar un control exhaustivo de su estado físico.

Figura 1.1.: Smartwach de salud

Este capítulo tiene como objetivo realizar un análisis comparativo de los tipos de datos que manejan los principales modelos de relojes inteligentes, centrándonos en tres marcas representativas: Apple Watch, Garmin y Fitbit. Se describen los tipos de datos recogidos, su estructura, y las características únicas que cada dispositivo ofrece en términos de monitoreo de la salud.

Figura 1.2.: Smartwach de salud

1.1. Apple Watch

El Apple Watch, a través de su integración con HealthKit [7], recopila una amplia gama de datos relacionados con la salud del usuario. Los tipos de datos que gestiona HealthKit están organizados en categorías específicas que permiten un análisis detallado de la salud y el estado físico. A continuación, se describen los principales tipos de datos gestionados por el Apple Watch.

1.1.1. Tipos de Datos en Apple HealthKit

Apple HealthKit clasifica los datos en diferentes tipos según la naturaleza de la información que manejan:

Nombre tipo de dato	Descripción
HKCharacteristicType	Representa datos estáticos que normalmente no cambian con el tiempo. Estos datos describen características permanentes del usuario. *Ejemplos*: sexo biológico, tipo de sangre, fecha de nacimiento.
HKQuantityType	Identifica muestras que almacenan valores numéricos continuos o discretos. Estos datos se recopilan con frecuencia y proporcionan información cuantitativa sobre diversas métricas de salud. *Ejemplos*: conteo de pasos, distancia caminada o corrida, velocidad al correr, frecuencia cardíaca, índice de masa corporal (IMC).
HKCategoryType	Almacena información que pertenece a una categoría específica y se utiliza para representar eventos que se pueden clasificar dentro de un conjunto de posibles valores. *Ejemplos*: hora de pie de Apple, eventos de baja condición cardio, flujo menstrual, resultado de prueba de ovulación.
HKCorrelationType	Agrupa múltiples submuestras en una sola muestra para proporcionar una visión más compleja de los datos relacionados. *Ejemplos*: presión arterial (valores sistólicos y diastólicos), datos de alimentos (calorías, macronutrientes).
HKActivitySummaryType	Agrupa y resume la actividad física diaria, proporcionando un resumen de los movimientos y ejercicios que el usuario ha realizado a lo largo del día.
HKAudiogramSampleType	Identifica muestras que contienen datos de audiogramas, permitiendo el monitoreo de la capacidad auditiva del usuario.

Nombre tipo de dato	Descripción
HKElectrocardiogramType	Almacena datos relacionados con electrocardiogramas (ECG), permitiendo la detección de irregularidades en el ritmo cardíaco.
HKSeriesType	Indica datos almacenados en una serie de muestras, útil cuando se recopilan datos continuos o secuenciales a lo largo de un período de tiempo. *Ejemplos*: serie de latidos del corazón, rutas de entrenamiento durante la carrera o el ciclismo.
HKClinicalType	Identifica muestras que contienen datos de registros clínicos, permitiendo integrar información obtenida de instituciones médicas, como resultados de laboratorio o informes de salud.
HKWorkoutType	Almacena información detallada sobre un entrenamiento, como el tipo de ejercicio, duración, distancia recorrida, calorías quemadas y frecuencia cardíaca durante la actividad.
HKObjectType	Superclase abstracta para varios tipos específicos de datos almacenados en HealthKit. Esta clase se utiliza para definir los diferentes objetos que representan tipos de datos dentro de la plataforma HealthKit.
HKSampleType	Similar a HKObjectType, es una superclase abstracta para tipos específicos de muestras en HealthKit. Abarca varios subtipos de muestras que se pueden almacenar en la base de datos de HealthKit.
HKAttachment	Representa un archivo adjunto a una muestra en HealthKit. Por ejemplo, una imagen o documento asociado a una muestra de datos clínicos o un resultado de laboratorio.
Exposición de Audio Ambiental	Mide la exposición del usuario al sonido ambiental a lo largo del día. Este tipo de datos es único del Apple Watch y está diseñado para advertir sobre la exposición a niveles de ruido perjudiciales para la audición.
Variabilidad de la Frecuencia Cardíaca (SDNN)	Mide la desviación estándar de los intervalos entre latidos del corazón, importante para evaluar la salud cardiovascular y la respuesta del sistema nervioso autónomo.
VO2 Max	Mide el consumo máximo de oxígeno durante el ejercicio. Este es un indicador clave del rendimiento cardiovascular y la capacidad aeróbica del usuario.

1.1.2. Características del Apple Watch

El Apple Watch se distingue por ofrecer varias características exclusivas que no están disponibles en otros dispositivos de monitoreo de salud. Las características únicas del Apple

Watch incluyen:

- **Electrocardiograma (ECG)**: El Apple Watch permite realizar un electrocardiograma directamente desde la muñeca. Esta función permite detectar ritmos cardíacos irregulares, como la fibrilación auricular, y es especialmente útil para el monitoreo de la salud cardiovascular.
- **Exposición al Sonido Ambiental**: El Apple Watch monitorea la exposición al ruido ambiental, alertando al usuario cuando los niveles de sonido alcanzan un umbral que podría ser dañino para la audición a largo plazo. Esta función es única del Apple Watch y está diseñada para proteger la salud auditiva.
- **Alertas de Ritmo Cardíaco Irregular**: El dispositivo ofrece notificaciones cuando detecta un ritmo cardíaco anormal, ya sea demasiado alto, bajo o irregular. Esta característica es fundamental para identificar posibles problemas cardíacos antes de que se conviertan en emergencias.
- **Detección de Caídas**: El Apple Watch detecta automáticamente cuando el usuario sufre una caída brusca y ofrece la opción de contactar con los servicios de emergencia si el usuario no responde en un determinado tiempo. Esta función es particularmente útil para personas mayores o con problemas de movilidad.

1.2. Garmin

Garmin es una de las marcas más reconocidas en el ámbito del seguimiento de la actividad física y la salud, destacándose especialmente por su enfoque en los deportes y en la recopilación detallada de datos de rendimiento. Garmin ofrece varios tipos de datos de salud y actividad física a través de sus APIs [6] y SDKs [2], que permiten a los desarrolladores acceder a información valiosa para monitorear diferentes parámetros del cuerpo. A continuación, se describen los principales tipos de datos procesados por las plataformas de Garmin.

1.2.1. Tipos de Datos Procesados por Garmin Health API

La *Garmin Health API* [6] permite el acceso a una variedad de datos relacionados con la salud y la actividad física del usuario. Entre los tipos de datos más importantes se incluyen:

- **Pasos**: Registro del número de pasos dados por el usuario a lo largo del día. Este tipo de dato es fundamental para medir la actividad diaria y se usa como referencia para calcular otras métricas, como las calorías quemadas.
- **Minutos de Intensidad**: Tiempo total de actividades físicas moderadas o vigorosas realizadas durante el día. Garmin clasifica estas actividades de acuerdo con la intensidad para ofrecer una visión más precisa del esfuerzo físico.
- **Sueño**: Información detallada sobre las fases y la calidad del sueño. Garmin divide el sueño en varias etapas (ligero, profundo, REM) y proporciona una visión general del descanso nocturno del usuario.
- **Calorías**: Las calorías quemadas durante las actividades diarias y deportivas. Este dato es crucial para los usuarios que desean hacer un seguimiento de su gasto energético total.
- **Frecuencia Cardíaca**: Monitoreo continuo de la frecuencia cardíaca en reposo y durante la actividad física. Garmin utiliza estos datos para calcular otros parámetros, como las zonas de frecuencia cardíaca y el esfuerzo relativo durante el ejercicio.
- **Estrés**: Nivel de estrés del usuario, calculado a partir de la variabilidad de la frecuencia cardíaca. Esta métrica ayuda a los usuarios a controlar su bienestar emocional y a detectar períodos de estrés elevado.
- **Pulse Ox**: Nivel de saturación de oxígeno en la sangre. Esta métrica es especialmente útil para deportistas que realizan actividades en altitudes elevadas o para personas con condiciones respiratorias.
- **Body Battery**: Estimación de los niveles de energía del cuerpo. Esta función exclusiva de Garmin calcula una batería corporal que se recarga con el descanso y se descarga con la actividad física y el estrés, proporcionando una visión holística del nivel de fatiga del usuario.
- **Composición Corporal**: Datos relacionados con la composición corporal, incluyendo la grasa corporal, la masa muscular, el porcentaje de agua en el cuerpo, entre otros. Estos datos se utilizan para dar una visión más completa del estado físico del usuario.

- **Respiración**: Tasa de respiración durante el día y el sueño. La frecuencia respiratoria es importante para monitorizar el bienestar general y detectar posibles anomalías.
- **Presión Arterial**: Registro de la presión sanguínea del usuario. Aunque no todos los dispositivos Garmin miden la presión arterial, esta es una métrica que se puede monitorear en dispositivos compatibles y a través de integración con otros equipos médicos.

1.2.2. Tipos de Datos Procesados por Garmin Activity API

La *Garmin Activity API* proporciona acceso a datos relacionados con actividades deportivas específicas, permitiendo un análisis detallado de la información sobre el rendimiento físico del usuario. Los principales tipos de datos incluyen:

- **Detalles de Actividades**: Información detallada sobre actividades específicas como correr, ciclismo, natación, yoga, y entrenamiento de fuerza. Garmin ofrece un seguimiento exhaustivo de cada tipo de actividad, adaptando las métricas y los análisis según la disciplina.
- **Archivos de Actividad**: Garmin permite el acceso a archivos de actividad en formatos como FIT, GPX, y TCX, que contienen los datos completos de las actividades realizadas por el usuario. Estos archivos incluyen información sobre la ruta, velocidad, elevación, frecuencia cardíaca, y otros parámetros clave.

1.2.3. Garmin Health SDK

Además de sus APIs, Garmin ofrece un *Software Development Kit* (SDK) [3] que permite a los desarrolladores acceder directamente a los datos de los dispositivos Garmin. Las dos principales versiones del SDK son:

- **Standard SDK**: Permite un control total sobre los dispositivos Garmin, accediendo a los datos de salud y actividad directamente desde la aplicación móvil sin necesidad de integrar servicios web adicionales. Esto resulta útil para aplicaciones que necesitan acceder a los datos locales en tiempo real sin depender de la nube.
- **Companion SDK**: Permite la transmisión en tiempo real de datos de sensores desde los dispositivos Garmin. Con este SDK, es posible obtener datos en vivo como el conteo de pasos, la frecuencia cardíaca, los niveles de estrés, y otros parámetros mientras el usuario lleva el dispositivo puesto.

1.2.4. Características de Garmin

Garmin ofrece algunas características exclusivas que lo diferencian de otras plataformas de monitoreo de actividad física y salud. Entre ellas, destacan:

- **Body Battery**: Garmin proporciona una métrica única llamada *Body Battery*, que estima los niveles de energía del cuerpo a lo largo del día. Esta métrica se recarga con el descanso y se reduce con la actividad física y el estrés, permitiendo a los usuarios optimizar su rendimiento y descanso basándose en esta información.

- **Actividades Específicas**: Garmin ofrece un seguimiento detallado para una amplia variedad de actividades deportivas específicas, como correr, ciclismo, natación, yoga y entrenamiento de fuerza. Esta característica es especialmente útil para usuarios que practican diferentes deportes y desean obtener análisis precisos y personalizados para cada uno.
- **Presión Arterial**: Algunos dispositivos Garmin permiten la monitorización de la presión arterial, una función clave para usuarios que requieren un control constante de este parámetro. La posibilidad de rastrear la presión sanguínea directamente desde el dispositivo es un valor añadido significativo en el monitoreo de la salud.

1.3. Fitbit

Fitbit es una de las plataformas más populares para el monitoreo de la salud y la actividad física, destacándose por ofrecer una variedad de datos a través de su API web [5]. Estos datos proporcionan una visión detallada del estado físico del usuario y permiten realizar un seguimiento exhaustivo de múltiples métricas de salud. A continuación, se ofrece un análisis de los principales tipos de datos procesados por Fitbit y las variables asociadas a cada uno de ellos.

1.3.1. Análisis de los Tipos de Datos que Procesa Fitbit

Fitbit gestiona una amplia variedad de tipos de datos relacionados con la actividad física, la salud y el bienestar general. Estos datos se organizan en diferentes categorías, cada una de las cuales incluye variables específicas que se pueden acceder a través de diversos endpoints de la API [1].

1.3.1.1. Datos de Actividad

Fitbit recopila datos detallados relacionados con la actividad física diaria del usuario, como los pasos, la distancia recorrida y las calorías quemadas.

- **Variables**: pasos (`steps`), distancia (`distance`), elevación (`elevation`), pisos subidos (`floors`), calorías quemadas (`calories burned`), minutos activos (`active minutes`), detalles del ejercicio (`exercise details`) y estadísticas de por vida (`lifetime stats`).
- **Endpoints**: `Get Activity Goals`, `Get Activity Log List`, `Get Daily Activity Summary`, `Get Lifetime Stats`.

1.3.1.2. Serie Temporal de Actividad

Fitbit también ofrece datos intradiarios, permitiendo a los usuarios realizar un seguimiento en tiempo real de su actividad física a lo largo del día.

- **Variables**: datos intradiarios para pasos, distancia, pisos subidos y calorías quemadas.
- **Endpoints**: `Get Activity Time Series`.

1.3.1.3. Mediciones Corporales

Los datos relacionados con las mediciones corporales son importantes para el seguimiento de la composición física y el progreso en términos de pérdida de peso o ganancia muscular.

- **Variables**: peso (`weight`), índice de masa corporal (IMC) (`BMI`), porcentaje de grasa corporal (`body fat percentage`).
- **Endpoints**: `Get Body Weight`, `Get Body Fat Percentage`.

1.3.1.4. Frecuencia Respiratoria

Fitbit ofrece datos sobre la tasa de respiración, lo cual es útil para el seguimiento de la salud respiratoria y la capacidad aeróbica del usuario.

- **Variables**: tasa de respiración (`breathing rate`), medida en respiraciones por minuto.
- **Endpoints**: `Get Breathing Rate`.

1.3.1.5. Puntuación de Fitness Cardíaca (VO2 Max)

La estimación de VO2 Max es una métrica clave del rendimiento cardiovascular, y Fitbit ofrece una puntuación de fitness cardíaca basada en este valor.

- **Variables**: estimación de VO2 Max.
- **Endpoints**: `Get Cardio Fitness Score`.

1.3.1.6. Frecuencia Cardíaca

Fitbit monitoriza de manera continua la frecuencia cardíaca del usuario, tanto en reposo como durante el ejercicio, y proporciona información sobre las zonas de frecuencia cardíaca.

- **Variables**: frecuencia cardíaca en reposo (`resting heart rate`), zonas de frecuencia cardíaca (`heart rate zones`), frecuencia cardíaca máxima (`peak heart rate`).
- **Endpoints**: `Get Heart Rate`, `Get Heart Rate Time Series`.

1.3.1.7. Variabilidad de la Frecuencia Cardíaca (HRV)

La variabilidad de la frecuencia cardíaca (HRV) mide el tiempo entre latidos del corazón, un indicador clave de la salud cardiovascular y la respuesta del sistema nervioso autónomo.

- **Variables**: tiempo entre latidos del corazón.
- **Endpoints**: `Get HRV`.

1.3.1.8. Datos de Sueño

Fitbit recopila información detallada sobre las diferentes etapas del sueño y la calidad general del descanso del usuario.

- **Variables**: etapas del sueño (`sleep stages`) (ligero, profundo, REM), duración del sueño (`sleep duration`), eficiencia del sueño (`sleep efficiency`).
- **Endpoints**: `Get Sleep Logs`, `Get Sleep Goals`.

1.3.1.9. SpO2 (Niveles de Oxígeno en Sangre)

El nivel de saturación de oxígeno en la sangre es una métrica crucial para evaluar la salud respiratoria. Fitbit ofrece seguimiento continuo de este parámetro en dispositivos compatibles.

- **Variables**: niveles de saturación de oxígeno (`oxygen saturation levels`).
- **Endpoints**: `Get SpO2 Data`.

1.3.1.10. Temperatura

Fitbit también permite monitorear la variación de la temperatura de la piel, lo que puede ser útil para detectar cambios fisiológicos importantes.

- **Variables**: variación de la temperatura de la piel (`skin temperature variation`).
- **Endpoints**: `Get Skin Temperature Data`.

1.3.1.11. Datos de Nutrición

Fitbit facilita el seguimiento de los hábitos alimenticios y de hidratación del usuario, permitiendo un control detallado de la ingesta calórica y el balance energético.

- **Variables**: registros de alimentos (`food logs`), ingesta de agua (`water intake`), calorías consumidas frente a calorías quemadas (`calories in vs. calories out`).
- **Endpoints**: `Get Food Logs`, `Get Water Logs`, `Get Nutrition Goals`.

1.3.2. Características de Fitbit

Fitbit se destaca por ofrecer varias características exclusivas que no están presentes en otras plataformas de monitoreo de salud. Entre estas características se incluyen:

- **Tasa de Respiración**: Fitbit monitoriza la tasa de respiración del usuario, medida en respiraciones por minuto. Esta métrica es útil para evaluar la salud respiratoria y la capacidad aeróbica, y es especialmente valiosa durante el sueño para detectar posibles problemas respiratorios.
- **Registro de Alimentos**: Fitbit ofrece la posibilidad de realizar un seguimiento detallado de la ingesta de alimentos. Los usuarios pueden registrar sus comidas, controlar las calorías consumidas y hacer un seguimiento de los macronutrientes, lo que facilita el control de la nutrición y el balance energético diario.
- **Ingesta de Agua**: Fitbit permite a los usuarios llevar un registro de su consumo de agua a lo largo del día, ayudando a mantener una adecuada hidratación. Esta función es particularmente útil para quienes siguen dietas o programas de salud enfocados en la hidratación.
- **Temperatura de la Piel**: Fitbit monitoriza la variación de la temperatura de la piel del usuario, proporcionando datos valiosos sobre cambios fisiológicos que pueden estar relacionados con la salud o el ciclo menstrual. Este seguimiento es importante para detectar anomalías o cambios significativos en el estado del cuerpo.
- **Niveles de Glucosa**: Fitbit permite a los usuarios realizar un seguimiento de los niveles de glucosa en sangre (en dispositivos compatibles o mediante integración con otros sistemas), lo cual es fundamental para personas con diabetes o aquellas que necesiten monitorizar su nivel de azúcar en sangre regularmente.

Parte II.

Capítulo 2 - Objetivos y Motivaciones

2.1. Objetivos

- Desarrollar e implementar un sistema avanzado de monitoreo de salud mediante dispositivos portátiles (wearables), enfocado en diferentes grupos de usuarios.
- Diseñar e implementar servicios web basados en una arquitectura orientada a servicios (SOA) para la recopilación de datos de salud desde wearables de diferentes marcas.
- Almacenar los datos de salud en un sistema de información, garantizando su disponibilidad para el análisis.
- Proporcionar una interfaz de usuario intuitiva que permita visualizar los datos captados, facilitando la comprensión de la información de salud.

Dado que no fue posible obtener datos directamente de los dispositivos wearables, el enfoque del proyecto cambió hacia la investigación, extracción y limpieza de datos a partir de diferentes conjuntos de datos (datasets) disponibles. Estos datos serán detallados y procesados en el desarrollo del TFG.

2.2. Motivaciones

Las principales motivaciones para llevar a cabo este proyecto incluyen un profundo interés en el campo de la salud digital, que es un área en pleno crecimiento. La capacidad de los wearables para proporcionar datos de salud en tiempo real presenta un enorme potencial para mejorar la calidad de vida. Además, el proyecto pretende contribuir al bienestar personal y colectivo, desarrollando herramientas que permitan a los usuarios monitorear y analizar su estado de salud, fomentando un enfoque preventivo que pueda tener un impacto significativo en la prevención de enfermedades y la promoción de estilos de vida saludables.

Otra motivación importante es la aplicación de tecnologías emergentes. La integración de dispositivos wearables con servicios web y el uso de técnicas de análisis de datos permite explorar el potencial de la tecnología para la gestión de la salud de una manera más eficiente y personalizada. Finalmente, la motivación de innovar en el área del cuidado preventivo es clave, desarrollando soluciones que empoderen a las personas para gestionar mejor su salud y bienestar.

Parte III.

Capítulo 3 - Planificación y presupuesto

3.1. Sprints

Los sprints representan iteraciones de desarrollo en las que se aborda una parte específica del proyecto, de manera organizada y eficiente. Cada sprint tiene una duración definida y un conjunto de historias técnicas seleccionadas, de acuerdo con la prioridad y estimación de esfuerzo. El objetivo de cada sprint es asegurar que el equipo pueda enfocarse en entregar funcionalidades incrementales, asegurando un progreso constante hacia el objetivo final.

A continuación, se describe la planificación de los sprints que abarcan las historias del proyecto, con el propósito de garantizar una distribución equilibrada del trabajo y la entrega de valor continuo.

3.1.1. Cronograma

Figura 3.3.: Diagrama de Gantt

3.1.2. Iteraciones

Sprint	Historias Incluidas	Estimación Total	Número de Tareas
Sprint 1	HT. 1, HT. 2	5	2
Sprint 2	HT. 3, HT. 4	5	2
Sprint 3	HT. 5, HT. 6	8	2
Sprint 4	HU. 1, HU. 3	6	2
Sprint 5	HU. 4, HU. 5	8	2
Sprint 6	HU. 6	4	1
Sprint 7	HU. 2, HU. 7.	8	2

Tabla 3.1.: Planificación de Sprints

Sprint	Detalles
Sprint 1	▪ **HT. 1 - Configuración del Entorno de Desarrollo:** Prioridad 1, Estimación 2. ▪ **HT. 2 - Implementación de la Base de Datos:** Prioridad 2, Estimación 3.
Sprint 2	▪ **HT. 3 - Despliegue de la API con FastAPI:** Prioridad 1, Estimación 3. ▪ **HT. 4 - Integración de Middleware CORS:** Prioridad 2, Estimación 2.
Sprint 3	▪ **HT. 5 - Pruebas Unitarias:** Prioridad 1, Estimación 4. ▪ **HT. 6 - Optimización de la Base de Datos:** Prioridad 2, Estimación 4.
Sprint 4	▪ **HU. 1 - Acceso a Datos Filtrados:** Prioridad 1, Estimación 3. ▪ **HU. 3 - Visualización de Datos de Salud:** Prioridad 3, Estimación 2.
Sprint 5	▪ **HU. 4 - Visualización de Gráficas:** Prioridad 1, Estimación 4. ▪ **HU. 5 - Comparación entre Dispositivos:** Prioridad 3, Estimación 4.
Sprint 6	▪ **HU. 6 - Exportación de datos filtrados en formato CSV:** Prioridad 1, Estimación 4.
Sprint 7	▪ **HU. 2 - Integración con la API :** Prioridad 2 Estimación 3. ▪ **HU. 7 - Autenticación y Seguridad:** Prioridad 4 Estimación 5.

Tabla 3.2.: Detalles de los Sprints

3.2. Presupuesto

Para calcular el presupuesto del proyecto, se estimarán las horas de trabajo dedicadas al desarrollo y análisis. El precio por hora se determinará según la tarifa estándar de un desarrollador de software en España. Además, se incluirán los costos asociados al uso de un ordenador personal y la conexión a Internet. Se dará prioridad al uso de tecnologías gratuitas y accesibles para reducir los gastos en licencias y software.

Para elaborar el presupuesto del proyecto se han tenido en cuenta los siguientes aspectos:

3.2.1. Costo del Personal

Se ha calculado el tiempo necesario para completar las tareas de desarrollo y análisis, utilizando la tarifa estándar de un desarrollador de software en España. Considerando el costo por hora de un ingeniero informático en España, que es de aproximadamente 15 €/hora, la duración estimada del proyecto de 14 semanas, y una dedicación de 10 horas semanales, el coste del personal sería:

$$\text{Costo del personal} = \text{Horas semanales} \times \text{Semanas} \times \text{Costo por hora}$$

$$\text{Costo del personal} = 10 \text{ horas/semana} \times 15 \text{ semanas} \times 15 \text{ €/hora} = 2250€$$

3.2.2. Costo del Equipo

Este costo incluye el uso de un ordenador personal y su prorrateo durante el proyecto. Asumiendo un coste de 1200 € para un ordenador y considerando un ciclo de vida de 8 años, el costo prorrateado sería de aproximadamente 12,5 € por mes. Además, se incluye el costo de la conexión a Internet, que se estima en 30 € al mes durante la duración del proyecto de 4 meses:

$$\text{Costo de la conexión a Internet} = 4 \text{ meses} \times 30 \text{ €/mes} = 120€$$

3.2.3. Costo del Software

Se ha dado prioridad al uso de tecnologías gratuitas y accesibles para minimizar este costo, por lo que se considera un coste cero para las licencias de software.

3.2.4. Costo Total

El costo total del proyecto se obtiene sumando los costos del personal, el equipo y los recursos de software:

$$\text{Costo total} = \text{Costo del personal} + \text{Costo del equipo} + \text{Costo del software}$$

$$\text{Costo total} = 2250 + 1200 + 120 = 3570€$$

3.3. Arquitectura de la aplicación

Se ha diseñado la aplicación siguiente el patrón de arquitectura de Modelo Vista Controlador (MVC). Este patrón de diseño separa las responsabilidades de la aplicación en estas tres

componentes principales. Esto facilita el desarrollo, escalabilidad y mantenimiento de la misma.

3.3.1. Diagrama de arquitectura de la aplicación

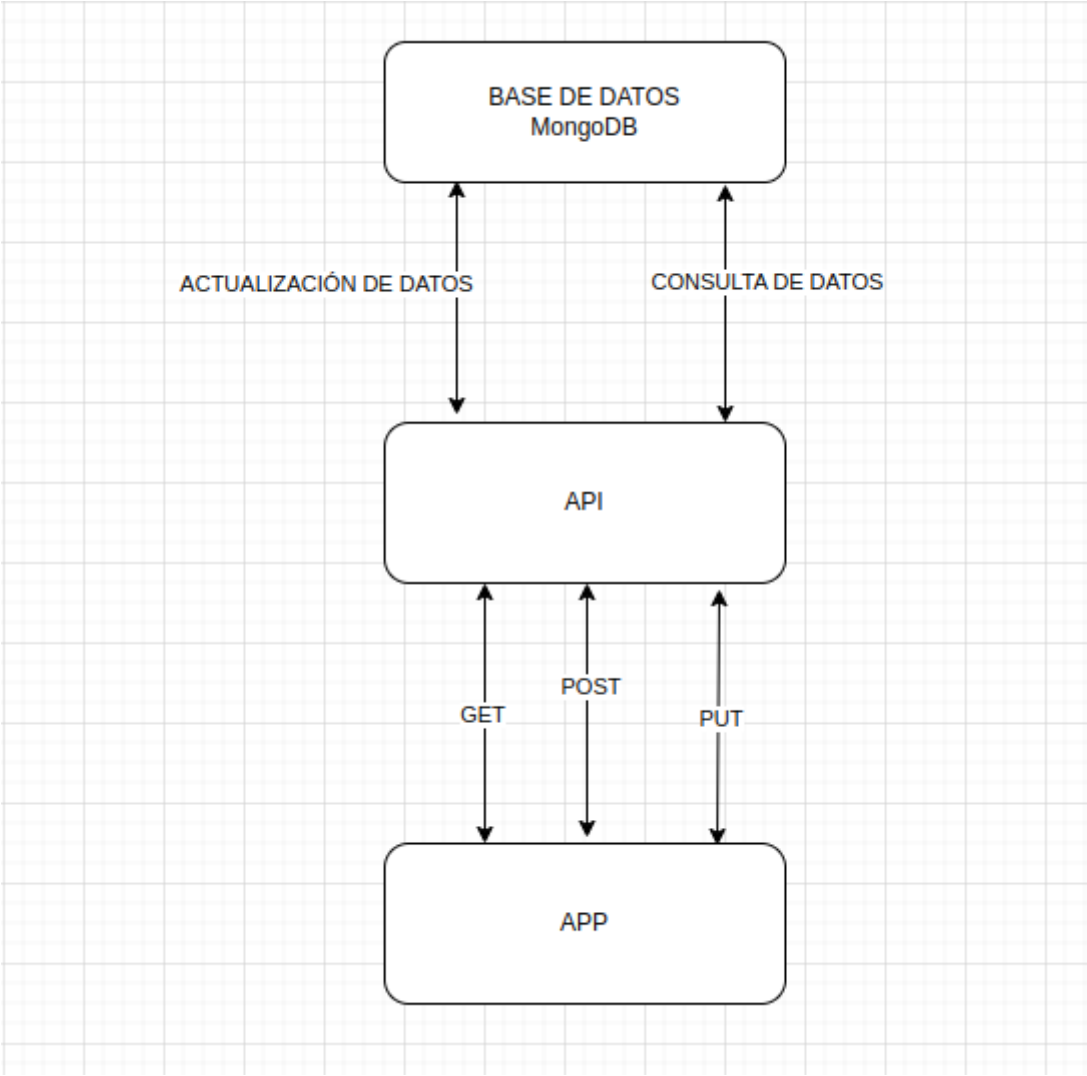


Figura 3.4.: Diagrama de arquitectura de aplicación

3.3.2. Modelo Vista Controlador (MVC)

El patrón **MVC** es una **arquitectura de software** que divide una **aplicación** en tres partes interrelacionadas:

- **Modelo**: Representa la lógica de negocio y los datos de la aplicación. El modelo se

encarga de gestionar la información de **usuarios**, recursos disponibles, etc.

- **Vista**: Es la interfaz de usuario, es decir, lo que el usuario ve en pantalla. En una **aplicación web**, la vista suele estar construida con **HTML, CSS y JavaScript**. Su función es presentar la información del modelo de manera clara y atractiva al usuario.
- **Controlador**: Actúa como intermediario entre el modelo y la vista. Gestiona las solicitudes del usuario, actualiza el modelo en consecuencia y selecciona la vista adecuada para mostrar al usuario. En este contexto, el controlador gestiona las acciones.

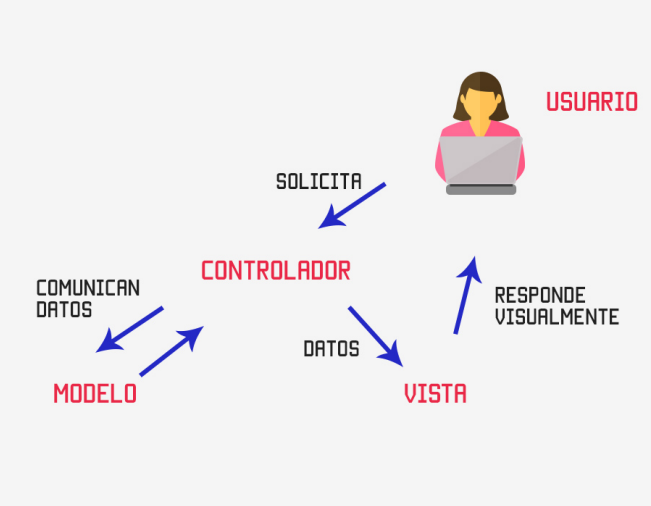


Figura 3.5.: Modelo-Vista-Controlador

3.3.3. Implementación del MVC

Veamos cómo se implementa el patrón MVC en la aplicación:

- **Modelo**:
  - **Base de Datos**: Se utiliza MongoDB, una base de datos NoSQL, para almacenar los datos de los usuarios, dispositivos y estadísticas de salud. MongoDB permite una estructura flexible de los datos, lo cual es ideal para la variedad de información recolectada por los dispositivos.
  - **ORM**: Aunque MongoDB no es una base de datos relacional, se emplean librerías como Pydantic para interactuar con la base de datos de una manera más estructurada y orientada a objetos. Esto permite definir esquemas para validar los datos antes de almacenarlos.
- **Vista**:
  - **Framework Frontend**: Para construir la interfaz de usuario, se utiliza React, que permite una experiencia de usuario dinámica y modular. React facilita el desarrollo de una UI fluida, permitiendo a los usuarios visualizar los datos en tiempo real.

    - **Componentes**: La interfaz se divide en componentes reutilizables. Estos componentes hacen que el desarrollo sea más organizado y permiten la reutilización de código a lo largo de la aplicación.
    - **Templates**: Se utilizan plantillas JSX en React para definir la estructura de cada componente, permitiendo una integración directa con la lógica del frontend, facilitando así la actualización y visualización de datos de manera dinámica.
- **Controlador**:
    - **Framework Backend**: El controlador está implementado usando FastAPI. FastAPI maneja las solicitudes HTTP entrantes, realiza las validaciones necesarias y devuelve respuestas adecuadas.
    - **Rutas**: Se definen rutas en FastAPI que mapean las URL a funciones específicas del controlador. Por ejemplo, rutas para obtener datos de los dispositivos o agregar nuevos registros. Estas rutas gestionan las solicitudes de los usuarios y coordinan la interacción con el modelo y la vista.
    - **Lógica de Negocio**: Las funciones del controlador se encargan de validar los datos de las solicitudes, interactuar con la base de datos (a través del modelo) y devolver los datos a la vista de manera estructurada. Además, se emplean validaciones automáticas proporcionadas por Pydantic para garantizar la integridad de los datos recibidos.

Los beneficios de usar el patrón **MVC** son numerosos, entre ellos podemos destacar que separa las responsabilidades, facilitando así el desarrollo, mantenimiento y pruebas de la aplicación. Este patrón permite crear componentes reutilizables y un código más modular, lo cual contribuye a agregar nuevas funcionalidades de manera más sencilla, además de hacer más eficiente la localización y corrección de errores.

La elección del patrón **MVC** para esta aplicación web ha sido fundamental para garantizar una arquitectura sólida, escalable y mantenible. Al separar las diferentes responsabilidades de la aplicación, se ha logrado un código más limpio y organizado, lo que facilita su desarrollo y evolución a largo plazo.

3.4. Descripción de la metodología

En esta sección se presenta la metodología **SCRUM**, que ha sido seleccionada como marco de trabajo para la realización de este Trabajo de Fin de Grado (TFG). Se explicará en detalle qué es SCRUM, sus principales características, roles, artefactos y eventos, así como su aplicabilidad en el contexto de este proyecto.

SCRUM es un marco de trabajo para la gestión de proyectos basado en metodologías ágiles, que ayuda a los equipos a estructurar y gestionar el trabajo mediante un conjunto de valores, principios y prácticas. Las metodologías ágiles consisten en un conjunto de técnicas aplicadas en ciclos de trabajo cortos, que buscan aumentar la eficiencia en el proceso de entrega de un proyecto. Con cada ciclo completado se generan avances, evitando la necesidad de esperar hasta la finalización del proyecto completo para tener entregables funcionales.

Es importante aclarar que **SCRUM** es un marco específico dentro de las metodologías ágiles, no sinónimo de éstas. Las metodologías ágiles constituyen un conjunto de principios, mientras que SCRUM es una metodología específica que se centra en la obtención de resultados a través de roles, artefactos y eventos claramente definidos.

Las principales características de SCRUM que han motivado su elección para este proyecto son:

- **Iterativo e Incremental**: SCRUM se basa en ciclos de trabajo denominados *sprints*, que tienen una duración fija y entregan incrementos funcionales al final de cada iteración. Un sprint es un período breve de tiempo en el cual el equipo trabaja para completar una serie de tareas previamente establecidas.
- **Flexibilidad y Adaptabilidad**: La metodología SCRUM permite responder rápidamente a los cambios en los requisitos del proyecto y a las necesidades del cliente, lo cual la hace altamente adaptable.
- **Transparencia y Colaboración**: SCRUM fomenta la transparencia y la colaboración entre todos los miembros del equipo, promoviendo una comunicación continua y una toma de decisiones compartida.

Dentro de SCRUM se encuentran tres roles fundamentales que colaboran estrechamente en la ejecución del proyecto:

- **Product Owner**: Es el encargado de maximizar el valor del producto mediante la gestión del *Product Backlog*. Representa la voz del cliente dentro del equipo, tomando decisiones sobre el contenido y las prioridades del producto.
- **Scrum Master**: Actúa como un líder de servicio, ayudando al equipo a autogestionarse y alcanzar su máximo potencial. Su rol es garantizar que se sigan los principios y prácticas de SCRUM, eliminando impedimentos y facilitando el proceso de trabajo.
- **Equipo de Desarrollo**: Es un grupo de profesionales que trabaja para entregar incrementos funcionales al final de cada sprint. El equipo es auto-organizado y autogestionado, manteniendo altos estándares de calidad.

SCRUM también hace uso de tres artefactos clave para gestionar el trabajo y la información del proyecto:

- **Product Backlog**: Es una lista priorizada de trabajo para el equipo de desarrollo, que refleja la hoja de ruta del proyecto. El *Product Owner* es responsable de mantenerlo actualizado según los cambios del mercado y las necesidades del cliente.
- **Sprint Backlog**: Contiene las tareas específicas que el equipo se compromete a completar durante el sprint. Se deriva de los elementos seleccionados del *Product Backlog* durante la planificación del sprint.
- **Incremento**: Es el resultado de cada sprint, un producto que cumple con los criterios de aceptación y aporta valor al cliente. Los incrementos se acumulan y contribuyen a la evolución continua del producto.

En este proyecto, asumo los roles de **Product Owner**, **Scrum Master** y **Equipo de Desarrollo**, llevando a cabo las responsabilidades asociadas a cada uno. La planificación del proyecto se gestionará mediante la elaboración de un **Product Backlog** con todas las tareas y funcionalidades, priorizando en función del valor aportado al producto y actualizándolo a medida que evolucionen los requisitos.

Los sprints tendrán una duración breve, de aproximadamente dos semanas, y se enfocarán en la entrega de incrementos funcionales. Cada sprint concluirá con una revisión junto al tutor, quien desempeñará el rol de cliente para proporcionar una evaluación objetiva y retroalimentación sobre el avance del proyecto, permitiendo ajustes y mejoras en los siguientes sprints.

3.5. Tarjetas de Usuario para el Proyecto

Las tarjetas de usuario son una herramienta ágil que se utiliza para describir las necesidades, expectativas y comportamientos de los usuarios de la aplicación. Son una forma concisa y visual de comunicar la perspectiva del usuario al equipo de desarrollo. En mi aplicación encontramos dos tipos de usuarios: los administradores que son los responsables de la gestión de la aplicación y usuarios finales.

3.5.1. Personaje: Administrador

El administrador es responsable de la supervisión general y la gestión de la aplicación. Sus tareas incluyen acceder a todos los datos almacenados, aplicar filtros específicos para supervisar la calidad de los datos y gestionar la infraestructura técnica de la aplicación. El administrador tiene un conocimiento técnico avanzado y es el encargado de asegurar que la aplicación funcione de manera eficiente.

3.5.2. Personaje: Usuario final

El usuario final es quien interactúa directamente con la aplicación para acceder a los datos de salud. El usuario desea ver gráficas de actividad, comparar datos entre dispositivos, y filtrar los datos para obtener información específica. El objetivo principal del usuario final es utilizar la aplicación para su estudio.

3.6. Prioridad

Para definir la prioridad de cada historia de usuario, se ha desarrollado una rúbrica que toma en cuenta diversos factores, como la importancia para alcanzar los objetivos del proyecto. La prioridad se determina en función de su relevancia y urgencia con respecto a los objetivos del proyecto y las necesidades del usuario. Esta clasificación nos facilita la planificación y ejecución de los sprints. La prioridad se ha clasificado en cuatro niveles, que se describen a continuación en la tabla:

ID	Prioridad	Descripción
1	Alta	Historias de usuario que son críticas para el funcionamiento básico del sistema o que tienen un gran impacto en la satisfacción del usuario.
2	Media	Historias de usuario importantes, pero no indispensables para el funcionamiento básico del sistema.
3	Baja	Historias de usuario que tienen un impacto limitado en la funcionalidad general del sistema o que pueden ser aplazadas sin afectar significativamente el desarrollo del proyecto.
4	Muy Baja	Historias de usuario con un impacto mínimo en la funcionalidad del sistema o que pueden posponerse indefinidamente sin afectar el éxito del proyecto.

Tabla 3.3.: Rúbrica de prioridad

3.7. Estimación tiempo

Para estimar el esfuerzo requerido para cada historia de usuario, se ha creado una rúbrica que considera la dificultad y la dedicación necesaria para completar cada tarea. Esta información es esencial para planificar y priorizar las historias de usuario durante los sprints, garantizando una distribución equitativa del trabajo. La rúbrica se presenta a continuación en la Tabla 3.4:

ID	Esfuerzo	Tiempo Estimado
1	Bajo	Unos minutos
2	Leve	Horas
3	Moderado	Un día
4	Alto	Semanas
5	Intenso	Meses

Tabla 3.4.: Rúbrica para la estimación del esfuerzo y tiempo

3.8. Historias de Usuario

La planificación del proyecto seguirá la metodología Scrum, un enfoque ágil para la gestión de proyectos que se enfoca en entregas iterativas e incrementales. En Scrum, el trabajo se divide en iteraciones llamadas "sprints", que suelen durar entre 2 y 4 semanas, permitiendo una adaptación rápida a posibles imprevistos y proporcionando mayor flexibilidad, ya que se pueden organizar las tareas según su complejidad y tiempo necesario.

Por ello, se desarrollarán historias de usuario, que son descripciones cortas y sencillas de las funcionalidades que debe tener el sistema, desde la perspectiva del usuario final. Estas historias se agrupan en épicas, las cuales ofrecen una visión general de las funcionalidades a implementar y facilitan la organización y priorización del trabajo en el backlog. En resumen, son los objetivos que buscamos alcanzar al finalizar el proyecto.

A continuación se muestran las historias de usuario creadas para el proyecto:

3.8.1. Historias de Usuario para el Administrador

3.8.1.1. HU. 1 - Acceso a Datos Filtrados

ID	HU. 1
Nombre	**Acceso a Datos Filtrados**
Descripción	*Como administrador, quiero tener la capacidad de acceder a todos los datos almacenados en la base de datos con filtros específicos, para supervisar y mantener la calidad de los datos.*
Criterios de Aceptación	▪ Filtros específicos para buscar por dispositivo, edad y actividad. ▪ Resultados precisos excluyendo campos innecesarios como identificadores únicos.
Estimación	3
Prioridad	1

Tabla 3.5.: Historia de Usuario HU. 1

3.8.1.2. HU. 2 - Integración con la API (Posible futura mejora)

ID	HU. 2
Nombre	**Integración con la API (Posible mejora futura)**
Descripción	*Como administrador, quiero acceder a la API para obtener datos de los dispositivos en tiempo real, para mostrar visualizaciones en la aplicación y supervisar el rendimiento.*
Criterios de Aceptación	▪ Documentación clara de los endpoints. ▪ Respuestas rápidas y eficientes de la API.
Estimación	3
Prioridad	2

Tabla 3.6.: Historia de Usuario HU. 2

3.8.2. Historias de Usuario para el Usuario Final

3.8.2.1. HU. 3 - Visualización de Datos de Salud

ID	HU. 3
Nombre	**Visualización de Datos de Salud**
Descripción	*Como usuario final, quiero visualizar un resumen de los datos de actividad física y salud.*
Criterios de Aceptación	▪ Mostrar total de pasos y calorías quemadas por cada tipo de actividad de cada tipo de reloj.
Estimación	3
Prioridad	2

Tabla 3.7.: Historia de Usuario HU. 3

3.8.2.2. HU. 4 - Visualización de Gráficas

ID	HU. 4
Nombre	**Visualización de Gráficas**
Descripción	*Como usuario, quiero ver gráficas de la frecuencia cardíaca a lo largo de las medidas, para evaluar el impacto de los entrenamientos en la salud.*
Criterios de Aceptación	▪ Gráficos para mostrar las calorías gastadas por tipo de ejercicio. ▪ Posibilidad de seleccionar diferentes tipos de ejercicios. ▪ Mostrar los gráficos los datos de ambos relojes, de distinto color para poder distinguirlos.
Estimación	4
Prioridad	1

Tabla 3.8.: Historia de Usuario HU. 4

3.8.2.3. HU. 5 - Comparación entre Dispositivos

ID	HU. 5
Nombre	**Comparación entre Dispositivos**
Descripción	*Quiero comparar los datos recogidos por diferentes dispositivos, para entender las diferencias en las mediciones.*
Criterios de Aceptación	▪ Comparación de datos visualizada lado a lado.
Estimación	4
Prioridad	3

Tabla 3.9.: Historia de Usuario HU. 5

3.8.2.4. HU. 6 - Filtrado de Datos de Salud

ID	HU. 6
Nombre	**Exportación de datos filtrados en formato CSV**
Descripción	*Como usuario final, quiero exportar los datos de actividad física filtrados para poder tener acceso a ellos para diversas finalidades.*
Criterios de Aceptación	▪ Posibilidad de exportar los datos filtrados.
Estimación	4
Prioridad	1

Tabla 3.10.: Historia de Usuario HU. 6

3.8.2.5. HU. 7 - Autenticación y Seguridad (Posible Mejora Futura)

ID	HU. 7
Nombre	**Autenticación y Seguridad (Posible Mejora Futura)**
Descripción	*Como usuario registrado, quiero tener la seguridad de que los datos personales de salud están protegidos mediante autenticación segura.*
Criterios de Aceptación	▪ Inicio de sesión para acceso a datos personales privados.
Estimación	5
Prioridad	4

Tabla 3.11.: Historia de Usuario HU. 7

3.9. Historias Técnicas

Las historias técnicas son fundamentales para garantizar que el sistema sea robusto, escalable y fácil de mantener. Estas historias se centran en los aspectos técnicos del sistema que no son visibles para los usuarios finales, pero son cruciales para el funcionamiento de la aplicación.

A continuación se presentan las historias técnicas para el proyecto:

ID	**HT. 1**
Nombre	**Configuración del Entorno de Desarrollo**
Descripción	*Como desarrollador, quiero configurar el entorno de desarrollo con todas las herramientas necesarias, para asegurar que el equipo pueda comenzar a trabajar rápidamente.*
Criterios de Aceptación	▪ Instalación de Python, React, npm, fastapi, uvicorn, IDEs, dependencias y librerías necesarias. ▪ Configuración de Git para el control de versiones.
Estimación	2
Prioridad	1

Tabla 3.12.: Historia Técnica HT. 1

ID	**HT. 2**
Nombre	**Implementación de la Base de Datos**
Descripción	*Como desarrollador, quiero implementar la base de datos MongoDB, para almacenar de manera eficiente los datos de los usuarios y dispositivos.*
Criterios de Aceptación	▪ Configuración de la conexión con MongoDB. ▪ Creación de colecciones y estructura de datos necesaria.
Estimación	3
Prioridad	2

Tabla 3.13.: Historia Técnica HT. 2

ID	HT. 3
Nombre	**Despliegue de la API con FastAPI**
Descripción	*Como desarrollador, quiero desplegar la API utilizando FastAPI, para proporcionar acceso a los datos de los dispositivos de manera eficiente.*
Criterios de Aceptación	▪ Endpoints disponibles para consultar y filtrar datos. ▪ Documentación automática generada con Swagger.
Estimación	3
Prioridad	1

Tabla 3.14.: Historia Técnica HT. 3

ID	HT. 4
Nombre	**Integración de Middleware CORS**
Descripción	*Como desarrollador, quiero integrar un middleware CORS en la API, para permitir que diferentes clientes puedan acceder a los datos de forma segura.*
Criterios de Aceptación	▪ Configuración correcta del middleware CORS para permitir solicitudes desde orígenes específicos. ▪ Pruebas que confirmen el acceso seguro desde distintos clientes.
Estimación	2
Prioridad	2

Tabla 3.15.: Historia Técnica HT. 4

ID	HT. 5
Nombre	**Pruebas Unitarias**
Descripción	*Como desarrollador, quiero implementar pruebas unitarias para los servicios clave de la aplicación, para garantizar que cualquier cambio en el código no rompa las funcionalidades existentes.*
Criterios de Aceptación	▪ Pruebas unitarias cubriendo el código crítico
Estimación	4
Prioridad	3

Tabla 3.16.: Historia Técnica HT. 5

ID	HT. 6
Nombre	**Optimización de la Base de Datos**
Descripción	*Como desarrollador, quiero optimizar la base de datos para asegurar un acceso rápido y eficiente a los datos, especialmente cuando se trate de grandes volúmenes de información.*
Criterios de Aceptación	▪ Creación de índices para mejorar la velocidad de consulta. ▪ Pruebas de rendimiento para verificar la eficiencia de las consultas.
Estimación	4
Prioridad	4

Tabla 3.17.: Historia Técnica HT. 6

3.10. Cálculo de la Velocidad

Se ha considerado que dedicará un tiempo fijo al proyecto cada semana. Se estima que esta dedicación será de aproximadamente 8 horas semanales en promedio.

Dado que cada iteración del proyecto está planificada para tener una duración de alrededor de 2,5 semanas, se puede calcular la velocidad media para el desarrollo de la aplicación: Velocidad = 15 semanas * 10 horas/semana = 150 horas.

Parte IV.

Capítulo 4 - Implementación

4.1. Elección de las tecnologías

4.1.1. Base de datos

Para la gestión de datos, utilizaré MongoDB Cloud, una herramienta de administración de bases de datos multiplataforma que ofrece una interfaz gráfica intuitiva para interactuar con varias bases de datos NoSQL.

Los intentos de extraer datos de los dispositivos resultaron insuficientes, ya que no se logró obtener la variedad y cantidad de datos dispersos requeridos para un análisis significativo. Por esta razón, se recurrió a la búsqueda de conjuntos de datos ya disponibles que contuvieran la información necesaria. Posteriormente, se muestra cómo se obtuvieron estos datos, cómo fueron limpiados y analizados, y finalmente cómo se insertaron en una base de datos MongoDB.

4.1.1.1. Búsqueda de Datos

Para suplir la falta de acceso directo a los datos de los dispositivos de salud, se llevó a cabo una búsqueda exhaustiva de conjuntos de datos que pudieran cumplir con los objetivos de este estudio. Esta búsqueda se realizó a través de diversas plataformas, como Kaggle, Google Search, y páginas especializadas en tesis y estudios científicos. Se exploraron diferentes recursos para encontrar la mayor cantidad posible de datos que provinieran de relojes de salud inteligentes.

Durante esta fase de investigación, se encontraron varios conjuntos de datos, pero la mayoría no cumplían con los criterios de calidad o no tenían la diversidad de información necesaria para el análisis. Entre los conjuntos de datos disponibles, solo se consideraron realmente factibles los correspondientes al Apple Watch y Fitbit, ya que los datos de otros dispositivos no estaban disponibles o no cumplían con los requisitos de completitud y calidad para su estudio. Estos dos conjuntos de datos fueron seleccionados para el análisis, dado que proporcionaban información relevante sobre la actividad física y los parámetros de salud de los usuarios.

4.1.1.2. Limpieza de Datos

Una vez obtenidos los conjuntos de datos de Apple Watch y Fitbit, fue necesario llevar a cabo una limpieza exhaustiva para garantizar que los datos estuvieran en un formato adecuado para su posterior almacenamiento y análisis. La limpieza de datos es un paso fundamental para eliminar inconsistencias, datos faltantes y posibles errores que puedan sesgar los resultados del análisis.

Se utilizaron varios scripts de Python para transformar estos conjuntos de datos. Los scripts se encargaron de normalizar los nombres de los campos, eliminar registros incompletos, y convertir las unidades de medida para que fueran consistentes a lo largo de todo el conjunto de datos. Además, se llevaron a cabo procesos de codificación para transformar las variables categóricas en valores numéricos, lo cual facilitó el análisis estadístico y la integración en la base de datos. El objetivo principal de esta etapa fue garantizar que los datos estuvieran limpios y fueran consistentes, lo cual es esencial para obtener resultados fiables en el análisis posterior.

4.1.1.3. Almacenamiento de Datos

Tras la limpieza de los conjuntos de datos, se procedió a su almacenamiento en una base de datos NoSQL utilizando MongoDB. MongoDB fue seleccionada debido a su flexibilidad para manejar datos no estructurados y su capacidad para escalar de acuerdo con el volumen de datos. La estructura de la base de datos se diseñó de manera que cada dispositivo de salud tuviera su propia colección, almacenando información como los pasos diarios, la frecuencia cardiaca, las calorías quemadas y la distancia recorrida, entre otros.

Figura 4.6.: Logo de MongoDB

En este contexto, se presentaron dos colecciones principales: una para los datos de Apple Watch y otra para los datos de Fitbit. Estas colecciones fueron diseñadas con el objetivo de ser lo suficientemente flexibles como para adaptarse a futuros cambios en la estructura de los datos o incorporar nuevos dispositivos.

La organización de los datos en estas colecciones permite un acceso eficiente y facilita el análisis de las relaciones entre las distintas variables de salud y actividad física. Esto es especialmente importante al considerar la gran cantidad de datos generados por los usuarios de estos dispositivos de salud inteligentes. Además, la base de datos NoSQL facilita la incorporación de nuevos tipos de datos y nuevas colecciones en caso de que se requiera ampliar el análisis en el futuro.

A continuación se presentan las tablas con la descripción de los campos almacenados en la base de datos. Estas tablas contienen los tipos de datos utilizados en cada colección junto con una explicación detallada de cada campo.

Dato	Descripción
Unnamed: 0	Índice de la entrada de datos, que sirve como identificador único.
age	Edad del participante en años.
gender	Género del participante, representado numéricamente (por ejemplo, 1 para hombre, 2 para mujer).
height	Altura del participante en centímetros.
weight	Peso del participante en kilogramos.
Applewatch.Steps_LE	Pasos registrados por el Apple Watch, posiblemente ajustados para una estimación logarítmica.
Applewatch.Heart_LE	Frecuencia cardiaca registrada por el Apple Watch, posiblemente ajustada para una estimación logarítmica.
Applewatch.Calories_LE	Calorías quemadas según el Apple Watch, posiblemente ajustadas para una estimación logarítmica.
Applewatch.Distance_LE	Distancia recorrida registrada por el Apple Watch, posiblemente ajustada para una estimación logarítmica.
EntropyApplewatchHeartPerDay_LE	Cálculo de la entropía de los datos diarios de frecuencia cardiaca del Apple Watch.
EntropyApplewatchStepsPerDay_LE	Cálculo de la entropía del conteo diario de pasos del Apple Watch.
RestingApplewatchHeartrate_LE	Frecuencia cardiaca en reposo registrada por el Apple Watch.
CorrelationApplewatchHeartrateSteps_LE	Correlación entre la frecuencia cardiaca y los pasos registrados por el Apple Watch.
NormalizedApplewatchHeartrate_LE	Frecuencia cardiaca normalizada a un rango específico para el análisis.
ApplewatchIntensity_LE	Intensidad del ejercicio estimada a partir de la frecuencia cardiaca y datos de movimiento del Apple Watch.
SDNormalizedApplewatchHR_LE	Desviación estándar de la frecuencia cardiaca normalizada del Apple Watch.
ApplewatchStepsXDistance_LE	Producto de los pasos y la distancia, representando el nivel de actividad del Apple Watch.
activity_trimmed	Estado de actividad categorizado del participante (por ejemplo, acostado, caminata a ritmo propio).

Tabla 4.18.: Descripción de los datos de Apple Watch

Dato	Descripción
Unnamed: 0	Índice de la entrada de datos, que sirve como identificador único.
age	Edad del participante en años.
gender	Género del participante, representado numéricamente (por ejemplo, 1 para hombre, 2 para mujer).
height	Altura del participante en centímetros.
weight	Peso del participante en kilogramos.
Fitbit.Steps_LE	Pasos registrados por el Fitbit, posiblemente ajustados para una estimación logarítmica.
Fitbit.Heart_LE	Frecuencia cardiaca registrada por el Fitbit, posiblemente ajustada para una estimación logarítmica.
Fitbit.Calories_LE	Calorías quemadas según Fitbit, posiblemente ajustadas para una estimación logarítmica.
Fitbit.Distance_LE	Distancia recorrida registrada por el Fitbit, posiblemente ajustada para una estimación logarítmica.
EntropyFitbitHeartPerDay_LE	Cálculo de la entropía de los datos diarios de frecuencia cardiaca del Fitbit.
EntropyFitbitStepsPerDay_LE	Cálculo de la entropía del conteo diario de pasos del Fitbit.
RestingFitbitHeartrate_LE	Frecuencia cardiaca en reposo registrada por el Fitbit.
CorrelationFitbitHeartrateSteps_LE	Correlación entre la frecuencia cardiaca y los pasos registrados por el Fitbit.
NormalizedFitbitHeartrate_LE	Frecuencia cardiaca normalizada a un rango específico para el análisis.
FitbitIntensity_LE	Intensidad del ejercicio estimada a partir de la frecuencia cardiaca y datos de movimiento del Fitbit.
SDNormalizedFitbitHR_LE	Desviación estándar de la frecuencia cardiaca normalizada del Fitbit.
FitbitStepsXDistance_LE	Producto de los pasos y la distancia, representando el nivel de actividad del Fitbit.
activity_trimmed	Estado de actividad categorizado del participante (por ejemplo, acostado, caminata a ritmo propio).

Tabla 4.19.: Descripción de los datos de Fitbit

4.1.2. Backend

Para el desarrollo del backend, he utilizado **FastAPI**, un framework moderno y de alto rendimiento para la creación de APIs utilizando Python. La elección de FastAPI se debe a sus múltiples ventajas:

- **Simplicidad y Productividad**: Permite el desarrollo rápido de APIs RESTful con menos líneas de código, y proporciona una interfaz de usuario automática con Swagger para la documentación de los endpoints.
- **Alto Rendimiento**: Está basado en Starlette y Pydantic, lo cual garantiza tiempos de respuesta bajos y alta eficiencia en la gestión de las solicitudes.
- **Validación Automática**: FastAPI integra automáticamente la validación de datos de entrada, lo cual reduce la probabilidad de errores y mejora la calidad del código.
- **Integración Sencilla con MongoDB**: Gracias a su compatibilidad con librerías como pymongo, FastAPI facilita la conexión con bases de datos como **MongoDB**, que es la base de datos utilizada en este proyecto.

FastAPI me ha permitido construir un backend robusto y escalable, garantizando la eficiencia y la seguridad de las operaciones sobre los datos de salud recopilados y almacenados en **MongoDB**.

Figura 4.7.: Logo de FastAPI

4.1.3. Frontend

Para el desarrollo del frontend, he decidido utilizar **React**, una biblioteca de JavaScript ampliamente usada para construir interfaces de usuario. **React** permite desarrollar aplicaciones dinámicas y de alto rendimiento, con una experiencia de usuario fluida e interactiva.

React ofrece varias ventajas que lo hacen ideal para el desarrollo de este proyecto:

- **Componentización**: React permite dividir la interfaz de usuario en componentes reutilizables, lo que facilita el desarrollo y el mantenimiento del código. Cada componente puede gestionarse de forma independiente, haciendo que el código sea más modular y escalable.
- **Virtual DOM**: Utiliza un Virtual DOM que mejora significativamente el rendimiento de la aplicación al minimizar las actualizaciones reales del DOM en el navegador, lo cual se traduce en una experiencia de usuario más rápida y fluida.

- **Amplia Comunidad y Ecosistema**: React tiene una comunidad muy grande y un ecosistema robusto que incluye una gran cantidad de bibliotecas, extensiones y herramientas de desarrollo. Esto hace que el desarrollo sea más rápido y se puedan implementar características avanzadas sin necesidad de reinventar la rueda.

El uso de **React** permite la creación de una interfaz de usuario atractiva y eficiente que puede comunicarse de forma fluida con el backend desarrollado con FastAPI. Además, React se combina perfectamente con librerías para realizar solicitudes HTTP a la API, y con otras para gestionar el estado de la aplicación de manera centralizada, garantizando una mejor organización y predictibilidad del estado de la UI.

En resumen, la elección de **FastAPI** para el backend, junto con **React** para el frontend, proporciona una arquitectura moderna y eficiente que facilita tanto el desarrollo como el mantenimiento del proyecto.

Figura 4.8.: Logo de react

4.2. Arquitectura de la aplicación

Se ha diseñado la aplicación siguiente el patrón de arquitectura de Modelo Vista Controlador (MVC). Este patrón de diseño separa las responsabilidades de la aplicación en estas tres componentes principales. Esto facilita el desarrollo, escalabilidad y mantenimiento de la misma.

4.2.1. Diagrama de arquitectura de la aplicación

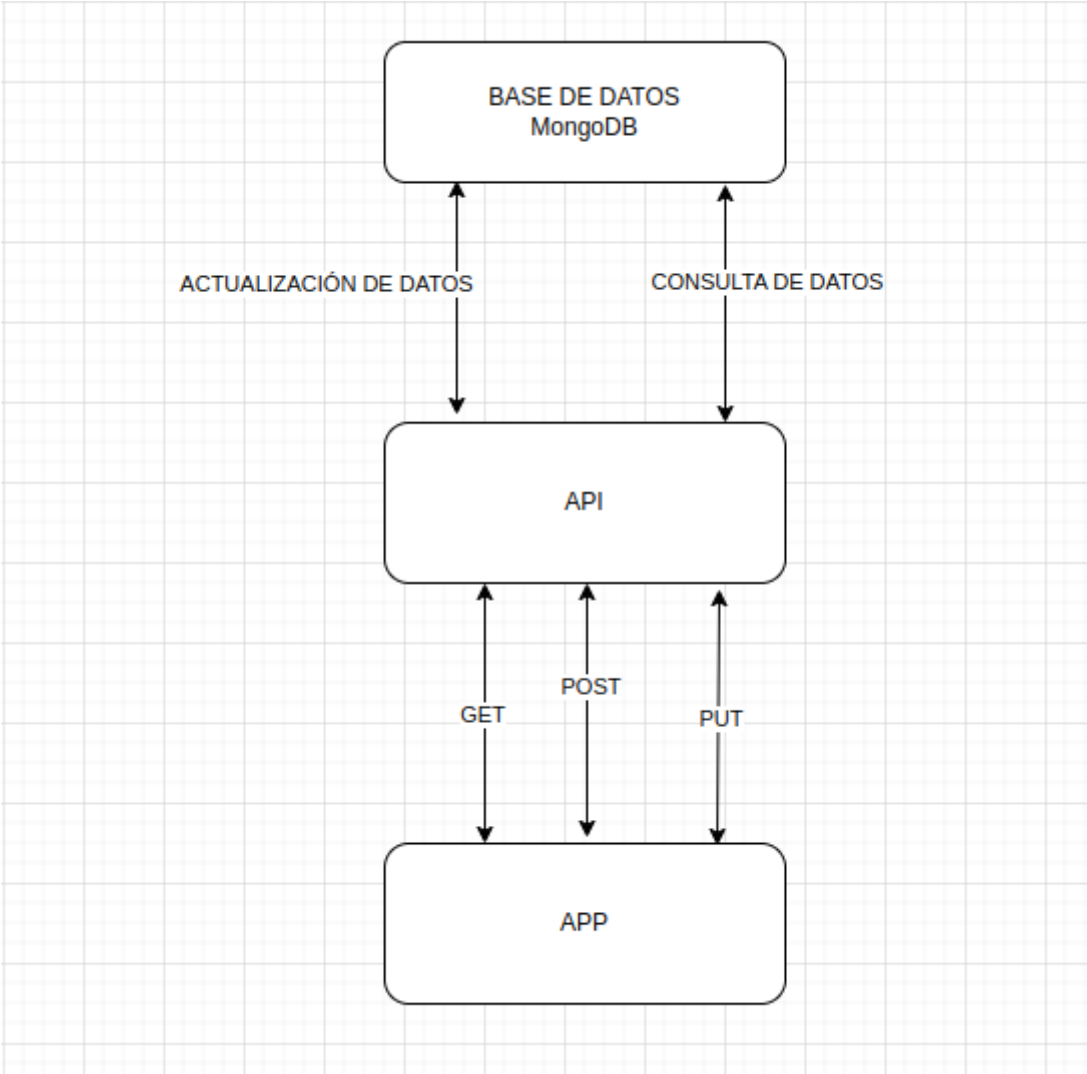


Figura 4.9.: Diagrama de arquitectura de aplicación

4.2.2. Modelo Vista Controlador (MVC)

El patrón **MVC** es una **arquitectura de software** que divide una **aplicación** en tres partes interrelacionadas:

- **Modelo**: Representa la lógica de negocio y los datos de la aplicación. El modelo se encarga de gestionar la información de **usuarios**, recursos disponibles, etc.
- **Vista**: Es la interfaz de usuario, es decir, lo que el usuario ve en pantalla. En una **aplicación web**, la vista suele estar construida con **HTML, CSS y JavaScript**. Su función es presentar la información del modelo de manera clara y atractiva al usuario.
- **Controlador**: Actúa como intermediario entre el modelo y la vista. Gestiona las solicitudes del usuario, actualiza el modelo en consecuencia y selecciona la vista adecuada para mostrar al usuario. En este contexto, el controlador gestiona las acciones.

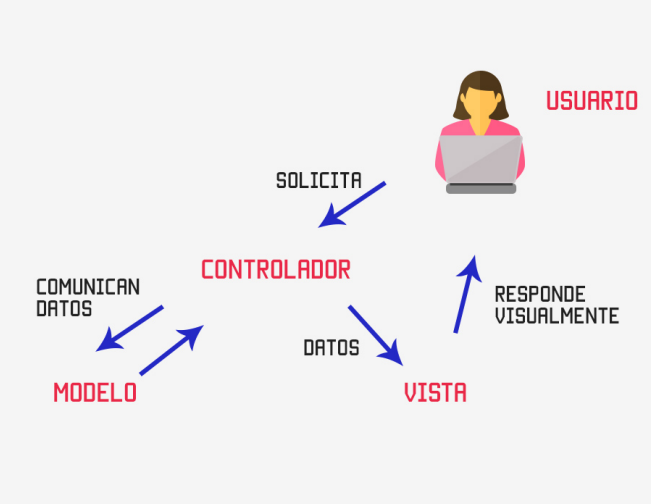


Figura 4.10.: Modelo-Vista-Controlador

4.2.3. Implementación del MVC

Veamos cómo se implementa el patrón MVC en la aplicación:

- **Modelo**:
  - **Base de Datos**: Se utiliza MongoDB, una base de datos NoSQL, para almacenar los datos de los usuarios, dispositivos y estadísticas de salud. MongoDB permite una estructura flexible de los datos, lo cual es ideal para la variedad de información recolectada por los dispositivos.
  - **ORM**: Aunque MongoDB no es una base de datos relacional, se emplean librerías como Pydantic para interactuar con la base de datos de una manera más estructurada y orientada a objetos. Esto permite definir esquemas para validar los datos antes de almacenarlos.

- **Vista**:
  - **Framework Frontend**: Para construir la interfaz de usuario, se utiliza React, que permite una experiencia de usuario dinámica y modular. React facilita el desarrollo de una UI fluida, permitiendo a los usuarios visualizar los datos en tiempo real.
  - **Componentes**: La interfaz se divide en componentes reutilizables. Estos componentes hacen que el desarrollo sea más organizado y permiten la reutilización de código a lo largo de la aplicación.
  - **Templates**: Se utilizan plantillas JSX en React para definir la estructura de cada componente, permitiendo una integración directa con la lógica del frontend, facilitando así la actualización y visualización de datos de manera dinámica.
- **Controlador**:
  - **Framework Backend**: El controlador está implementado usando FastAPI. FastAPI maneja las solicitudes HTTP entrantes, realiza las validaciones necesarias y devuelve respuestas adecuadas.
  - **Rutas**: Se definen rutas en FastAPI que mapean las URL a funciones específicas del controlador. Por ejemplo, rutas para obtener datos de los dispositivos o agregar nuevos registros. Estas rutas gestionan las solicitudes de los usuarios y coordinan la interacción con el modelo y la vista.
  - **Lógica de Negocio**: Las funciones del controlador se encargan de validar los datos de las solicitudes, interactuar con la base de datos (a través del modelo) y devolver los datos a la vista de manera estructurada. Además, se emplean validaciones automáticas proporcionadas por Pydantic para garantizar la integridad de los datos recibidos.

Los beneficios de usar el patrón **MVC** son numerosos, entre ellos podemos destacar que separa las responsabilidades, facilitando así el desarrollo, mantenimiento y pruebas de la aplicación. Este patrón permite crear componentes reutilizables y un código más modular, lo cual contribuye a agregar nuevas funcionalidades de manera más sencilla, además de hacer más eficiente la localización y corrección de errores.

La elección del patrón **MVC** para esta aplicación web ha sido fundamental para garantizar una arquitectura sólida, escalable y mantenible. Al separar las diferentes responsabilidades de la aplicación, se ha logrado un código más limpio y organizado, lo que facilita su desarrollo y evolución a largo plazo.

4.3. Implementación de los sprints

A continuación se va a indicar como fueron implementados y desarrollados las distintas iteraciones

4.3.1. Sprint 1

El Sprint 01 tiene como objetivo establecer la base del proyecto mediante la configuración del entorno de desarrollo y la implementación inicial de la base de datos. Esto asegurará que todos los desarrolladores puedan trabajar de manera eficiente y que se cuente con una estructura sólida para el almacenamiento de los datos. A continuación se detalla el desarrollo del sprint, los objetivos, las historias a implementar, y la descomposición en tareas específicas.

4.3.1.1. Objetivos del Sprint

Los principales objetivos del Sprint 01 son configurar el entorno de desarrollo con todas las herramientas necesarias para el equipo, implementar la base de datos MongoDB para el almacenamiento eficiente de los datos del sistema, y garantizar que todo el equipo esté alineado en el uso de las herramientas y la infraestructura.

4.3.1.2. Historias a Desarrollar

ID	**HT. 1**
Nombre	**Configuración del Entorno de Desarrollo**
Descripción	*Como desarrollador, quiero configurar el entorno de desarrollo con todas las herramientas necesarias, para asegurar que el equipo pueda comenzar a trabajar rápidamente.*
Criterios de Aceptación	▪ Instalación de Python, React, npm, fastapi, uvicorn, IDEs, dependencias y librerías necesarias. ▪ Configuración de Git para el control de versiones.
Estimación	2
Prioridad	1

Tabla 4.20.: Historia Técnica HT. 1

ID	HT. 2
Nombre	**Implementación de la Base de Datos**
Descripción	*Como desarrollador, quiero implementar la base de datos MongoDB, para almacenar de manera eficiente los datos de los usuarios y dispositivos.*
Criterios de Aceptación	▪ Configuración de la conexión con MongoDB. ▪ Creación de colecciones y estructura de datos necesaria.
Estimación	3
Prioridad	2

Tabla 4.21.: Historia Técnica HT. 2

4.3.1.3. Descomposición en Tareas

Para cumplir con los objetivos y completar las historias seleccionadas, se han descompuesto en las siguientes tareas de desarrollo e implementación:

4.3.1.4. HT. 1 - Configuración del Entorno de Desarrollo

- Instalación de herramientas de desarrollo: Python, React, npm, FastAPI, Uvicorn, IDEs y librerías necesarias.

  He desarrollado un script automatizado en Linux para instalar todas las herramientas necesarias para el desarrollo del proyecto. Este script incluye la instalación de Python, React, npm, FastAPI, Uvicorn, y los entornos de desarrollo como IntelliJ IDEA y PyCharm. A continuación, se presenta el código del script:

```
#!/bin/bash

# Update package list and upgrade existing packages
sudo apt update && sudo apt upgrade -y

# Install Python
sudo apt install -y python3 python3-pip

# Install Node.js and npm (includes npm, which is needed for React)
sudo apt install -y nodejs npm

# Install FastAPI and Uvicorn
sudo apt install -y python3-fastapi python3-uvicorn

# Install React globally using npm
sudo npm install -g create-react-app

# Download and install IntelliJ IDEA (Community Edition) and PyCharm (
    Community Edition) from JetBrains
```

```
# For Ubuntu-based systems, JetBrains Toolbox can be used to manage
    installations
curl -L "https://download.jetbrains.com/toolbox/jetbrains-toolbox
    -1.26.3.13458.tar.gz" -o jetbrains-toolbox.tar.gz
tar -xzf jetbrains-toolbox.tar.gz
cd jetbrains-toolbox-*/
./jetbrains-toolbox

# Clean up installation files
cd ..
rm -rf jetbrains-toolbox*
```

Código 4.1: Script de instalación automática de las herramientas necesarias

A continuación instalo los IDEs intellij IDEA y pyCharm que son los que he utilizado para el desarrollo del software. Desde la aplicación jetbrains toolbox es muy intuitivo.

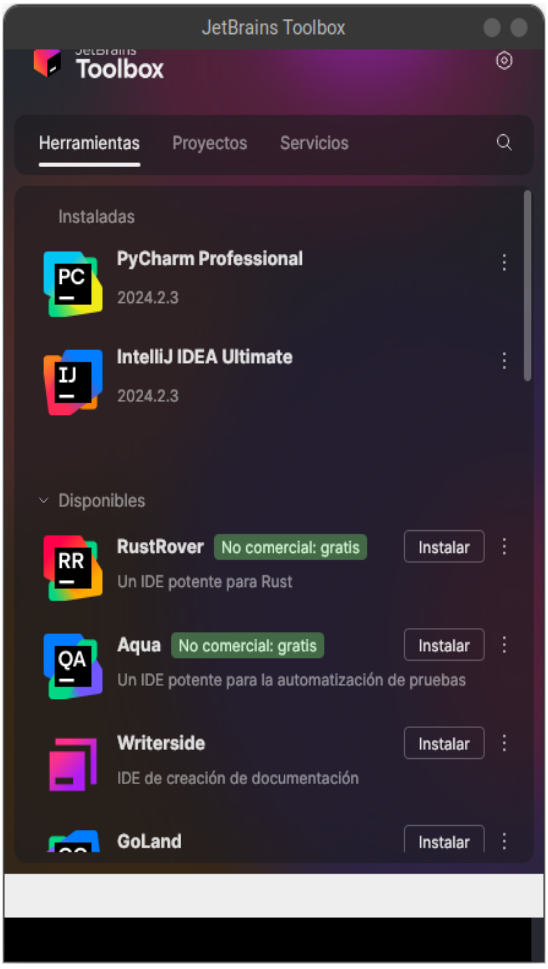


Figura 4.11.: Instalación de IDEs necesarios desde jetbrains toolbox

- Configuración de Git para el control de versiones y creación de un repositorio centralizado.

  En el desarrollo de este proyecto, usaremos GitHub como la plataforma para el control de versiones y el seguimiento de los sprints. GitHub es una herramienta en la nube que facilita la colaboración entre desarrolladores y permite gestionar el código fuente de manera eficiente. Entre las principales ventajas que ofrece se encuentran:

  El control de versiones, que nos permitirá mantener un historial detallado de los cambios realizados en el código a lo largo del tiempo.

  La colaboración, ya que GitHub ofrece herramientas útiles como las pull requests para revisar y fusionar cambios, así como gestionar problemas y tareas. Aunque en este proyecto solo estaré colaborando yo, estas herramientas serán útiles para estructurar y organizar mi trabajo.

  También cuenta con un sistema de seguimiento de problemas, que facilita informar sobre errores, hacer solicitudes de nuevas funcionalidades y gestionar el avance de las tareas, lo cual será útil para mantener un backlog organizado de los sprints.

  Otra ventaja es la integración con diversas herramientas y servicios que permiten automatizar procesos como pruebas y despliegues, haciendo más fácil la integración y entrega continuas.

  En el proyecto seguiremos una estrategia de desarrollo basada en ramas. Tendremos una rama principal llamada 'main', que servirá como el punto central del proyecto. A partir de ella, crearé ramas secundarias para trabajar en las distintas etapas y sprints, asegurando que el desarrollo esté bien organizado y controlado.

Figura 4.12.: Logo de github

Para configurar Git y crear un repositorio centralizado, esto es lo que hice:

Primero, me aseguré de tener Git instalado en mi sistema. Si no estaba instalado, utilicé el comando *sudo apt install git* para instalarlo. Luego, configuré mi identidad en Git para que los commits estuvieran correctamente identificados, usando *git config –global*

Después, inicialicé un repositorio Git en el directorio raíz del proyecto con *git init*. Esto creó un directorio oculto llamado '.git' que contiene todos los archivos necesarios para el control de versiones. Añadí todos los archivos del proyecto al área de preparación con *git add .* y luego realicé un commit inicial con *git commit -m "Commit inicial del proyecto"* para guardar el estado inicial.

Luego, fui a GitHub y creé un nuevo repositorio. Copié el enlace HTTPS o SSH del repositorio y lo añadí a mi proyecto local con
git remote add origin https://github.com/phuertass/tfg.git

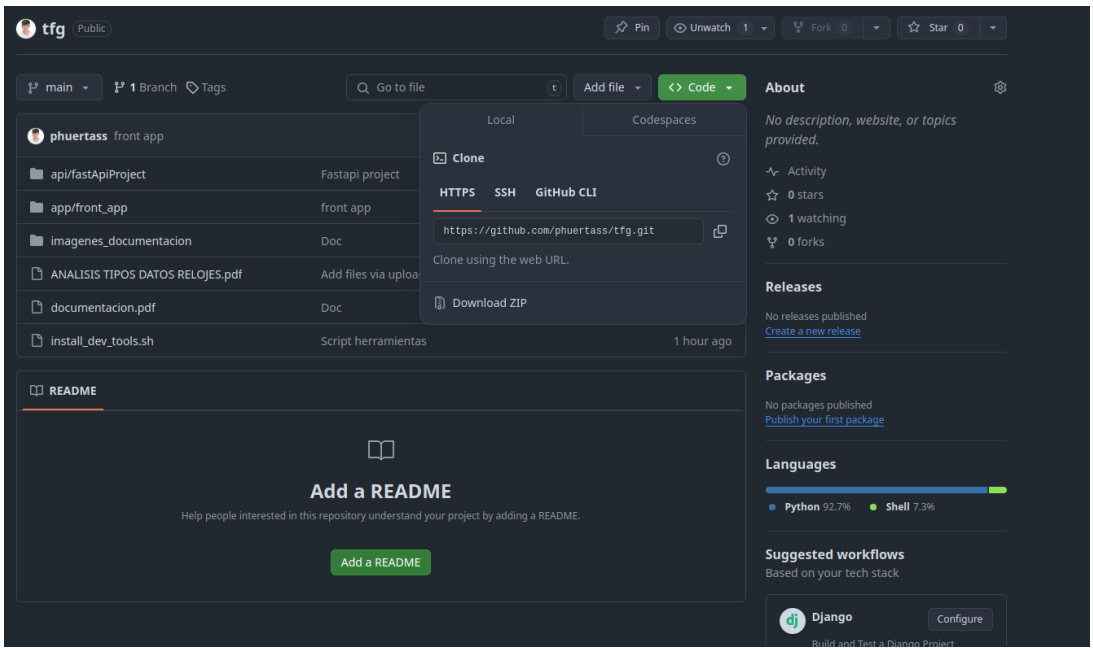


Figura 4.13.: Inicialización de repositorio git

Como GitHub ya no permite la autenticación con contraseña, generé un Token de Acceso Personal (PAT). Fui a [GitHub Tokens], seleccioné Generate new token y elegí los permisos necesarios, como *repo*. Copié el token generado y lo guardé de manera segura, ya que no podría verlo nuevamente.

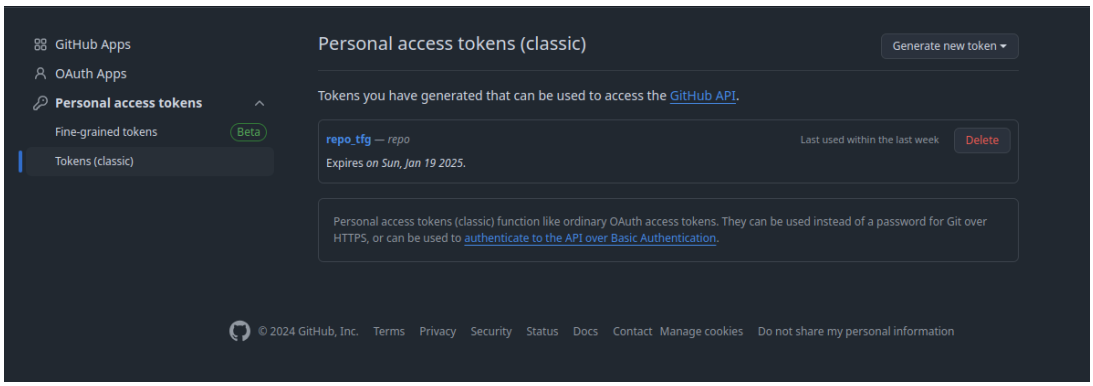


Figura 4.14.: Generar token válido para acceder al repositorio

Finalmente, subí los cambios al repositorio remoto con *git push -u origin main*. Cuando me pidió la contraseña, ingresé el token de acceso personal que había generado. Como el repositorio remoto ya contenía algunos archivos (como un README), tuve que hacer un 'pull' para fusionar los cambios antes de hacer el push, usando *git pull origin main –allow-unrelated-histories* Resolví cualquier conflicto que se presentó y luego volví a hacer el push.

Estos pasos me permitieron configurar Git para el control de versiones y crear un repositorio centralizado para el proyecto.

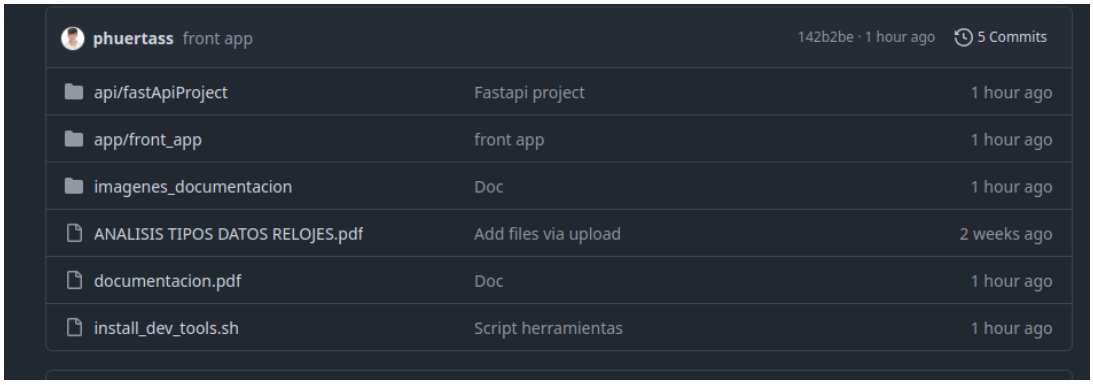
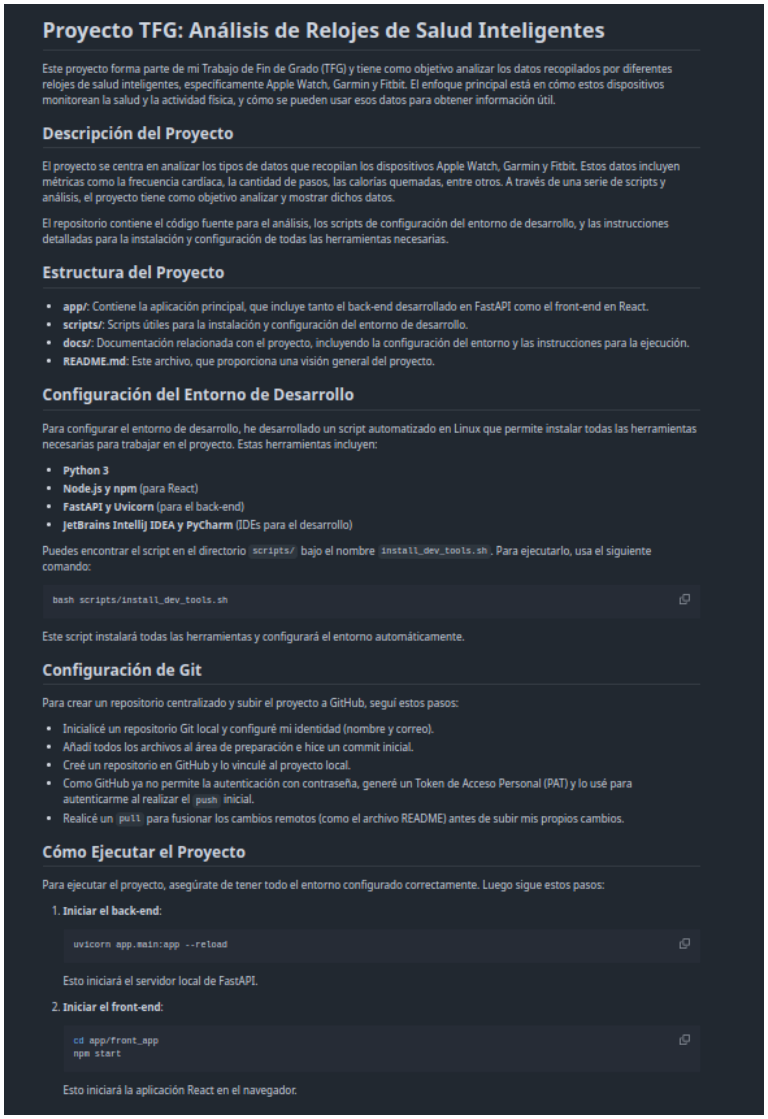


Figura 4.15.: Configuración de git finalizada

- Creación de un archivo README con instrucciones para configurar el entorno de manera uniforme.

  He creado un archivo README como portada del repositorio de git



Proyecto TFG: Análisis de Relojes de Salud Inteligentes

Este proyecto forma parte de mi Trabajo de Fin de Grado (TFG) y tiene como objetivo analizar los datos recopilados por diferentes relojes de salud inteligentes, específicamente Apple Watch, Garmin y Fitbit. El enfoque principal está en cómo estos dispositivos monitorean la salud y la actividad física, y cómo se pueden usar esos datos para obtener información útil.

Descripción del Proyecto

El proyecto se centra en analizar los tipos de datos que recopilan los dispositivos Apple Watch, Garmin y Fitbit. Estos datos incluyen métricas como la frecuencia cardíaca, la cantidad de pasos, las calorías quemadas, entre otros. A través de una serie de scripts y análisis, el proyecto tiene como objetivo analizar y mostrar dichos datos.

El repositorio contiene el código fuente para el análisis, los scripts de configuración del entorno de desarrollo, y las instrucciones detalladas para la instalación y configuración de todas las herramientas necesarias.

Estructura del Proyecto

- **app/**: Contiene la aplicación principal, que incluye tanto el back-end desarrollado en FastAPI como el front-end en React.
- **scripts/**: Scripts útiles para la instalación y configuración del entorno de desarrollo.
- **docs/**: Documentación relacionada con el proyecto, incluyendo la configuración del entorno y las instrucciones para la ejecución.
- **README.md**: Este archivo, que proporciona una visión general del proyecto.

Configuración del Entorno de Desarrollo

Para configurar el entorno de desarrollo, he desarrollado un script automatizado en Linux que permite instalar todas las herramientas necesarias para trabajar en el proyecto. Estas herramientas incluyen:

- **Python 3**
- **Node.js y npm** (para React)
- **FastAPI y Uvicorn** (para el back-end)
- **JetBrains Intellij IDEA y PyCharm** (IDEs para el desarrollo)

Puedes encontrar el script en el directorio `scripts/` bajo el nombre `install_dev_tools.sh`. Para ejecutarlo, usa el siguiente comando:

```
bash scripts/install_dev_tools.sh
```

Este script instalará todas las herramientas y configurará el entorno automáticamente.

Configuración de Git

Para crear un repositorio centralizado y subir el proyecto a GitHub, seguí estos pasos:

- Inicialicé un repositorio Git local y configuré mi identidad (nombre y correo).
- Añadí todos los archivos al área de preparación e hice un commit inicial.
- Creé un repositorio en GitHub y lo vinculé al proyecto local.
- Como GitHub ya no permite la autenticación con contraseña, generé un Token de Acceso Personal (PAT) y lo usé para autenticarme al realizar el `push` inicial.
- Realicé un `pull` para fusionar los cambios remotos (como el archivo README) antes de subir mis propios cambios.

Cómo Ejecutar el Proyecto

Para ejecutar el proyecto, asegúrate de tener todo el entorno configurado correctamente. Luego sigue estos pasos:

1. **Iniciar el back-end**:

   ```
   uvicorn app.main:app --reload
   ```

   Esto iniciará el servidor local de FastAPI.

2. **Iniciar el front-end**:

   ```
   cd app/front_app
   npm start
   ```

   Esto iniciará la aplicación React en el navegador.

Figura 4.16.: Creación archivo README para el repositorio

4.3.1.5. HT. 2 - Implementación de la Base de Datos

En esta etapa, me enfoqué en la instalación y configuración de MongoDB para el servidor de desarrollo. Esto incluyó la creación de las colecciones necesarias para almacenar la información. Además, implementé la estructura de datos inicial y realicé pruebas de conexión desde la API para asegurar que la comunicación entre la base de datos y la aplicación funcionara correctamente.

MongoDB Atlas fue elegido para este proyecto por sus múltiples ventajas. Al ser una solución en la nube, ofrece una alta disponibilidad y escalabilidad sin necesidad de una configuración compleja. Además, su facilidad para gestionar el acceso y las credenciales, junto con la capacidad de automatizar tareas de mantenimiento, permite ahorrar tiempo y

esfuerzo en la administración de la base de datos. Estas características lo hacen ideal para proyectos que necesitan crecer de manera eficiente y segura.

Figura 4.17.: Mongodb Cloud

- Instalación y configuración de MongoDB en el servidor de desarrollo.

  En esta etapa, me enfoqué en la instalación y configuración de MongoDB para el servidor de desarrollo. Primero, configuré una base de datos en MongoDB Atlas, una plataforma en la nube que facilita la gestión y escalabilidad de bases de datos MongoDB. Creé un clúster gratuito en MongoDB Atlas, definiendo la ubicación geográfica del servidor para minimizar la latencia.

  Generé las credenciales de usuario necesarias para conectarme de forma segura a la base de datos.

  Para realizar la conexión desde la aplicación, utilicé la URI proporcionada por MongoDB Atlas, que tenía el siguiente formato:

  mongodb+srv://<usuario>:<contraseña>@clustero.m2rzk8b.mongodb.net/
  ssl=true&tlsAllowInvalidCertificates=true

  Reemplacé <usuario> y <contraseña> con las credenciales que generé previamente, y añadí esta URI a los archivos de configuración del proyecto. Luego, añadí esta URI a los archivos de configuración del proyecto para poder conectarme desde la API.

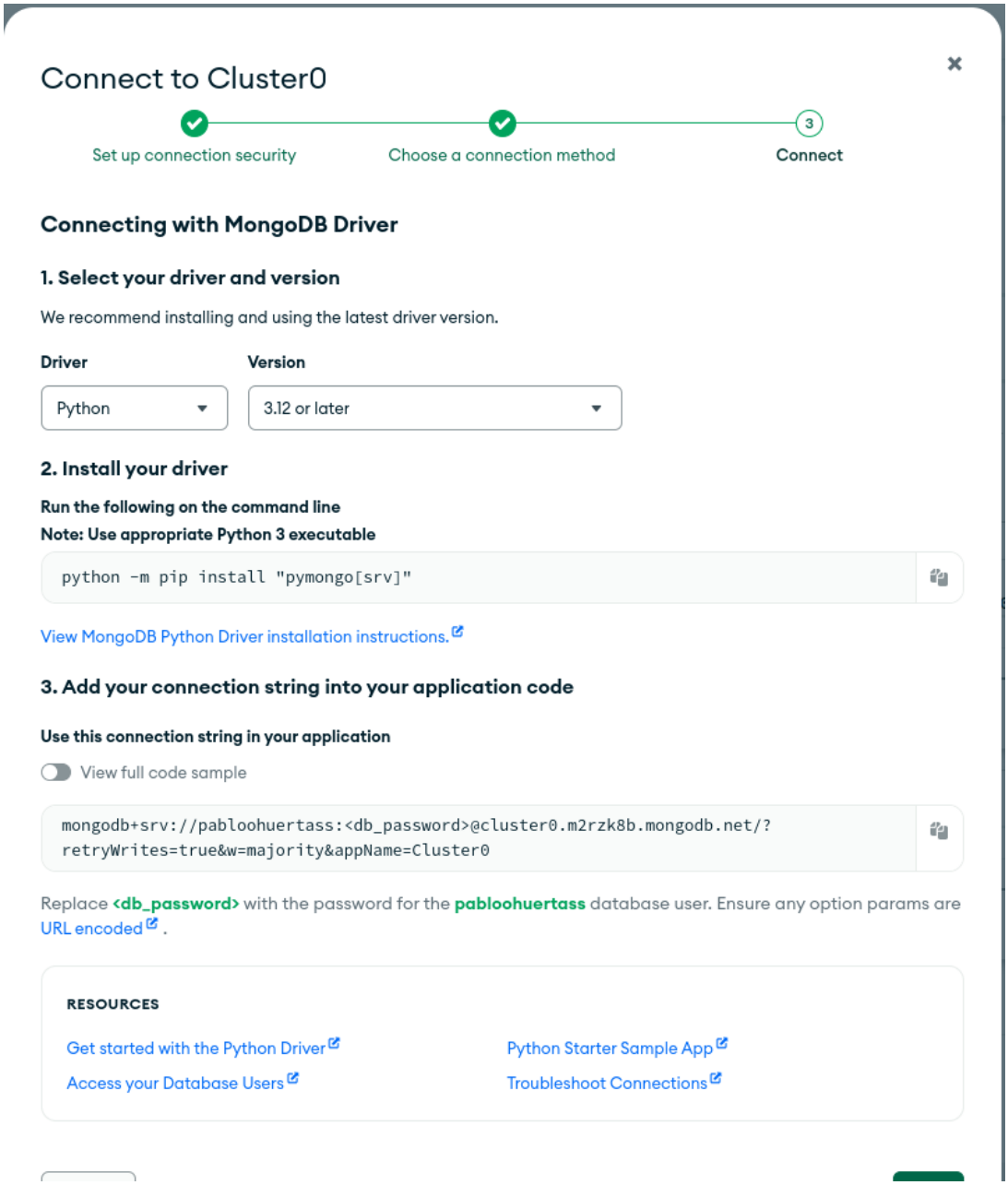


Figura 4.18.: Mongodb Cloud

- Creación de las colecciones necesarias para almacenar los datos de los usuarios y los dispositivos.

  Para almacenar los datos de los usuarios y dispositivos, creé dos colecciones en MongoDB Atlas: *aw_data* y *fitbit_data* La colección *aw_data* se utilizó para almacenar la información de los usuarios que utilizan Apple Watch, mientras que la colección *fit-*

bit_data se usó para los usuarios con dispositivos Fitbit.

Un ejemplo de documento almacenado en la colección *aw_data* es el siguiente:

```
{
  "_id": {"$oid": "?"},
  "age": {"$numberInt": "?"},
  "gender": {"$numberInt": "?"},
  "height": {"$numberDouble": "?"},
  "weight": {"$numberDouble": "?"},
  "Applewatch": {
    "Steps_LE": {"$numberDouble": "?"},
    "Heart_LE": {"$numberDouble": "?"},
    "Calories_LE": {"$numberDouble": "?"},
    "Distance_LE": {"$numberDouble": "?"}
  },
  "EntropyApplewatchHeartPerDay_LE": {"$numberDouble": "?"},
  "EntropyApplewatchStepsPerDay_LE": {"$numberDouble": "?"},
  "RestingApplewatchHeartrate_LE": {"$numberDouble": "?"},
  "CorrelationApplewatchHeartrateSteps_LE": {"$numberDouble": "?"},
  "NormalizedApplewatchHeartrate_LE": {"$numberDouble": "?"},
  "ApplewatchIntensity_LE": {"$numberDouble": "?"},
  "SDNormalizedApplewatchHR_LE": {"$numberDouble": "?"},
  "ApplewatchStepsXDistance_LE": {"$numberDouble": "?"},
  "activity_trimmed": "?"
}
```

Código 4.2: Ejemplo de documento almacenado en la colección aw_data

Asimismo, un ejemplo de documento almacenado en la colección *fitbit_data* es el siguiente:

```
{
  "_id": {"$oid": "?"},
  "age": {"$numberInt": "?"},
  "gender": {"$numberInt": "?"},
  "height": {"$numberDouble": "?"},
  "weight": {"$numberDouble": "?"},
  "Fitbit": {
    "Steps_LE": {"$numberDouble": "?"},
    "Heart_LE": {"$numberDouble": "?"},
    "Calories_LE": {"$numberDouble": "?"},
    "Distance_LE": {"$numberDouble": "?"}
  },
  "EntropyFitbitHeartPerDay_LE": {"$numberDouble": "?"},
  "EntropyFitbitStepsPerDay_LE": {"$numberDouble": "?"},
  "RestingFitbitHeartrate_LE": {"$numberDouble": "?"},
  "CorrelationFitbitHeartrateSteps_LE": {"$numberDouble": "?"},
  "NormalizedFitbitHeartrate_LE": {"$numberDouble": "?"},
  "FitbitIntensity_LE": {"$numberDouble": "?"},
  "SDNormalizedFitbitHR_LE": {"$numberDouble": "?"},
  "FitbitStepsXDistance_LE": {"$numberDouble": "?"},
  "activity_trimmed": "?"
}
```

Código 4.3: Ejemplo de documento almacenado en la colección fitbit_data

Estas colecciones permiten almacenar los datos necesarios para el análisis de la actividad física y la salud de los usuarios de manera estructurada y accesible.

Posteriormente, todos los datos fueron insertados a través de los csv correspondientes, teniendo al final un total de 3656 documentos en la coleccion *aw_data* y 2608 documentos en *fitbit_data*

4.3.1.6. Resumen del Sprint

Historia	Estimación	Prioridad
HT. 1	2	1
HT. 2	3	2

Tabla 4.22.: Historias del Sprint 01

- **Número Total de Tareas**: 5
- **Estimación Total**: 5

4.3.2. Sprint 2

El Sprint 02 tiene como objetivo desplegar la API para el acceso a los datos de los dispositivos y garantizar que se pueda acceder de manera segura a través de configuraciones específicas de seguridad. A continuación, se detalla el desarrollo del sprint, los objetivos, las historias a implementar, y la descomposición en tareas específicas.

4.3.2.1. Objetivos del Sprint

Los principales objetivos del Sprint 02 son desplegar la API utilizando FastAPI para proporcionar acceso a los datos, y configurar el middleware CORS para permitir el acceso seguro desde diferentes orígenes. Esto permitirá una gestión eficiente del acceso a la información recopilada por los dispositivos.

4.3.2.2. Historias a Desarrollar

ID	**HT. 3**
Nombre	**Despliegue de la API con FastAPI**
Descripción	*Como desarrollador, quiero desplegar la API utilizando FastAPI, para proporcionar acceso a los datos de los dispositivos de manera eficiente.*
Criterios de Aceptación	▪ Endpoints disponibles para consultar y filtrar datos. ▪ Documentación automática generada con Swagger.
Estimación	3
Prioridad	1

Tabla 4.23.: Historia Técnica HT. 3

ID	**HT. 4**
Nombre	**Integración de Middleware CORS**
Descripción	*Como desarrollador, quiero integrar un middleware CORS en la API, para permitir que diferentes clientes puedan acceder a los datos de forma segura.*
Criterios de Aceptación	▪ Configuración correcta del middleware CORS para permitir solicitudes desde orígenes específicos. ▪ Pruebas que confirmen el acceso seguro desde distintos clientes.
Estimación	2
Prioridad	2

Tabla 4.24.: Historia Técnica HT. 4

4.3.2.3. Descomposición en Tareas

Para cumplir con los objetivos y completar las historias seleccionadas, se han descompuesto en las siguientes tareas de desarrollo e implementación:

4.3.2.4. HT. 3 - Despliegue de la API con FastAPI

En esta etapa, el objetivo fue desplegar la API utilizando FastAPI para proporcionar un acceso eficiente a los datos almacenados en MongoDB Atlas. Para este propósito, desarrollé una estructura que permitiera una fácil gestión de los endpoints y un despliegue adecuado.

- Creación de los endpoints para consultar y filtrar los datos de los dispositivos.

  El primer apéndice incluye todos los aspectos de la organización de la api elaborada, así como su estructura, sus pruebas y sus endpoints.

  Véase A.1

```python
# main.py
from fastapi import FastAPI, HTTPException
from mongo_utils import connect_to_mongo, list_databases,
    list_collections, get_all_documents, get_documents_with_filter,
    get_documents_with_projection
import config
from fastapi.middleware.cors import CORSMiddleware

# Crear una instancia de la aplicación FastAPI
app = FastAPI(
    title="MongoDB API",
    description="API para interactuar con una base de datos MongoDB,
        proporcionando operaciones de consulta y recuperación de datos.
        ",
    version="1.0.0"
)

# Configurar CORS para permitir peticiones desde cualquier origen
origins = ["*"]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Conectar a MongoDB utilizando la cadena de conexión desde la
    configuración
client = connect_to_mongo(config.connection_string)

# Endpoint raíz para verificar si la API está funcionando
@app.get("/", summary="Endpoint raíz", tags=["General"])
async def root():
    """
```

```
103     Devuelve un mensaje de bienvenida para comprobar que la API está
            funcionando correctamente.
104     """
105     return {"message": "Hello World"}
106 
107 # Endpoint para saludar a un usuario por su nombre
108 @app.get("/hello/{name}", summary="Saludar a un usuario", tags=["
        General"])
109 async def say_hello(name: str):
110     """
111     Saluda al usuario proporcionando su nombre.
112 
113     - **name**: El nombre del usuario a saludar.
114     """
115     return {"message": f"Hello {name}"}
116 
117 # Endpoint para listar todas las bases de datos disponibles
118 @app.get("/databases", summary="Listar bases de datos", tags=["MongoDB
        "])
119 async def get_databases():
120     """
121     Obtiene una lista de todas las bases de datos disponibles en el
            cliente MongoDB.
122     """
123     databases = list_databases(client)
124     if not databases:
125         raise HTTPException(status_code=404, detail="No databases
                found")
126     return {"databases": databases}
127 
128 # Endpoint para listar todas las colecciones de una base de datos
        especificada
129 @app.get("/collections/{db_name}", summary="Listar colecciones", tags
        =["MongoDB"])
130 async def get_collections(db_name: str):
131     """
132     Lista todas las colecciones de una base de datos proporcionada.
133 
134     - **db_name**: El nombre de la base de datos.
135     """
136     collections = list_collections(client, db_name)
137     if not collections:
138         raise HTTPException(status_code=404, detail=f"No collections
                found in database {db_name}")
139     return {"collections": collections}
140 
141 # Endpoint para obtener todos los documentos de una colección
        especificada
142 @app.get("/documents/{db_name}/{collection_name}", summary="Obtener
        documentos", tags=["MongoDB"])
143 async def get_documents(db_name: str, collection_name: str):
144     """
```

```
145     Obtiene todos los documentos de una colección específica.
146 
147     - **db_name**: El nombre de la base de datos.
148     - **collection_name**: El nombre de la colección.
149     """
150     documents = get_all_documents(client, db_name, collection_name)
151     if not documents:
152         raise HTTPException(status_code=404, detail=f"No documents
            found in collection {collection_name}")
153     return {"documents": documents}
154 
155 # Endpoint para obtener documentos filtrados por un campo específico
156 @app.get("/documents/{db_name}/{collection_name}/filter", summary="
    Obtener documentos con filtro", tags=["MongoDB"])
157 async def get_filtered_documents(db_name: str, collection_name: str,
    age: int = None):
158     """
159     Obtiene documentos de una colección que cumplen con un filtro
        específico.
160 
161     - **db_name**: El nombre de la base de datos.
162     - **collection_name**: El nombre de la colección.
163     - **age**: (Opcional) Filtro por edad.
164     """
165     filter = {}
166     if age is not None:
167         filter["age"] = age
168     documents = get_documents_with_filter(client, db_name,
        collection_name, filter)
169     if not documents:
170         raise HTTPException(status_code=404, detail=f"No documents
            found with filter {filter}")
171     return {"documents": documents}
172 
173 # Endpoint para obtener documentos con filtro y proyección
174 @app.get("/documents/{db_name}/{collection_name}/projection", summary=
    "Obtener documentos con proyección", tags=["MongoDB"])
175 async def get_projected_documents(db_name: str, collection_name: str,
    age: int = None):
176     """
177     Obtiene documentos de una colección aplicando un filtro y una
        proyección específicos.
178 
179     - **db_name**: El nombre de la base de datos.
180     - **collection_name**: El nombre de la colección.
181     - **age**: (Opcional) Filtro por edad.
182     """
183     filter = {}
184     if age is not None:
185         filter["age"] = age
186     projection = {"_id": 0, "age": 1, "gender": 1}
187     documents = get_documents_with_projection(client, db_name,
```

```
        collection_name, filter, projection)
188     if not documents:
189         raise HTTPException(status_code=404, detail=f"No documents
                found with filter {filter} and projection {projection}")
190     return {"documents": documents}
```

Código 4.4: Fichero main de la api

- Implementación de la documentación automática de la API utilizando Swagger y Redoc.

  Para generar la documentación automática de la API en el archivo main.py, utilicé FastAPI, que ofrece una integración directa con Swagger y Redoc para generar documentación de manera sencilla.

  En el código, simplemente al inicializar la aplicación FastAPI, la herramienta automáticamente habilita la generación de documentación. Para ver esta documentación:

  - Utiliza la interfaz Swagger accediendo a la ruta /docs mientras el servidor de la API está en ejecución.

    A continuación se muestran algunas imágenes de como se puede ver esta documentación generada en una vista general. Además una prueba de solicitud a la api con retorno exitoso y otro con error.

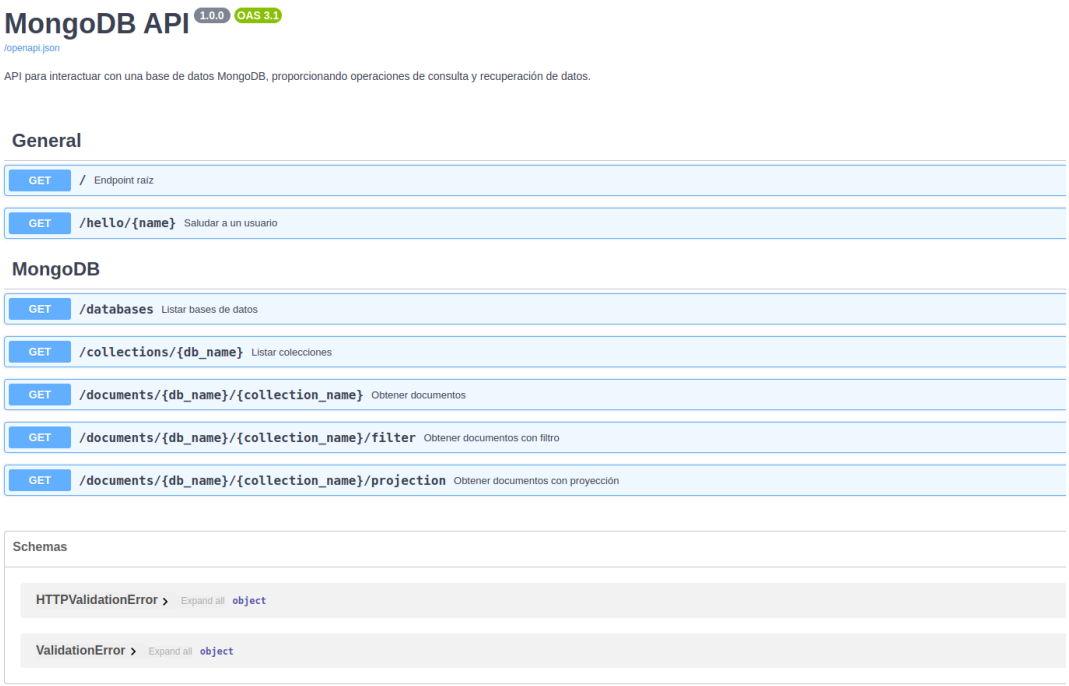


Figura 4.19.: Vista general documentación generada por swagger

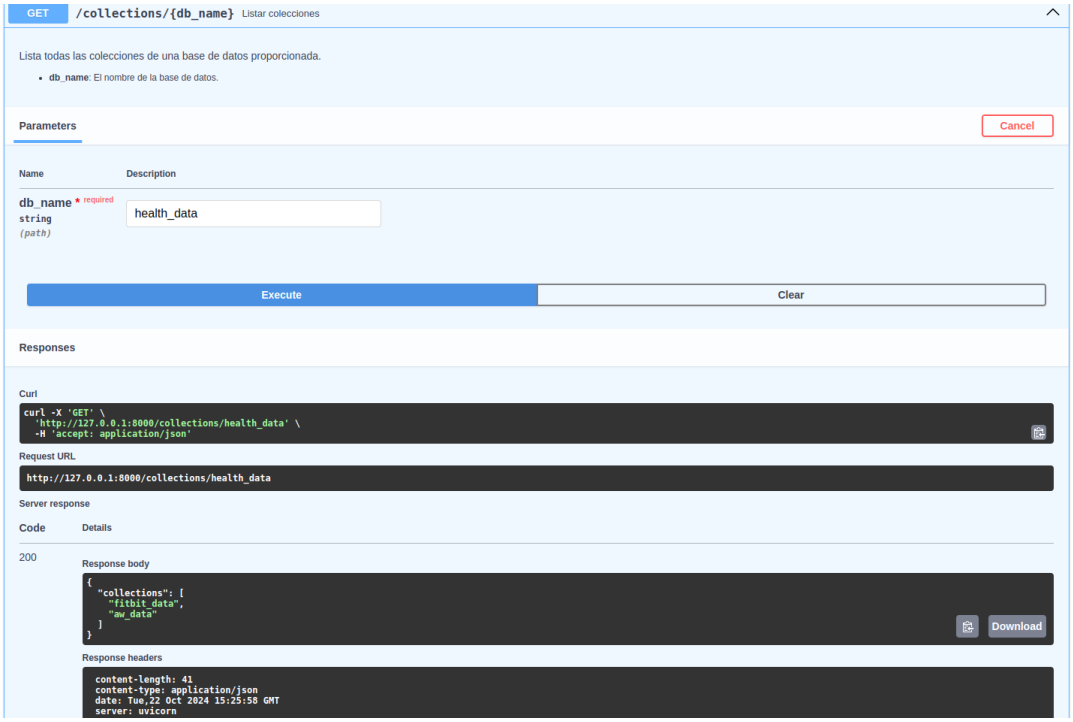


Figura 4.20.: Solicitud correcta con swagger

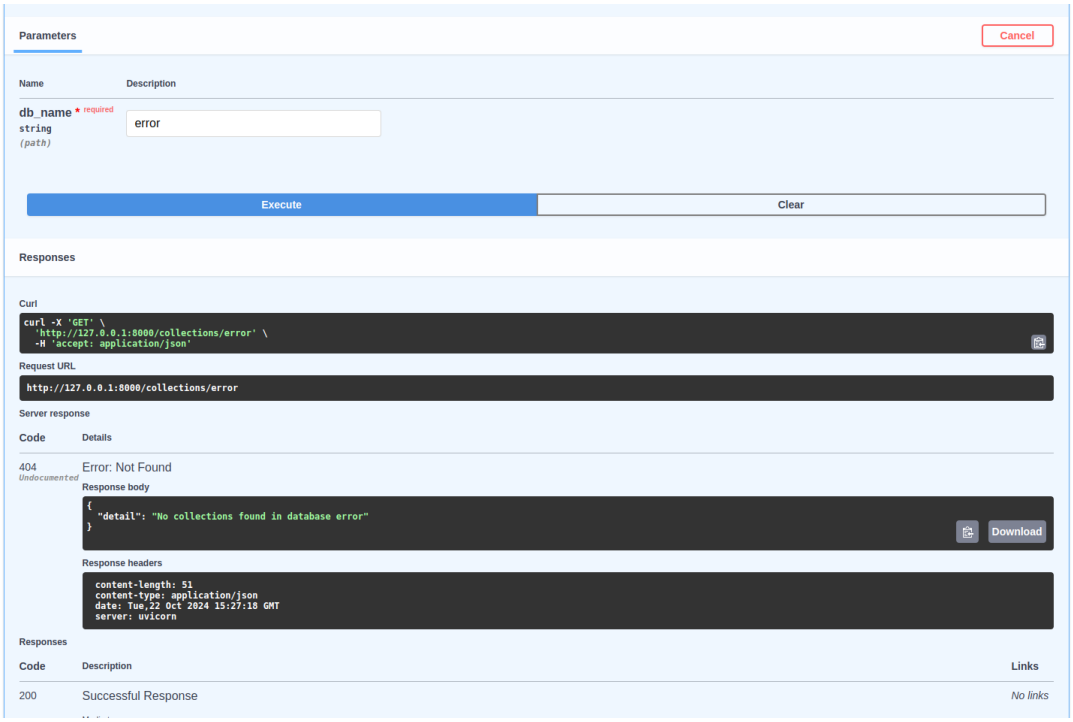


Figura 4.21.: Solicitud con retorno de error

- También puedes acceder a la documentación en formato Redoc visitando la ruta /redoc.

  En el caso de redoc, tenemos una interfaz un poco más moderna.

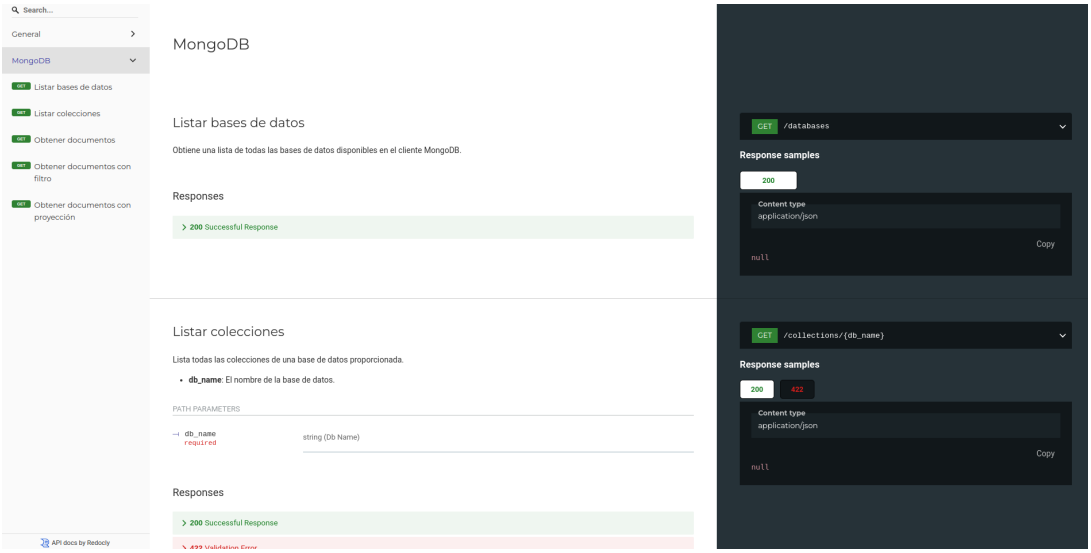


Figura 4.22.: Redoc vista general

De todas formas, la herramienta que más he usado para esto ha sido swagger.

Esto facilita la exploración de los endpoints, sus parámetros, respuestas posibles y ejemplos, lo cual es útil tanto para desarrolladores como para otros usuarios que deseen consumir la API.

- Pruebas de despliegue en un servidor de desarrollo para garantizar que la API funcione correctamente.

  Para garantizar que la API funcione correctamente, realicé las siguientes pruebas de despliegue:

  - **Despliegue en Local**: Inicialmente, desplegué la API en mi máquina local utilizando Uvicorn para asegurarme de que todo funcionaba correctamente. Utilicé `uvicorn main:app –reload` y realicé pruebas con herramientas como Postman y el navegador, accediendo a la documentación en `http://localhost:8000/docs`.
  - **Despliegue en un Entorno de Pruebas**: Configuré un servidor en la nube para simular un entorno de producción. Instalé Uvicorn y FastAPI en el servidor, y ejecuté la API con `uvicorn main:app –host 0.0.0.0 –port 8000`. Aseguré que el puerto 8000 estuviera abierto para recibir las solicitudes.
  - **Automatización de las Pruebas**: Creé un script de pruebas automatizadas para verificar que todos los endpoints respondían con los códigos de estado esperados (`200 OK`) y que los datos devueltos eran correctos y estaban en el formato adecuado.
  - **Pruebas de Concurrencia**: Utilicé herramientas como Apache JMeter y locust para realizar pruebas de estrés y verificar la capacidad de la API para manejar múltiples solicitudes simultáneas, asegurando que el rendimiento sea adecuado bajo carga.

  Después de realizar todas las pruebas mencionadas, los resultados demostraron que la API funcionó correctamente tanto en el entorno local como en el entorno de pruebas.

Las pruebas automatizadas y de concurrencia verificaron que los endpoints respondieran de manera adecuada y que el sistema pudiera manejar múltiples solicitudes sin problemas. En general, el despliegue fue exitoso, y la API mostró un rendimiento estable y eficiente bajo diferentes condiciones de carga.

4.3.2.5. HT. 4 - Integración de Middleware CORS

El objetivo de esta historia técnica fue integrar un middleware CORS (Cross-Origin Resource Sharing) en la API para permitir que diferentes clientes pudieran acceder de manera segura a los datos, asegurando que el acceso esté restringido según sea necesario.

Para cumplir este objetivo, se realizaron las siguientes tareas:

- **Instalación e Integración del Middleware CORS en la API**:

  Para permitir el acceso desde diferentes orígenes, instalé la librería `fastapi.middleware.cors`. Esta librería facilita la gestión de las políticas de CORS en FastAPI.

  Luego, se integró el middleware CORS en el archivo `main.py` de la API. El código para la configuración del middleware CORS es el siguiente:

  ```
  from fastapi.middleware.cors import CORSMiddleware

  app.add_middleware(
      CORSMiddleware,
      allow_origins=["*"],  # Permitir todos los orígenes
      allow_credentials=True,
      allow_methods=["*"],
      allow_headers=["*"],
  )
  ```

  En este proyecto, como se trata de un entorno de investigación, se decidió permitir todos los orígenes, utilizando la configuración `allow_origins=["*"]`, ya que esto facilita la interacción con la API desde cualquier cliente sin restricciones. Esta decisión está justificada porque el objetivo es abrir la API para realizar pruebas y experimentación.

- **Configuración de las Reglas para Permitir Solicitudes desde Orígenes Específicos**:

  Aunque la configuración utilizada en este entorno es abierta a todos los orígenes, es importante mencionar que, en un entorno de producción, lo ideal sería ser más restrictivo para evitar posibles vulnerabilidades de seguridad. Esto se podría hacer reemplazando `allow_origins=["*"]` con una lista de URLs específicas de clientes autorizados, por ejemplo:

  ```
  allow_origins=[
      "https://mi-cliente.com",
      "https://otro-cliente-autorizado.com"
  ]
  ```

- **Pruebas para Verificar que la API Esté Accesible desde Diferentes Clientes sin Comprometer la Seguridad**:

  Realicé pruebas desde diferentes clientes, incluyendo herramientas como Postman y navegadores, para verificar que las solicitudes se permitieran según lo esperado. Como la configuración permitía todos los orígenes, todas las solicitudes legítimas desde diferentes clientes fueron aceptadas sin problema.

  Las pruebas se realizaron tanto en el entorno local como en el servidor de desarrollo. Los resultados fueron satisfactorios, ya que todos los clientes pudieron acceder a la API correctamente, facilitando así la fase de investigación y desarrollo sin restricciones de acceso.

En resumen, la integración del middleware CORS permitió abrir el acceso a la API desde cualquier origen, facilitando el desarrollo y la investigación sin restricciones. En un entorno de producción, se recomendaría limitar el acceso a orígenes específicos para garantizar un mayor nivel de seguridad.

4.3.2.6. Resumen del Sprint

Historia	Estimación	Prioridad
HT. 3	3	1
HT. 4	2	2

Tabla 4.25.: Historias del Sprint 02

- **Número Total de Tareas**: 6
- **Estimación Total**: 5

4.3.3. Sprint 3

El Sprint 03 tiene como objetivo implementar pruebas unitarias para asegurar la calidad del código y optimizar la base de datos para mejorar la eficiencia en el acceso a los datos. A continuación, se detalla el desarrollo del sprint, los objetivos, las historias a implementar, y la descomposición en tareas específicas.

4.3.3.1. Objetivos del Sprint

Los principales objetivos del Sprint 03 son implementar pruebas unitarias para los servicios clave de la aplicación, garantizando que cualquier cambio en el código no afecte las funcionalidades existentes, y optimizar la base de datos para mejorar el rendimiento y la eficiencia en la consulta de datos. Estas tareas son esenciales para asegurar la estabilidad del proyecto y garantizar la escalabilidad y eficiencia del sistema.

4.3.3.2. Historias a Desarrollar

ID	**HT. 5**
Nombre	**Pruebas Unitarias**
Descripción	*Como desarrollador, quiero implementar pruebas unitarias para los servicios clave de la aplicación, para garantizar que cualquier cambio en el código no rompa las funcionalidades existentes.*
Criterios de Aceptación	▪ Pruebas unitarias cubriendo el código crítico
Estimación	4
Prioridad	3

Tabla 4.26.: Historia Técnica HT. 5

ID	**HT. 6**
Nombre	**Optimización de la Base de Datos**
Descripción	*Como desarrollador, quiero optimizar la base de datos para asegurar un acceso rápido y eficiente a los datos, especialmente cuando se trate de grandes volúmenes de información.*
Criterios de Aceptación	▪ Creación de índices para mejorar la velocidad de consulta. ▪ Pruebas de rendimiento para verificar la eficiencia de las consultas.
Estimación	4
Prioridad	4

Tabla 4.27.: Historia Técnica HT. 6

4.3.3.3. Descomposición en Tareas

Para cumplir con los objetivos y completar las historias seleccionadas, se han descompuesto en las siguientes tareas de desarrollo e implementación:

4.3.3.4. HT. 5 - Pruebas Unitarias

El objetivo de esta historia técnica es implementar pruebas unitarias para garantizar que los servicios clave de la aplicación continúen funcionando correctamente, incluso cuando se realicen cambios en el código. De este modo, se minimizan los riesgos de introducir errores en el proyecto al realizar mejoras o actualizaciones.

Para cumplir este objetivo, se realizaron las siguientes tareas:

- **Identificación del Código Crítico**:

  Primero, identifiqué las funcionalidades más críticas que requieren cobertura con pruebas unitarias. Esto incluyó las partes del código que son más susceptibles de cambiar y aquellas en las que una falla tendría el mayor impacto en el funcionamiento del sistema.

- **Implementación de Pruebas Unitarias**:

  Implementé las pruebas unitarias utilizando la librería `pytest` y la clase `TestClient` de FastAPI para simular solicitudes a los endpoints. Las pruebas se realizaron para los endpoints de la API, incluyendo:

  - **Pruebas del Endpoint `/get_aw_data`**: Verifiqué que el endpoint devolviera todos los datos de los usuarios que utilizan Apple Watch, y que la respuesta tuviera el formato esperado.
  - **Pruebas del Endpoint `/get_fitbit_data`**: Verifiqué que el endpoint devolviera todos los datos de los usuarios que utilizan Fitbit, asegurándome de que los datos fueran correctos.
  - **Pruebas de Errores**: Realicé pruebas para situaciones donde se soliciten datos de un usuario inexistente, comprobando que la respuesta fuera un error con el código adecuado (por ejemplo, `404 Not Found`).

- **Cobertura del Código Crítico**:

  Me aseguré de que las pruebas unitarias cubrieran todas las funcionalidades esenciales, enfocándome en cubrir el código crítico del proyecto. Esto incluyó la verificación de las respuestas de los endpoints y el manejo de errores para garantizar la robustez del sistema.

- **Ejecución de las Pruebas**:

  Las pruebas se ejecutaron utilizando `pytest`, y todas se realizaron en un entorno de desarrollo local. Al ejecutar `pytest`, se generaron reportes automáticos que permitieron visualizar el resultado de cada caso de prueba, identificando si había algún fallo o si todas las funcionalidades cubiertas por las pruebas funcionaban correctamente.

En resumen, la implementación de pruebas unitarias cubriendo el código crítico asegura que cualquier cambio en el sistema no afecte a las funcionalidades principales. Esto permite un desarrollo más seguro y minimiza los riesgos de introducir errores en el proyecto.

4.3.3.5. HT. 6 - Optimización de la Base de Datos

El objetivo de esta historia técnica es optimizar la base de datos para asegurar un acceso rápido y eficiente a los datos, especialmente cuando se trate de grandes volúmenes de información. Para lograr este objetivo, se realizaron las siguientes tareas:

- **Creación de Índices en las Colecciones de MongoDB**:

  Se crearon índices en las colecciones más consultadas de MongoDB, lo cual permitió mejorar significativamente la velocidad de las consultas y reducir la carga de la base de datos.

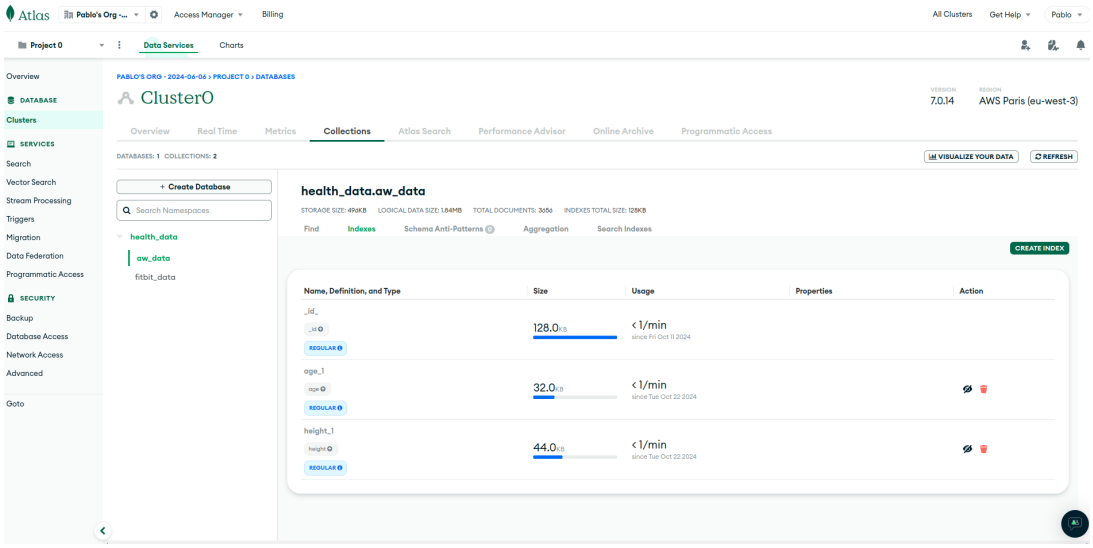


Figura 4.23.: Creación de índices en mongodb

- **Pruebas de Rendimiento**:

  El objetivo de esta subtarea es optimizar la base de datos para asegurar un acceso rápido y eficiente a los datos, especialmente cuando se trate de grandes volúmenes de información.

  Pruebas de Rendimiento Utilizando el Profiler de MongoDB Atlas:

  En mi intento por utilizar la funcionalidad de **Performance Advisor** para obtener recomendaciones automáticas de índices y optimizaciones, me encontré con una limitación.

  Debido a que el clúster utilizado es de nivel gratuito y no M10 o superior, no pude usar el **Performance Advisor** para sugerencias automáticas de optimización

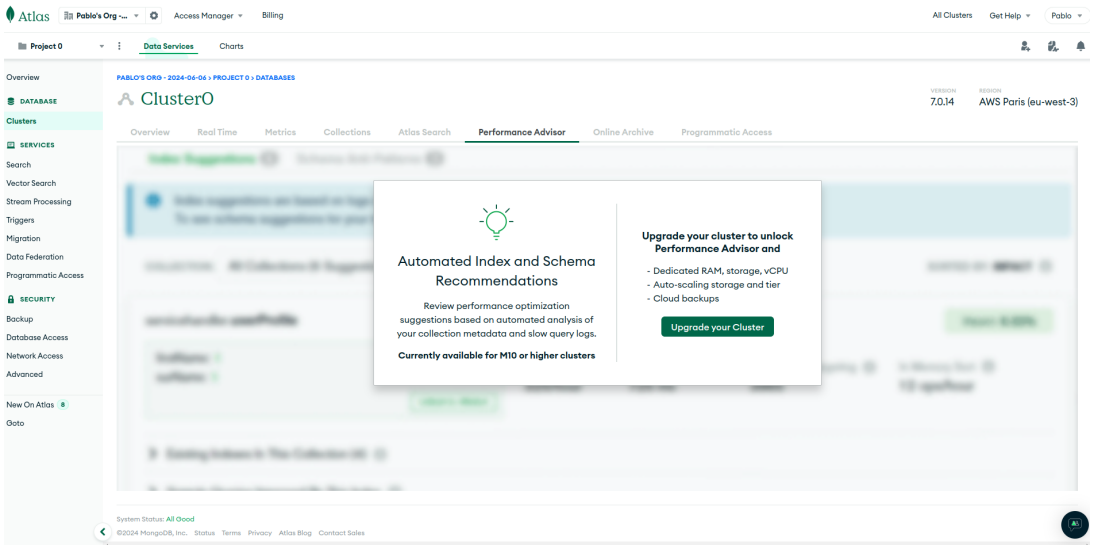


Figura 4.24.: Error al acceder al performance advisor

En resumen, la optimización de la base de datos no fue resuelta debido a la limitación de permisos de mi base de datos.

4.3.3.6. Resumen del Sprint

Historia	Estimación	Prioridad
HT. 5	4	3
HT. 6	4	4

Tabla 4.28.: Historias del Sprint 03

- **Número Total de Tareas**: 6
- **Estimación Total**: 5

4.3.4. Sprint 4

4.3.4.1. Objetivos del Sprint

El Sprint 04 tiene como objetivo implementar funcionalidades clave para la administración y visualización de datos de salud, garantizando que los usuarios puedan acceder a la información de manera eficiente y obtener resúmenes útiles de su actividad física y salud. A continuación, se detallan los objetivos, las historias a desarrollar, y la descomposición en tareas específicas.

4.3.4.2. Historias a Desarrollar

4.3.4.3. HU. 1 - Acceso a Datos Filtrados

ID	**HU. 1**
Nombre	**Acceso a Datos Filtrados**
Descripción	*Como administrador, quiero tener la capacidad de acceder a todos los datos almacenados en la base de datos con filtros específicos, para supervisar y mantener la calidad de los datos.*
Criterios de Aceptación	▪ Filtros específicos para buscar por dispositivo, edad y actividad. ▪ Resultados precisos excluyendo campos innecesarios como identificadores únicos.
Estimación	3
Prioridad	1

Tabla 4.29.: Historia de Usuario HU. 1

4.3.4.4. HU. 3 - Visualización de Datos de Salud

ID	**HU. 3**
Nombre	**Visualización de Datos de Salud**
Descripción	*Como usuario final, quiero visualizar un resumen de los datos de actividad física y salud.*
Criterios de Aceptación	▪ Mostrar total de pasos y calorías quemadas por cada tipo de actividad de cada tipo de reloj.
Estimación	3
Prioridad	2

Tabla 4.30.: Historia de Usuario HU. 3

4.3.4.5. Descomposición en Tareas

Para cumplir con los objetivos y completar las historias seleccionadas, se han descompuesto en las siguientes tareas de desarrollo e implementación:

4.3.4.6. HU. 1 - Acceso a Datos Filtrados

- **Aprendizaje de React y Creación Inicial de la App**:

  Comencé aprendiendo los fundamentos de React, ya que la interfaz de usuario se desarrollará con esta biblioteca. Luego, configuré el entorno inicial de desarrollo, incluyendo la instalación de npm para gestionar las dependencias del proyecto. Utilicé 'create-react-app' para crear la estructura base del proyecto, lo que me permitió establecer una base sólida y organizada para trabajar. Además, desarrollé un archivo principal (app.js) con la lógica inicial de la aplicación, que actúa como punto de partida para la interacción con el usuario.

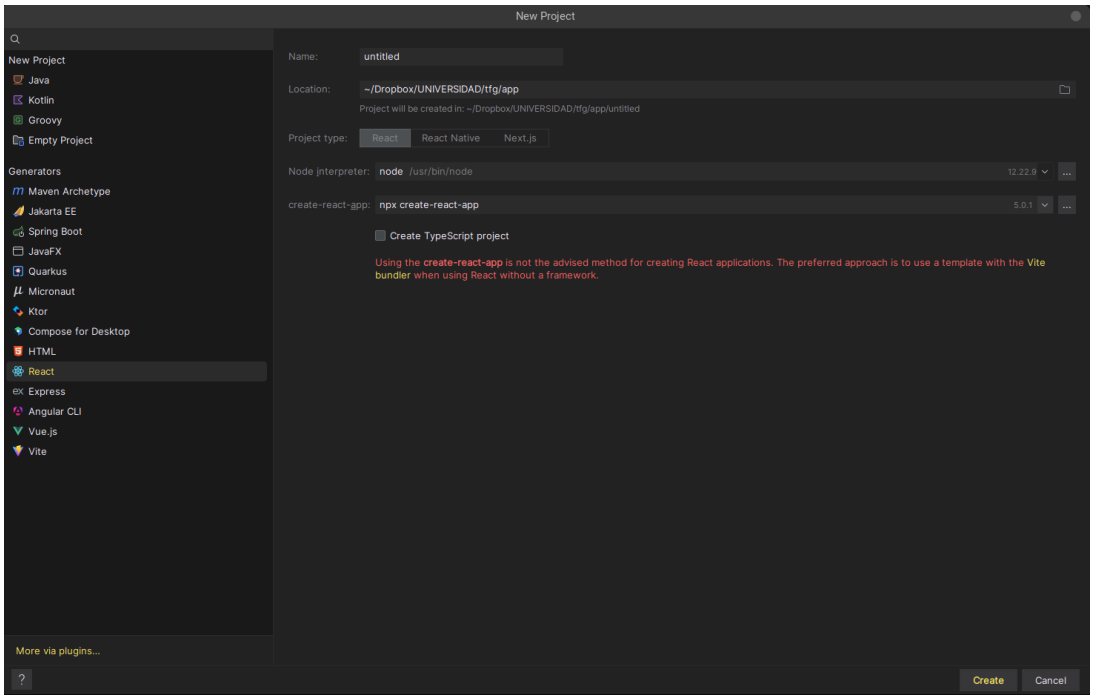


Figura 4.25.: Creación de app react con IntellIJ idea

- **Conexión de la Aplicación con la API**:

  Integré la aplicación React con la API desarrollada en FastAPI para obtener los datos almacenados. Utilicé el archivo 'App.js' para definir la lógica que conecta con la API y trae los datos. Mediante el uso de 'fetchData' y el hook 'useEffect()', se envían solicitudes a los endpoints de la API (/documents/health_data/aw_data y /documents/health_data/fitbit_data). Los datos recibidos son almacenados en los estados 'awData' y 'fitbitData', permitiendo su uso para generar visualizaciones y análisis en la

aplicación. Además, el componente *FilterDataPage* se diseñó para facilitar la gestión y visualización de los datos filtrados, lo cual es clave para cumplir con los requisitos del proyecto

```
import React, { useEffect, useState } from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-
    router-dom';
import './styles/App.css';
import ActivityChart from './components/ActivityChart';
import FilteredDataPage from './pages/FilterDataPage';
import { fetchData } from './api';


function App() {
    // Estados para almacenar todos los datos de Apple Watch y Fitbit
    const [awData, setAwData] = useState({ documents: [] });
    const [fitbitData, setFitbitData] = useState({ documents: [] });
    // Obtener datos de la API
    useEffect(() => {
        async function loadData() {
            const awData = await fetchData('http://127.0.0.1:8000/
                documents/health_data/aw_data');
            setAwData(awData);

            const fitbitData = await fetchData('http://127.0.0.1:8000/
                documents/health_data/fitbit_data');
            setFitbitData(fitbitData);


        }

        loadData();
    }, []);

    return (
        <Router>
            <div className="App">
                <h1>Datos de Salud - Comparaciones de Relojes
                    Inteligentes</h1>
                <nav>
                    <Link to="/">Inicio</Link>  <Link
                        to=/filtered-data»Datos
                        Filtrados</Link></nav></div></Router>);export default
                        App;
```

Código 4.5: Primera conexión de la aplicación con la api

- **Desarrollo de Filtros para Buscar por Dispositivo, Edad y Actividad**:

  Para permitir la búsqueda específica, desarrollé filtros personalizados que permiten al administrador buscar datos de los usuarios según dispositivo, edad y actividad. Estos filtros se implementaron como componentes de React reutilizables. Aseguré que los filtros fueran intuitivos y fáciles de usar, proporcionando un formulario claro para la selección de los criterios de búsqueda.

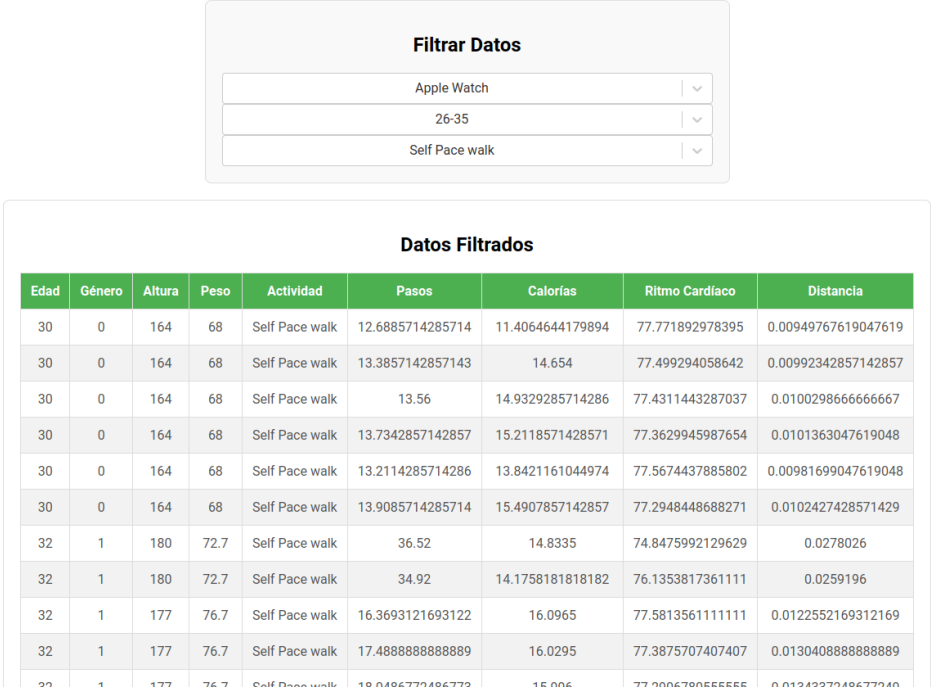
Un primer vistazo de como sería esta página y muestra de su correcto funcionamiento.

```
import React from 'react';
import FilterSection from '../components/FilterSection';
import FilteredDataTable from '../components/FilteredDataTable';
import '../styles/FilterDataPage.css';

/**
 * Página para mostrar los datos filtrados.
 *
 * @component
 * @param {Object} props - Props del componente.
 * @param {Object} props.awData - Datos del Apple Watch.
 * @param {Object} props.fitbitData - Datos del Fitbit.
 * @param {Function} props.setSelectedDevice - Función para actualizar el dispositivo seleccionado.
 * @param {Function} props.setSelectedAge - Función para actualizar el rango de edad seleccionado.
 * @param {Function} props.setSelectedActivity - Función para actualizar la actividad seleccionada.
 * @param {Array} props.filteredData - Datos filtrados que se mostrarán en la tabla.
 * @returns {JSX.Element} El componente de la página de datos filtrados.
 */
const FilteredDataPage = ({ awData, fitbitData, setSelectedDevice, setSelectedAge, setSelectedActivity, filteredData }) => {
    return (
        <div className="filtered-data-page">
            <div className="filter-section-container">
                <h2>Filtrar Datos</h2>
                <FilterSection
                    awData={awData}
                    setSelectedDevice={setSelectedDevice}
                    setSelectedAge={setSelectedAge}
                    setSelectedActivity={setSelectedActivity}
                />
            </div>
            <FilteredDataTable filteredData={filteredData} />
        </div>
    );
};

export default FilteredDataPage;
```

Código 4.6: Filtros de datos



Edad	Género	Altura	Peso	Actividad	Pasos	Calorías	Ritmo Cardíaco	Distancia
30	0	164	68	Self Pace walk	12.6885714285714	11.4064644179894	77.771892978395	0.00949767619047619
30	0	164	68	Self Pace walk	13.3857142857143	14.654	77.499294058642	0.00992342857142857
30	0	164	68	Self Pace walk	13.56	14.9329285714286	77.4311443287037	0.0100298666666667
30	0	164	68	Self Pace walk	13.7342857142857	15.2118571428571	77.3629945987654	0.0101363047619048
30	0	164	68	Self Pace walk	13.2114285714286	13.8421161044974	77.5674437885802	0.00981699047619048
30	0	164	68	Self Pace walk	13.9085714285714	15.4907857142857	77.2948448688271	0.0102427428571429
32	1	180	72.7	Self Pace walk	36.52	14.8335	74.8475992129629	0.0278026
32	1	180	72.7	Self Pace walk	34.92	14.1758181818182	76.1353817361111	0.0259196
32	1	177	76.7	Self Pace walk	16.3693121693122	16.0965	77.5813561111111	0.0122552169312169
32	1	177	76.7	Self Pace walk	17.4888888888889	16.0295	77.3875707407407	0.0130408888888889

Figura 4.26.: Muestra funcionamiento de filtro de datos

```
import React from 'react';
import '../styles/FilteredDataTable.css';

/**
 * Componente para mostrar los datos filtrados en formato de tabla.
 *
 * @component
 * @param {Object} props - Props del componente.
 * @param {Array} props.filteredData - Datos filtrados que se mostrará
    n en la tabla.
 * @returns {JSX.Element} El componente de la tabla de datos filtrados
    .
 */
const FilteredDataTable = ({ filteredData }) => {
    return (
        <div className="filtered-data">
            <h2>Datos Filtrados</h2>
            {filteredData.length > 0 ? (
                <table>
                    <thead>
                    <tr>
                        <th>Edad</th>
                        <th>Género</th>
                        <th>Altura</th>
```

```
                        <th>Peso</th>
                        <th>Actividad</th>
                        <th>Pasos</th>
                        <th>Calorías</th>
                        <th>Ritmo Cardíaco</th>
                        <th>Distancia</th>
                    </tr>
                    </thead>
                    <tbody>
                    {filteredData.map((doc, index) => (
                        <tr key={index}>
                            <td>{doc.age}</td>
                            <td>{doc.gender}</td>
                            <td>{doc.height}</td>
                            <td>{doc.weight}</td>
                            <td>{doc.activity_trimmed}</td>
                            <td>{doc.Applewatch ? doc.Applewatch.
                                Steps_LE : doc.Fitbit ? doc.Fitbit.
                                Steps_LE : 'N/A'}</td>
                            <td>{doc.Applewatch ? doc.Applewatch.
                                Calories_LE : doc.Fitbit ? doc.Fitbit.
                                Calories_LE : 'N/A'}</td>
                            <td>{doc.Applewatch ? doc.Applewatch.
                                Heart_LE : doc.Fitbit ? doc.Fitbit.
                                Heart_LE : 'N/A'}</td>
                            <td>{doc.Applewatch ? doc.Applewatch.
                                Distance_LE : doc.Fitbit ? doc.Fitbit.
                                Distance_LE : 'N/A'}</td>
                        </tr>
                    ))}
                    </tbody>
                </table>
            ) : (
                <p>No hay datos disponibles según los filtros
                    seleccionados.</p>
            )}
        </div>
    );
};

export default FilteredDataTable;
```

Código 4.7: Código de tabla de datos filtrados

- **Implementación de Lógica para Excluir Campos Innecesarios en los Resultados**:

  Implementé lógica adicional para limpiar los resultados obtenidos, de modo que se excluyan los identificadores únicos (como '_id') y otros campos no relevantes para la visualización. Para lograr esto, realicé modificaciones en los endpoints de la API, especificando qué campos deben ser excluidos al enviar la respuesta.

```python
# Obtiene todos los documentos de una colección especificada, excluyendo el campo "_id"
def get_all_documents(client, db_name, collection_name):  4 usages
    try:
        # Selecciona la base de datos por nombre
        db = client[db_name]
        # Selecciona la colección por nombre
        collection = db[collection_name]
        # Define una proyección para excluir el campo "_id"
        projection = {"_id": 0}
        # Obtiene todos los documentos de la colección aplicando la proyección
        documents = list(collection.find({}, projection))
        return documents
    except Exception as e:
        # Imprime un mensaje de error si no se pudieron obtener los documentos
        print("Error al obtener los documentos: %s" % e)
        return []
```

Figura 4.27.: Lógica para exclusión de campo _id en la consulta

- **Garantizar el Correcto Funcionamiento de los Filtros**:

  Realicé diversas pruebas manuales para asegurarme de que los filtros funcionaran correctamente. También probé la aplicación en diferentes navegadores para asegurar la compatibilidad y usabilidad de los filtros.

  Comparé los datos obtenidos aplicando los filtros en la aplicación react y aplicando los filtros sobre la propia base de datos de mongodb cloud.

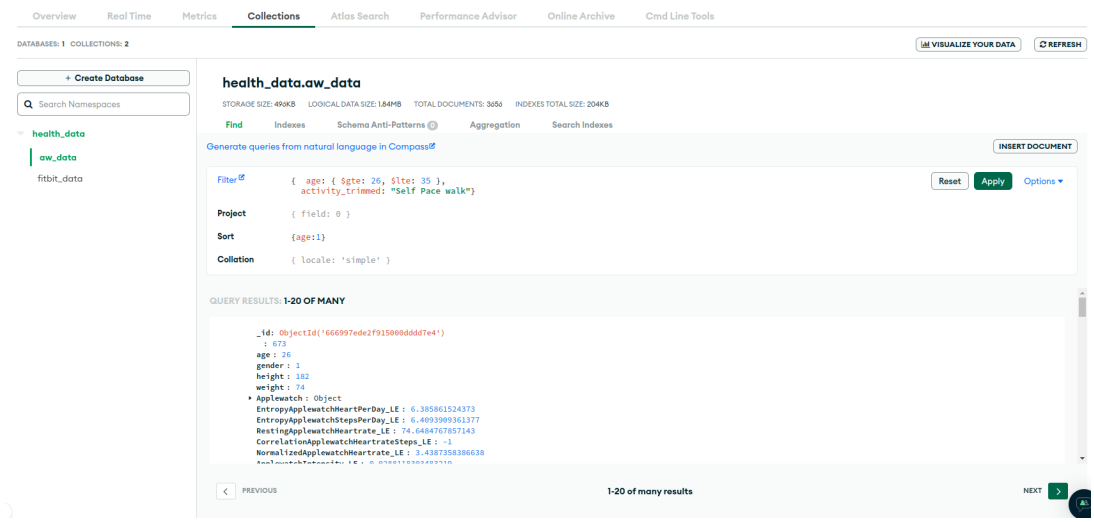


Figura 4.28.: Prueba de correcto funcionamiento de filtros

4.3.4.7. HU. 3 - Visualización de Datos de Salud

- Desarrollo de componentes para mostrar el total de pasos y calorías quemadas por cada tipo de ejercicio.

  Primero de todo, declaré la estructura de datos donde se iban a guardar los summary. Estos iban a contener los datos del total de pasos y calorías quemadas por cada tipo de actividad en cada tipo de reloj.

  En el siguiente código se puede ver la lógica aplicada para conseguir rellenar esta estructura de datos.

```
320 const [summary, setSummary] = useState({ applewatch: {}, fitbit: {} })
        ;
321
322 // Calcular el resumen de los datos para el Apple Watch y Fitbit
323 useEffect(() => {
324     if (awData.documents.length > 0) {
325         // Calcular el resumen para el Apple Watch
326         const summaryAW = awData.documents.reduce((acc, doc) => {
327             const activity = doc.activity_trimmed;
328             if (!acc[activity]) {
329                 acc[activity] = { steps: 0, calories: 0 };
330             }
331             acc[activity].steps += doc.Applewatch.Steps_LE;
332             acc[activity].calories += doc.Applewatch.Calories_LE;
333             return acc;
334         }, {});
335         setSummary(prevSummary => ({ ...prevSummary, applewatch:
                summaryAW }));
336     }
337
338     if (fitbitData.documents.length > 0) {
339         // Calcular el resumen para el Fitbit
340         const summaryFitbit = fitbitData.documents.reduce((acc, doc)
                => {
341             const activity = doc.activity_trimmed;
342             if (!acc[activity]) {
343                 acc[activity] = { steps: 0, calories: 0 };
344             }
345             acc[activity].steps += doc.Fitbit.Steps_LE;
346             acc[activity].calories += doc.Fitbit.Calories_LE;
347             return acc;
348         }, {});
349         setSummary(prevSummary => ({ ...prevSummary, fitbit:
                summaryFitbit }));
350     }
351 }, [awData, fitbitData]);
```

Código 4.8: Cálculo de los resumenes de actividad

- Implementación de la interfaz de usuario para visualizar un resumen de los datos de actividad física y salud.

Para la visualización de estos datos, he creado un componente en react llamado SummariesPage. Muestro el código a continuación, bien documentado donde se entiende que hace cada línea de código, que trata de mostrar estos summaries previamente generados para ser visualizados por el usuario y completar el objetivo del HU3.

Todos los estilos han sido definidos en los css correspondientes.

```
352 import React from 'react';
353 import '../styles/SummariesPage.css';
354
355 /**
356  * Componente para mostrar el resumen de los datos del Apple Watch y
        Fitbit.
357  *
358  * @component
359  * @param {Object} props - Props del componente.
360  * @param {Object} props.summary - Resumen de los datos de Apple Watch
        y Fitbit.
361  * @returns {JSX.Element} El componente de la página de resúmenes.
362  */
363 const SummariesPage = ({ summary }) => {
364     return (
365         <div className="summary-section">
366             {/* Sección para mostrar los resúmenes tanto de Apple
                Watch como de Fitbit */}
367
368             <div className="summary-column">
369                 {/* Columna que contiene el resumen de los datos del
                    Apple Watch */}
370                 <h2>Resumen de Datos del Apple Watch</h2>
371                 {Object.keys(summary.applewatch).length > 0 ? (
372                     // Si hay datos disponibles, mostrarlos
373                     <div className="summary-container">
374                         {Object.entries(summary.applewatch).map(([
                            activity, data]) => (
375                             // Iterar sobre cada actividad y mostrar
                                los datos correspondientes
376                             <div key={activity} className="summary-
                                item">
377                                 <h3>{activity}</h3>
378                                 {/* Mostrar la cantidad total de pasos
                                     */}
379                                 <p>Pasos Totales: {data.steps.toFixed
                                    (2)}</p>
380                                 {/* Mostrar la cantidad total de calor
                                    ías quemadas */}
381                                 <p>Calorías Quemadas: {data.calories.
                                    toFixed(2)}</p>
382                             </div>
383                         ))}
384                     </div>
385                 ) : (
386                     // Mostrar mensaje mientras se cargan los datos
```

```
                    <p>Cargando resumen del Apple Watch...</p>
                )}
            </div>

            <div className="summary-column">
                {/* Columna que contiene el resumen de los datos del
                    Fitbit */}
                <h2>Resumen de Datos del Fitbit</h2>
                {Object.keys(summary.fitbit).length > 0 ? (
                    // Si hay datos disponibles, mostrarlos
                    <div className="summary-container">
                        {Object.entries(summary.fitbit).map(([activity
                            , data]) => (
                            // Iterar sobre cada actividad y mostrar
                                los datos correspondientes
                            <div key={activity} className="summary-
                                item">
                                <h3>{activity}</h3>
                                {/* Mostrar la cantidad total de pasos
                                     */}
                                <p>Pasos Totales: {data.steps.toFixed
                                    (2)}</p>
                                {/* Mostrar la cantidad total de calor
                                    ías quemadas */}
                                <p>Calorías Quemadas: {data.calories.
                                    toFixed(2)}</p>
                            </div>
                        ))}
                    </div>
                ) : (
                    // Mostrar mensaje mientras se cargan los datos
                    <p>Cargando resumen del Fitbit...</p>
                )}
            </div>
        </div>
    );
};

export default SummariesPage;
```

Código 4.9: Muestra de los summaries

Figura 4.29.: Visualización de summaries en la app

4.3.4.8. Resumen del Sprint

Historia	Estimación	Prioridad
HU. 1	3	1
HU. 3	2	2

Tabla 4.31.: Historias del Sprint 04

- **Número Total de Tareas**: 6
- **Estimación Total**: 5

4.3.5. Sprint 5

4.3.5.1. Objetivos del Sprint

El objetivo del Sprint 05 es implementar funcionalidades avanzadas de visualización para permitir que los usuarios puedan ver gráficas sobre la frecuencia cardíaca y comparar los datos entre diferentes dispositivos. Esto contribuirá a una mejor comprensión del rendimiento de los dispositivos y de la actividad física de los usuarios.

4.3.5.2. Historias a Desarrollar

4.3.5.3. HU. 4 - Visualización de Gráficas

ID	**HU. 4**
Nombre	**Visualización de Gráficas**
Descripción	*Como usuario, quiero ver gráficas de la frecuencia cardíaca a lo largo de las medidas, para evaluar el impacto de los entrenamientos en la salud.*
Criterios de Aceptación	▪ Gráficos para mostrar las calorías gastadas por tipo de ejercicio. ▪ Posibilidad de seleccionar diferentes tipos de ejercicios. ▪ Mostrar los gráficos los datos de ambos relojes, de distinto color para poder distinguirlos.
Estimación	4
Prioridad	1

Tabla 4.32.: Historia de Usuario HU. 4

4.3.5.4. HU. 5 - Comparación entre Dispositivos

ID	**HU. 5**
Nombre	**Comparación entre Dispositivos**
Descripción	*Quiero comparar los datos recogidos por diferentes dispositivos, para entender las diferencias en las mediciones.*
Criterios de Aceptación	▪ Comparación de datos visualizada lado a lado.
Estimación	4
Prioridad	3

Tabla 4.33.: Historia de Usuario HU. 5

4.3.5.5. Descomposición en Tareas

Para cumplir con los objetivos y completar las historias seleccionadas, se han descompuesto en las siguientes tareas de desarrollo e implementación:

4.3.5.6. HU. 4 - Visualización de Gráficas

- **Implementación de Componentes Gráficos en React**:

  Para la visualización de las calorías gastadas, se utilizó la librería **react-google-charts**, particularmente para la generación de un *ComboChart* que muestra las calorías gastadas por actividad. Se desarrolló un componente genérico llamado **ChartComponent** (`ChartComponent.js`) que permite reutilizar la lógica de renderizado de gráficas en diferentes partes de la aplicación.

```
418 import React from 'react';
419 import { Chart } from 'react-google-charts';
420
421 /**
422  * Componente genérico para renderizar una gráfica usando react-google
        -charts.
423  *
424  * @component
425  * @param {Object} props - Props del componente.
426  * @param {string} props.title - El título de la gráfica.
427  * @param {Array} props.data - Los datos que se mostrarán en la grá
        fica.
428  * @param {Object} props.options - Las opciones de configuración para
        la gráfica.
429  * @returns {JSX.Element} El componente Chart que renderiza la gráfica
        .
430  */
431 const ChartComponent = ({ title, data, options }) => {
432     return (
433         <Chart
434             chartType="ComboChart"
435             width="100%"
436             height="400px"
437             data={data}
438             options={{ ...options, title }}
439         />
440     );
441 };
442
443 export default ChartComponent;
```

Código 4.10: Chart Component

- **Desarrollo de Lógica para Seleccionar Diferentes Tipos de Ejercicios**:

  Se creó el componente **ActivityChart** (`ActivityChart.js`), que permite a los usuarios seleccionar una actividad específica mediante un selector (**react-select**). Al elegir una actividad, los datos de Apple Watch y Fitbit se filtran según la actividad seleccionada,

y se muestra una gráfica comparativa que presenta las calorías gastadas en dicha actividad. La lógica de filtrado se implementó mediante el uso del hook **useEffect**, que reacciona a los cambios en la actividad seleccionada y actualiza los datos de la gráfica.

```
444 import React, { useEffect, useState } from 'react';
445 import Select from 'react-select';
446 import ChartComponent from './ChartComponent';
447
448 /**
449  * Componente para renderizar una gráfica de calorías según la
        actividad seleccionada para Apple Watch y Fitbit.
450  *
451  * @component
452  * @param {Object} props - Props del componente.
453  * @param {Object} props.awData - Datos del Apple Watch.
454  * @param {Object} props.fitbitData - Datos del Fitbit.
455  * @param {Array} props.awData.documents - Array de documentos con
        actividades y datos del Apple Watch.
456  * @param {Array} props.fitbitData.documents - Array de documentos con
        actividades y datos del Fitbit.
457  * @returns {JSX.Element} El componente ActivityChart que renderiza la
        gráfica y un selector de actividad.
458  */
459 const ActivityChart = ({ awData, fitbitData }) => {
460     const [chartData, setChartData] = useState([['Activity', 'Apple
            Watch Calories', 'Fitbit Calories']]);
461     const [selectedActivity, setSelectedActivity] = useState(null);
462
463     // Actualiza los datos de la gráfica según la actividad
            seleccionada
464     useEffect(() => {
465         if ((awData.documents.length > 0  fitbitData.documents.length
                > 0) && selectedActivity) {
466             const filteredAwData = awData.documents.filter(doc => doc.
                    activity_trimmed ===
467                 selectedActivity.value);
468             const formattedAwData = filteredAwData.map(doc => ['', doc
                    .Applewatch.Calories_LE, null]);
469
470             const filteredFitbitData = fitbitData.documents.filter(doc
                     => doc.activity_trimmed ===
471                 selectedActivity.value);
472             const formattedFitbitData = filteredFitbitData.map(doc =>
                    ['', null, doc.Fitbit.Calories_LE]);
473
474             const combinedData = [...formattedAwData, ...
                    formattedFitbitData];
475
476             setChartData([['Activity', 'Apple Watch Calories', 'Fitbit
                     Calories'], ...combinedData]);
477         }
478     }, [awData, fitbitData, selectedActivity]);
```

```
479
480     // Opciones de actividades para el selector
481     const activityOptions = [
482         ...new Set([...awData.documents, ...fitbitData.documents].map(
                doc => doc.activity_trimmed))
483     ].map(activity => ({
484         value: activity,
485         label: activity,
486     }));
487
488     // Opciones de configuración para la gráfica
489     const options = {
490         hAxis: { title: '' },
491         vAxis: { title: 'Calories' },
492             seriesType: 'bars',
493         legend: { position: 'bottom' },
494     };
495
496     return (
497         <div>
498             <h2>Selecciona la actividad:</h2>
499             <Select
500                 options={activityOptions}
501                 onChange={setSelectedActivity}
502                 placeholder="Selecciona una actividad"
503             />
504             {(awData.documents.length > 0  fitbitData.documents.length
                     > 0) && selectedActivity ? (
505                 <ChartComponent title="Calorías por Actividad - Apple
                        Watch y Fitbit" data={chartData}
506                                 options={options} />
507             ) : (
508                 <p>Cargando datos o selecciona una actividad...</p>
509             )}
510         </div>
511     );
512 };
513
514 export default ActivityChart;
```

Código 4.11: Activity Chart

- **Diferenciación de Datos de Ambos Relojes mediante Colores en los Gráficos**:

  Para diferenciar visualmente los datos de Apple Watch y Fitbit, se configuraron las series del *ComboChart* con distintos colores. En el componente `ActivityChart.js`, se utiliza la opción **seriesType** para definir el tipo de gráfico y la opción **legend** para asegurar que los usuarios puedan distinguir claramente entre ambos dispositivos. Los datos se presentan en forma de barras, cada una de un color diferente.

- **Pruebas para Garantizar la Correcta Visualización de las Gráficas**:

Finalmente, se realizaron pruebas para asegurar que las gráficas se muestren correctamente y que la información visualizada sea precisa y fácil de interpretar. Estas pruebas se llevaron a cabo en la página de gráficas (`GraphsPage.js`), donde se integró el componente **ActivityChart**. Se verificó el correcto funcionamiento de los filtros, la visualización de los datos y la interacción del usuario con los gráficos.

Figura 4.30.: Visualización de gráfica de la actividad Self Pace Walk

4.3.5.7. HU. 5 - Comparación entre Dispositivos

En esta sección, se desarrolló la funcionalidad para comparar los datos de los dispositivos **Apple Watch** y **Fitbit** de manera visual, mostrando las diferencias lado a lado.

- **Implementación de la Funcionalidad para Comparar los Datos**:

  Para esta funcionalidad, se desarrolló la página de comparación (`ComparisonPage.js`), la cual permite visualizar las diferencias entre los datos recolectados por Apple Watch y Fitbit. Se calcularon promedios de calorías quemadas, pasos dados, ritmo cardíaco y distancia recorrida para cada tipo de actividad registrada por los dispositivos.

  Los datos se obtienen a partir de las colecciones almacenadas en MongoDB, y se calcula el promedio de cada métrica por actividad. Para cada dispositivo, los datos se filtran utilizando la función filter(), y luego se utilizan funciones como reduce() para sumar los valores y calcular los promedios. Esto permite mostrar al usuario una comparación directa entre los datos de ambos dispositivos, resaltando las diferencias y similitudes.

```
515 import React from 'react';
516 import '../styles/ComparisonPage.css';
517
518 /**
519  * Página para comparar los datos de los dispositivos Apple Watch y
        Fitbit de manera visual.
520  *
521  * @component
522  * @param {Object} props - Props del componente.
523  * @param {Object} props.awData - Datos del Apple Watch.
524  * @param {Object} props.fitbitData - Datos del Fitbit.
525  * @returns {JSX.Element} El componente de la página de comparación.
```

```
 */
const ComparisonPage = ({ awData, fitbitData }) => {
    // Obtener todas las actividades únicas de ambos dispositivos
    const activities = [...new Set([...awData.documents, ...fitbitData
        .documents].map(doc => doc.activity_trimmed))];

    // Calcular los promedios de calorías, pasos, ritmo cardíaco, y
        distancia por tipo de actividad para Apple Watch
    const awAverages = activities.map(activity => {
        const filteredDocs = awData.documents.filter(doc => doc.
            activity_trimmed === activity);
        const totalCalories = filteredDocs.reduce((sum, doc) => sum +
            doc.Applewatch.Calories_LE, 0);
        const totalSteps = filteredDocs.reduce((sum, doc) => sum + doc
            .Applewatch.Steps_LE, 0);
        const totalHeartRate = filteredDocs.reduce((sum, doc) => sum +
             doc.Applewatch.Heart_LE, 0);
        const totalDistance = filteredDocs.reduce((sum, doc) => sum +
            doc.Applewatch.Distance_LE, 0);
        const count = filteredDocs.length;
        return {
            activity,
            avgCalories: count > 0 ? (totalCalories / count).toFixed
                (2) : 'N/A',
            avgSteps: count > 0 ? (totalSteps / count).toFixed(2) : 'N
                /A',
            avgHeartRate: count > 0 ? (totalHeartRate / count).toFixed
                (2) : 'N/A',
            avgDistance: count > 0 ? (totalDistance / count).toFixed
                (2) : 'N/A'
        };
    });

    // Calcular los promedios de calorías, pasos, ritmo cardíaco, y
        distancia por tipo de actividad para Fitbit
    const fitbitAverages = activities.map(activity => {
        const filteredDocs = fitbitData.documents.filter(doc => doc.
            activity_trimmed === activity);
        const totalCalories = filteredDocs.reduce((sum, doc) => sum +
            doc.Fitbit.Calories_LE, 0);
        const totalSteps = filteredDocs.reduce((sum, doc) => sum + doc
            .Fitbit.Steps_LE, 0);
        const totalHeartRate = filteredDocs.reduce((sum, doc) => sum +
             doc.Fitbit.Heart_LE, 0);
        const totalDistance = filteredDocs.reduce((sum, doc) => sum +
            doc.Fitbit.Distance_LE, 0);
        const count = filteredDocs.length;
        return {
            activity,
            avgCalories: count > 0 ? (totalCalories / count).toFixed
                (2) : 'N/A',
            avgSteps: count > 0 ? (totalSteps / count).toFixed(2) : 'N
```

```
                /A',
560             avgHeartRate: count > 0 ? (totalHeartRate / count).toFixed
                    (2) : 'N/A',
561             avgDistance: count > 0 ? (totalDistance / count).toFixed
                    (2) : 'N/A'
562         };
563     });
```

Código 4.12: Código de comparación

- **Desarrollo de la Interfaz para Mostrar los Datos de Manera Clara y Organizada**:

  La interfaz de usuario de la página de comparación presenta los datos en forma de tabla, permitiendo una fácil interpretación. Cada fila de la tabla muestra una actividad y las métricas correspondientes para ambos dispositivos. Las métricas incluidas son:

  - **Promedio de Calorías**: Comparación entre las calorías quemadas según el Apple Watch y el Fitbit.
  - **Promedio de Pasos**: Comparación de los pasos realizados en cada actividad.
  - **Promedio de Ritmo Cardíaco**: Comparación de la frecuencia cardíaca promedio medida durante la actividad.
  - **Promedio de Distancia**: Distancia recorrida durante la actividad.

  Se desarrollaron funciones específicas para calcular y mostrar los promedios de cada métrica de manera precisa. Para hacer esto visualmente claro, los datos de ambos dispositivos se presentan en columnas diferentes, permitiendo una comparación lado a lado. La interfaz fue diseñada para ser simple e intuitiva, ayudando a los usuarios a interpretar rápidamente las diferencias entre los dispositivos.

```
565     return (
566         <div className="comparison-page">
567             <h2>Comparación de Datos de Apple Watch y Fitbit</h2>
568             <div className="comparison-table">
569                 <table>
570                     <thead>
571                     <tr>
572                         <th>Actividad</th>
573                         <th>Apple Watch - Promedio de Calorías</th>
574                         <th>Fitbit - Promedio de Calorías</th>
575                         <th>Apple Watch - Promedio de Pasos</th>
576                         <th>Fitbit - Promedio de Pasos</th>
577                         <th>Apple Watch - Promedio de Ritmo Cardíaco</
                                th>
578                         <th>Fitbit - Promedio de Ritmo Cardíaco</th>
579                         <th>Apple Watch - Promedio de Distancia</th>
580                         <th>Fitbit - Promedio de Distancia</th>
581                     </tr>
582                     </thead>
583                     <tbody>
584                     {activities.map((activity, index) => (
585                         <tr key={activity}>
586                             <td>{activity}</td>
```

```
                                <td>{awAverages[index].avgCalories}</td>
                                <td>{fitbitAverages[index].avgCalories}</
                                    td>
                                <td>{awAverages[index].avgSteps}</td>
                                <td>{fitbitAverages[index].avgSteps}</td>
                                <td>{awAverages[index].avgHeartRate}</td>
                                <td>{fitbitAverages[index].avgHeartRate}</
                                    td>
                                <td>{awAverages[index].avgDistance}</td>
                                <td>{fitbitAverages[index].avgDistance}</
                                    td>
                            </tr>
                        ))}
                        </tbody>
                    </table>
                </div>
            </div>
        );
};

export default ComparisonPage;
```

Código 4.13: Código de comparación

A continuación, se muestra un ejemplo de la tabla desarrollada para comparar los datos:

Comparación de Datos de Apple Watch y Fitbit

Actividad	Apple Watch - Promedio de Calorías	Fitbit - Promedio de Calorías	Apple Watch - Promedio de Pasos	Fitbit - Promedio de Pasos	Apple Watch - Promedio de Ritmo Cardíaco	Fitbit - Promedio de Ritmo Cardíaco	Apple Watch - Promedio de Distancia	Fitbit - Promedio de Distancia
Lying	6.15	10.28	161.56	10.84	80.31	78.16	0.09	27.24
Sitting	5.38	20.50	216.13	9.83	84.85	76.23	0.11	33.13
Self Pace walk	7.62	57.92	125.99	14.90	85.66	78.05	0.06	37.33
Running 3 METs	7.57	59.17	148.87	12.41	86.83	82.55	0.06	31.95
Running 5 METs	5.72	67.35	180.15	8.92	98.07	80.58	0.08	33.89
Running 7 METs	2.48	33.74	248.13	7.13	113.31	78.73	0.10	37.23

Figura 4.31.: Tabla comparativa de los datos de Apple Watch y Fitbit

4.3.5.8. Resumen del Sprint

Historia	Estimación	Prioridad
HU. 4	4	1
HU. 5	4	3

Tabla 4.34.: Historias del Sprint 05

- **Número Total de Tareas**: 6
- **Estimación Total**: 8

4.3.6. Sprint 6

4.3.6.1. Objetivos del Sprint

El objetivo del Sprint 06 es proporcionar la capacidad a los usuarios de exportar los datos de actividad física filtrados en formato CSV. Esto permitirá a los usuarios analizar los datos fuera de la aplicación y utilizarlos con diversas finalidades, como estudios personales o investigaciones.

4.3.6.2. Historia a Desarrollar

4.3.6.3. HU. 6 - Filtrado de Datos de Salud

ID	**HU. 6**
Nombre	**Exportación de datos filtrados en formato CSV**
Descripción	*Como usuario final, quiero exportar los datos de actividad física filtrados para poder tener acceso a ellos para diversas finalidades.*
Criterios de Aceptación	▪ Posibilidad de exportar los datos filtrados.
Estimación	4
Prioridad	1

Tabla 4.35.: Historia de Usuario HU. 6

4.3.6.4. Descomposición en Tareas

Para cumplir con el objetivo y completar la historia seleccionada, se han descompuesto en las siguientes tareas de desarrollo e implementación:

4.3.6.5. HU. 6 - Exportación de Datos Filtrados en Formato CSV

- **Generación de Datos Filtrados**:

  Una vez que los usuarios hayan aplicado los filtros deseados, se generará una vista previa de los datos filtrados. Esto fue realizado en el sprint 4.

- **Exportación de Datos en Formato CSV**:

  Se implementó un botón de exportación que permite a los usuarios descargar los datos filtrados en un archivo CSV. Para esto, se utilizó la librería react-csv en el frontend para convertir los datos a formato CSV y proporcionar la opción de descarga al usuario. Esta funcionalidad se integró directamente en el componente de visualización de datos filtrados.

```
605     // Configuración para exportar los datos en CSV
606     const headers = [
607         { label: 'Edad', key: 'age' },
608         { label: 'Género', key: 'gender' },
609         { label: 'Altura', key: 'height' },
610         { label: 'Peso', key: 'weight' },
```

```
        { label: 'Actividad', key: 'activity_trimmed' },
        { label: 'Pasos', key: 'steps' },
        { label: 'Calorías', key: 'calories' },
        { label: 'Ritmo Cardíaco', key: 'heart_rate' },
        { label: 'Distancia', key: 'distance' }
    ];

    const csvData = filteredData.map(doc => ({
        age: doc.age,
        gender: doc.gender,
        height: doc.height,
        weight: doc.weight,
        activity_trimmed: doc.activity_trimmed,
        steps: doc.Applewatch ? doc.Applewatch.Steps_LE : doc.Fitbit ?
             doc.Fitbit.Steps_LE : 'N/A',
        calories: doc.Applewatch ? doc.Applewatch.Calories_LE : doc.
            Fitbit ? doc.Fitbit.Calories_LE : 'N/A',
        heart_rate: doc.Applewatch ? doc.Applewatch.Heart_LE : doc.
            Fitbit ? doc.Fitbit.Heart_LE : 'N/A',
        distance: doc.Applewatch ? doc.Applewatch.Distance_LE : doc.
            Fitbit ? doc.Fitbit.Distance_LE : 'N/A'
    }));

    return (
        <div className="filtered-data">
            <h2>Datos Filtrados</h2>
            {filteredData.length > 0 ? (
                <>
                    <CSVLink data={csvData} headers={headers} filename
                        ="datos_filtrados.csv" className="export-csv-
                        button">
                        Exportar Datos a CSV
                    </CSVLink>
```

Código 4.14: Código de creación de csv

- **Pruebas de la Exportación**:

  Se realizaron pruebas para asegurar que la funcionalidad de exportación funcione correctamente. Las pruebas incluyeron la verificación de que los filtros seleccionados se aplicaran correctamente, que los datos exportados correspondieran exactamente a los resultados filtrados mostrados en la aplicación y que el archivo CSV generado contuviera toda la información esperada con el formato adecuado.

Figura 4.32.: Botón de exportación para descargar los datos filtrados en formato CSV

4.3.6.6. Resumen del Sprint

Historia	Estimación	Prioridad
HU. 6	4	1

Tabla 4.36.: Historias del Sprint 06

- **Número Total de Tareas**: 3
- **Estimación Total**: 4

Conclusiones y trabajos futuros

El análisis comparativo entre los tres principales fabricantes de relojes inteligentes —**Apple**, **Garmin** y **Fitbit**— muestra que cada dispositivo tiene sus fortalezas y debilidades, así como enfoques diferenciados para el monitoreo de la salud y la actividad física. Estos relojes han revolucionado el mercado al ofrecer un acceso inmediato a información clave sobre nuestro bienestar, lo que permite a los usuarios tener un control más proactivo y personalizado de su salud. Sin embargo, cada dispositivo está dirigido a diferentes tipos de usuarios y ofrece características únicas que se alinean con necesidades específicas.

El **Apple Watch**, gracias a su integración con HealthKit, se ha posicionado como una herramienta poderosa para el monitoreo de la salud general. Entre sus puntos más fuertes se encuentran su capacidad para realizar electrocardiogramas (*ECG*) y su monitoreo de la exposición al ruido ambiental. Además, sus alertas de ritmo cardíaco irregular y la detección automática de caídas son características exclusivas que refuerzan su enfoque en la salud cardiovascular y la seguridad del usuario.

Sin embargo, el **Apple Watch** está más orientado hacia el bienestar general y la salud cotidiana que al seguimiento detallado de actividades deportivas extremas. A pesar de que ofrece métricas de actividad física básicas como el conteo de pasos y la frecuencia cardíaca, no se especializa en deportes específicos de la misma manera que Garmin. Además, su autonomía de batería, generalmente limitada a un día completo de uso, es un punto débil en comparación con otros dispositivos que pueden durar varios días sin necesidad de recarga.

Por otro lado, **Garmin** se ha consolidado como el dispositivo preferido para los entusiastas del deporte y los atletas. Una de sus principales fortalezas es su capacidad para ofrecer un seguimiento extremadamente detallado de una amplia gama de actividades deportivas, lo que lo convierte en la mejor opción para usuarios que desean datos precisos y profundos sobre su rendimiento físico en deportes como correr, ciclismo, natación o entrenamiento de fuerza.

Garmin también destaca por su métrica única de *Body Battery*, que permite a los usuarios gestionar mejor sus niveles de energía a lo largo del día, y su capacidad para medir la presión arterial en dispositivos compatibles. Estas características brindan una visión holística del estado físico del usuario y ayudan a planificar entrenamientos y momentos de descanso de manera más eficiente.

Sin embargo, **Garmin** puede ser percibido como menos accesible para usuarios que solo buscan monitorear su salud diaria y no tienen necesidades deportivas específicas. Aunque ofrece un seguimiento del sueño y la frecuencia cardíaca, su enfoque en deportes de alto rendimiento puede no ser relevante para todos los usuarios. Además, la complejidad de algunas de sus funciones puede resultar abrumadora para quienes buscan una experiencia más sencilla y directa.

Fitbit, por su parte, se ha posicionado como una opción excelente para aquellos usuarios que desean un equilibrio entre el seguimiento de la salud y la facilidad de uso. Sus puntos más fuertes radican en su capacidad para monitorear aspectos como la tasa de respiración, la temperatura de la piel y los niveles de glucosa, lo que lo convierte en una excelente opción para quienes buscan un control integral de su salud. Además, las funciones de registro de alimentos y la ingesta de agua son características únicas que facilitan el seguimiento de hábitos saludables y el balance energético.

Fitbit es también ideal para aquellos que buscan mejorar su salud a través del control de sus hábitos diarios sin enfocarse necesariamente en deportes de alto rendimiento. Su interfaz amigable y la simplicidad con la que presenta los datos lo hacen accesible a una amplia gama de usuarios, desde principiantes en el monitoreo de la salud hasta personas que requieren un control específico de su alimentación y hábitos de hidratación.

No obstante, **Fitbit** presenta algunas limitaciones en cuanto al seguimiento deportivo avanzado en comparación con **Garmin**. Aunque ofrece métricas básicas como la frecuencia cardíaca y el seguimiento de pasos, no está diseñado para proporcionar el nivel de detalle que demandan los atletas profesionales o los entusiastas de deportes específicos. Además, su capacidad para realizar mediciones avanzadas, como la presión arterial, es limitada en comparación con los otros dispositivos.

En términos generales, el **Apple Watch** se distingue por ser una opción potente para el monitoreo de la salud general y la seguridad del usuario, con características avanzadas centradas en la salud cardiovascular. Es ideal para aquellos que buscan un equilibrio entre un estilo de vida activo y la tranquilidad de recibir alertas de salud en tiempo real. No obstante, su menor duración de batería y su enfoque limitado en deportes lo hacen menos adecuado para atletas exigentes.

Garmin es la opción preferida para aquellos que buscan un análisis detallado de sus actividades deportivas. Su enfoque en deportes específicos y en métricas avanzadas como la *Body Battery* lo posiciona como el mejor dispositivo para deportistas serios. Sin embargo, su mayor complejidad y enfoque en deportes puede ser una barrera para usuarios más casuales.

Finalmente, **Fitbit** se destaca por su facilidad de uso y su enfoque en la mejora del bienestar diario a través de un control exhaustivo de la nutrición, la hidratación y parámetros clave como la tasa de respiración y la glucosa. Aunque no está tan orientado hacia el deporte de alto rendimiento como **Garmin**, su capacidad para proporcionar información integral sobre la salud diaria lo hace una opción accesible y eficaz para el usuario promedio.

En conclusión, los relojes inteligentes de **Apple**, **Garmin** y **Fitbit** ofrecen soluciones robustas para el monitoreo de la salud, cada uno con enfoques diferentes que responden a las necesidades específicas de sus usuarios. El **Apple Watch** es ideal para quienes buscan un dispositivo de monitoreo de salud integral, con una atención especial en la seguridad y la salud cardiovascular. **Garmin** se adapta mejor a atletas y entusiastas del deporte que requieren un seguimiento avanzado de sus entrenamientos y rendimiento físico, mientras que **Fitbit** es la opción más accesible para quienes desean un control exhaustivo de sus hábitos de vida y parámetros de salud, con un enfoque en la simplicidad y la facilidad de uso.

A medida que el mercado de los dispositivos portátiles continúa evolucionando, es probable que cada una de estas plataformas siga innovando en su oferta de características y funcionalidades. Elegir el reloj adecuado dependerá de las prioridades y necesidades individuales de cada usuario, ya sea que busquen optimizar su rendimiento deportivo, mejorar su bienestar diario o monitorizar de cerca su salud general.

Proyecto

A lo largo del desarrollo de este Trabajo de Fin de Grado, he tenido la oportunidad de enfrentarme a desafíos técnicos y de adquirir nuevas habilidades que han sido fundamentales para completar este proyecto. Aunque el objetivo principal de implementar un sistema avanzado de monitoreo de salud mediante dispositivos portátiles no se logró en su totalidad debido a las dificultades para acceder a datos directamente desde los wearables, el proyecto tomó un enfoque diferente que me permitió aprender valiosas lecciones en el ámbito de la gestión de datos y el desarrollo de software.

- **Desarrollo de un sistema avanzado de monitoreo de salud**: Aunque no pude obtener datos directamente de los dispositivos portátiles, el desarrollo de este sistema me llevó

a investigar, extraer y limpiar datos provenientes de diferentes fuentes, haciendo uso de conjuntos de datos disponibles públicamente. Esta experiencia fue enriquecedora, ya que tuve que adaptarme y aprender a trabajar con datos secundarios, aplicando técnicas de limpieza, normalización y preprocesamiento para garantizar la calidad de los datos utilizados en el proyecto.

- **Diseño e implementación de servicios web basados en SOA**: Logré implementar servicios web utilizando una arquitectura orientada a servicios (SOA), lo cual fue crucial para garantizar la modularidad y escalabilidad del sistema. El uso de FastAPI para el desarrollo del backend facilitó la creación de una API RESTful que permitió la recolección y procesamiento de datos de salud de forma eficiente. Cada servicio fue implementado de manera que pudiera ser reutilizable y fácilmente integrable, cumpliendo con los principios de SOA.

- **Almacenamiento de datos de salud**: Uno de los logros importantes fue diseñar y establecer un sistema de información para el almacenamiento de los datos de salud. Utilicé MongoDB como base de datos en la nube, lo que permitió almacenar la información de manera estructurada y garantizar su disponibilidad para el análisis. A través del uso de esta base de datos NoSQL, pude manejar los datos de manera eficiente, asegurando la escalabilidad y la flexibilidad necesarias para el tipo de información recolectada.

- **Provisión de una interfaz de usuario intuitiva**: Para cumplir con el objetivo de proporcionar una interfaz de usuario, utilicé React para desarrollar un frontend que permitiera a los usuarios visualizar los datos captados de manera intuitiva. Esta interfaz ofrece funcionalidades para filtrar y visualizar los datos en formato gráfico, facilitando la comprensión de la información. La posibilidad de comparar datos de diferentes dispositivos y de exportar los resultados en formatos útiles, como CSV, mejora la experiencia del usuario y permite que los datos sean utilizados con diferentes finalidades.

En general, el proyecto me permitió aplicar conocimientos adquiridos a lo largo de mi carrera, desde el desarrollo de APIs y la gestión de bases de datos, hasta el diseño de interfaces de usuario intuitivas. Pude poner en práctica metodologías ágiles, como Scrum, para la organización del trabajo en sprints, lo que me ayudó a planificar y ejecutar cada fase del desarrollo de forma ordenada y sistemática. Aunque enfrenté limitaciones en la obtención de datos directamente de los dispositivos wearables, la experiencia de adaptar el enfoque del proyecto y trabajar con conjuntos de datos disponibles fue igualmente desafiante y gratificante.

Trabajos Futuros

Existen varias áreas en las que este proyecto puede ser ampliado y mejorado en futuros trabajos:

- **Dashboard Personalizado**: Agregar un "Dashboard Personalizado"donde el usuario pueda configurar qué métricas quiere visualizar. Por ejemplo, permitir al usuario seleccionar si quiere ver solo datos de calorías y pasos, o incluir también otros parámetros como ritmo cardíaco y distancia. Esto proporcionará más flexibilidad y permitirá a los usuarios ajustar la aplicación a sus necesidades personales.

- **Comparativa Personalizada entre Usuarios**: Permitir que los usuarios ingresen manualmente los datos de otros miembros de la familia o amigos, o conectarse con sus amigos, para comparar estadísticas de actividad. Esto fomenta la competitividad amistosa, lo cual puede motivar a los usuarios a ser más activos.
- **Análisis de Tendencias y Predicciones**: Implementar una funcionalidad para analizar tendencias y hacer predicciones basadas en los datos históricos del usuario. Podría usar gráficos de línea para mostrar cómo evolucionan los pasos y las calorías a lo largo del tiempo, e incluso aplicar análisis de series temporales para predecir objetivos futuros. Esto proporciona una capa de análisis avanzada que puede ser útil para usuarios que buscan mejorar su condición física a largo plazo.

A. Apéndices

A.1. Conexión base de datos

En esta sección, se detalla la metodología utilizada para la conexión a la base de datos, así como las herramientas y tecnologías que facilitan la integración de datos en el sistema.

Para la implementación de la API, se utilizó **FastAPI**, un framework moderno de Python que destaca por su rendimiento y facilidad de uso. FastAPI permite crear API REST de manera rápida y eficiente, aprovechando el tipado estático de Python para generar **documentación automática**, como Swagger UI, lo cual facilita la interacción con los desarrolladores y usuarios finales. La estructura de la API está diseñada para ser modular y extensible, facilitando futuras ampliaciones y modificaciones sin afectar la funcionalidad existente.

Esta nos permite desacoplar la lógica de acceso a datos del resto de la aplicación, facilitando la mantenibilidad y la prueba del código. Además, se implementó el uso de **dependencias** en FastAPI para gestionar de manera eficiente la conexión a la base de datos, asegurando que cada solicitud tenga su propio contexto sin interferir con otras.

FastAPI ofrece también una integración sencilla con herramientas de validación y serialización de datos mediante **Pydantic** [4], lo cual garantiza que los datos recibidos y enviados a los clientes cumplan con los formatos esperados, minimizando errores y mejorando la calidad del código. Esto, junto con la capacidad de manejar operaciones de manera asincrónica, permite que la API tenga un rendimiento óptimo incluso bajo carga elevada.

La conexión se realiza al inicio de la aplicación, asegurando que todos los recursos necesarios estén disponibles antes de procesar cualquier solicitud. Este enfoque garantiza la disponibilidad y la eficiencia en el manejo de los datos, evitando problemas comunes relacionados con la concurrencia y maximizando la capacidad de respuesta del sistema.

En resumen, la elección de **FastAPI** está motivada por la necesidad de un sistema eficiente, fácil de escalar y mantener, que pueda manejar grandes volúmenes de solicitudes y proporcionar una interfaz clara y segura para los usuarios y desarrolladores.

A.1.1. Diseño API

El diseño de la API se basa en una arquitectura REST, con varios endpoints que permiten la gestión de los datos de las bases de datos MongoDB. A continuación se describen los principales endpoints, sus funciones y el código utilizado para su implementación de manera detallada:

- **/databases (GET)**: Obtiene una lista de todas las bases de datos disponibles en el cliente MongoDB. Utiliza la función extttlist_databases(client), que se conecta al cliente MongoDB y obtiene los nombres de todas las bases de datos. Este endpoint ayuda a

visualizar las bases de datos a las que el cliente tiene acceso. Si no se encuentran bases de datos, se devuelve un error 404.

- **/collections/{db_name} (GET)**: Lista todas las colecciones disponibles en una base de datos especificada por **db_name**. Utiliza la función extttlist_collections(client, db_name) que se conecta a la base de datos especificada y obtiene los nombres de las colecciones. Si no se encuentran colecciones, se devuelve un error 404.
- **/documents/{db_name}/{collection_name} (GET)**: Obtiene todos los documentos de una colección especificada en una base de datos determinada. Utiliza la función exttt-get_all_documents(client, db_name, collection_name) que se conecta a la base de datos y colección especificadas y devuelve todos los documentos excluyendo el campo _id. Si no se encuentran documentos, se devuelve un error 404.
- **/documents/{db_name}/{collection_name}/filter (GET)**: Obtiene documentos que cumplen con un filtro específico. Actualmente, se admite un filtro opcional por **age**. Utiliza la función extttget_documents_with_filter(client, db_name, collection_name, filter) que aplica un filtro a los documentos en la colección especificada. Si no se encuentran documentos con el filtro especificado, se devuelve un error 404.
- **/documents/{db_name}/{collection_name}/projection (GET)**: Obtiene documentos de una colección aplicando un filtro y una proyección especificados. La proyección define los campos que serán incluidos en la respuesta, como **age** y **gender**, excluyendo el campo _id. Utiliza la función extttget_documents_with_projection(client, db_name, collection_name, filter, projection) para aplicar tanto el filtro como la proyección. Si no se encuentran documentos que cumplan con el filtro y la proyección, se devuelve un error 404.

Cada uno de estos endpoints está desarrollado para proporcionar una funcionalidad específica de consulta sobre las bases de datos MongoDB, siguiendo el principio de responsabilidad única. Los comentarios incluidos en el código fuente proporcionan detalles adicionales sobre cómo cada endpoint gestiona los parámetros, las validaciones aplicadas, y los errores que podrían ocurrir. Las funciones en el fichero de utilidades han sido diseñadas para proporcionar acceso sencillo y eficiente a las operaciones de base de datos necesarias, con manejo adecuado de errores para asegurar la estabilidad de la API.

La API está diseñada para ser asincrónica, aprovechando las capacidades de FastAPI y las operaciones de MongoDB para asegurar una experiencia ágil incluso cuando se realizan múltiples consultas simultáneas. Esto permite mantener la eficiencia en la recuperación de datos y garantiza tiempos de respuesta óptimos.

A.1.2. Implementación API

A.1.2.1. Estructura del Proyecto

La API está organizada de manera modular para facilitar su mantenibilidad y escalabilidad. Los principales archivos y directorios del proyecto son:

- **main.py**: Archivo principal que inicia la aplicación FastAPI, define los endpoints y configura los middlewares.

- **mongo_utils.py**: Contiene las funciones de utilidad para interactuar con MongoDB, incluyendo la conexión, consultas y recuperación de documentos.
- **base_datos.py**: Script utilizado para probar el funcionamiento correcto de la conexión y las consultas a la base de datos.
- **config.py**: Archivo que almacena las variables de configuración, como la cadena de conexión a MongoDB.

Esta estructura permite desacoplar la lógica de negocio de la lógica de acceso a datos, facilitando futuras modificaciones y pruebas.

A.1.2.2. Dependencias y Configuración

Las dependencias clave incluyen:

- **FastAPI**: Utilizada para la creación de la API REST.
- **Pymongo**: Utilizada para interactuar con la base de datos MongoDB.
- **CORS Middleware**: Configurado para permitir peticiones desde cualquier origen, asegurando que la API pueda ser utilizada por diferentes clientes.

Las dependencias se configuran en `main.py`, garantizando que la API pueda responder a solicitudes desde diversos orígenes de manera segura.

A.1.2.3. Conexión con MongoDB

La conexión a MongoDB se realiza mediante la función `connect_to_mongo` en `mongo_utils.py`. Esta función utiliza la cadena de conexión proporcionada en la configuración y devuelve una instancia de `MongoClient`. La conexión se establece al iniciar la aplicación y se reutiliza en los diferentes endpoints para garantizar la eficiencia.

A.1.2.4. Endpoints de la API

Cada endpoint definido en `main.py` utiliza funciones de `mongo_utils.py` para interactuar con la base de datos.

Cada función está documentada con comentarios que describen su propósito, los parámetros que recibe y los posibles errores que podrían ocurrir.

A.1.2.5. Manejo de Errores y Validaciones

El manejo de errores se realiza utilizando `HTTPException` de FastAPI. Por ejemplo, si no se encuentran bases de datos o documentos, se lanza un error 404.

A.1.2.6. Operaciones Asincrónicas

La API aprovecha las capacidades asincrónicas de FastAPI para manejar múltiples solicitudes simultáneas de manera eficiente. Esto es especialmente importante cuando se realizan consultas que podrían tomar tiempo, garantizando una experiencia de usuario fluida.

A.1.2.7. Middleware y Seguridad

Se configuró el middleware CORS en `main.py` para permitir peticiones desde cualquier origen. Aunque actualmente la API no implementa autenticación, el uso de middleware facilita la futura implementación de mecanismos de seguridad como autenticación y autorización.

A.1.2.8. Pruebas de la API

La API se puede probar utilizando herramientas como **Postman** o **Swagger UI**, que permiten interactuar con los diferentes endpoints y verificar su funcionamiento. Además, se recomienda implementar pruebas unitarias e integradas para asegurar la estabilidad del sistema. Los intentos de extraer datos de los dispositivos resultaron insuficientes, ya que no se logró obtener la variedad y cantidad de datos dispersos requeridos para un análisis significativo. Por esta razón, se recurrió a la búsqueda de conjuntos de datos ya disponibles que contuvieran la información necesaria. Posteriormente, se muestra cómo se obtuvieron estos datos, cómo fueron limpiados y analizados, y finalmente cómo se insertaron en una base de datos MongoDB.

A.1.2.9. Búsqueda de Datos

Para suplir la falta de acceso directo a los datos de los dispositivos de salud, se llevó a cabo una búsqueda exhaustiva de conjuntos de datos que pudieran cumplir con los objetivos de este estudio. Esta búsqueda se realizó a través de diversas plataformas, como Kaggle, Google Search, y páginas especializadas en tesis y estudios científicos. Se exploraron diferentes recursos para encontrar la mayor cantidad posible de datos que provinieran de relojes de salud inteligentes.

Durante esta fase de investigación, se encontraron varios conjuntos de datos, pero la mayoría no cumplían con los criterios de calidad o no tenían la diversidad de información necesaria para el análisis. Entre los conjuntos de datos disponibles, solo se consideraron realmente factibles los correspondientes al Apple Watch y Fitbit, ya que los datos de otros dispositivos no estaban disponibles o no cumplían con los requisitos de completitud y calidad para su estudio. Estos dos conjuntos de datos fueron seleccionados para el análisis, dado que proporcionaban información relevante sobre la actividad física y los parámetros de salud de los usuarios.

A.1.2.10. Limpieza de Datos

Una vez obtenidos los conjuntos de datos de Apple Watch y Fitbit, fue necesario llevar a cabo una limpieza exhaustiva para garantizar que los datos estuvieran en un formato adecuado para su posterior almacenamiento y análisis. La limpieza de datos es un paso fundamental para eliminar inconsistencias, datos faltantes y posibles errores que puedan sesgar los resultados del análisis.

Se utilizaron varios scripts de Python para transformar estos conjuntos de datos. Los scripts se encargaron de normalizar los nombres de los campos, eliminar registros incompletos, y convertir las unidades de medida para que fueran consistentes a lo largo de todo el conjunto de datos. Además, se llevaron a cabo procesos de codificación para transformar las variables categóricas en valores numéricos, lo cual facilitó el análisis estadístico y la integración en

la base de datos. El objetivo principal de esta etapa fue garantizar que los datos estuvieran limpios y fueran consistentes, lo cual es esencial para obtener resultados fiables en el análisis posterior.

A.1.2.11. Almacenamiento de Datos

Tras la limpieza de los conjuntos de datos, se procedió a su almacenamiento en una base de datos NoSQL utilizando MongoDB. MongoDB fue seleccionada debido a su flexibilidad para manejar datos no estructurados y su capacidad para escalar de acuerdo con el volumen de datos. La estructura de la base de datos se diseñó de manera que cada dispositivo de salud tuviera su propia colección, almacenando información como los pasos diarios, la frecuencia cardiaca, las calorías quemadas y la distancia recorrida, entre otros.

Figura A.1.: Logo de MongoDB

En este contexto, se presentaron dos colecciones principales: una para los datos de Apple Watch y otra para los datos de Fitbit. Estas colecciones fueron diseñadas con el objetivo de ser lo suficientemente flexibles como para adaptarse a futuros cambios en la estructura de los datos o incorporar nuevos dispositivos.

La organización de los datos en estas colecciones permite un acceso eficiente y facilita el análisis de las relaciones entre las distintas variables de salud y actividad física. Esto es especialmente importante al considerar la gran cantidad de datos generados por los usuarios de estos dispositivos de salud inteligentes. Además, la base de datos NoSQL facilita la incorporación de nuevos tipos de datos y nuevas colecciones en caso de que se requiera ampliar el análisis en el futuro.

A continuación se presentan las tablas con la descripción de los campos almacenados en la base de datos. Estas tablas contienen los tipos de datos utilizados en cada colección junto con una explicación detallada de cada campo.

Dato	Descripción
Unnamed: 0	Índice de la entrada de datos, que sirve como identificador único.
age	Edad del participante en años.
gender	Género del participante, representado numéricamente (por ejemplo, 1 para hombre, 2 para mujer).
height	Altura del participante en centímetros.
weight	Peso del participante en kilogramos.
Applewatch.Steps_LE	Pasos registrados por el Apple Watch, posiblemente ajustados para una estimación logarítmica.
Applewatch.Heart_LE	Frecuencia cardiaca registrada por el Apple Watch, posiblemente ajustada para una estimación logarítmica.
Applewatch.Calories_LE	Calorías quemadas según el Apple Watch, posiblemente ajustadas para una estimación logarítmica.
Applewatch.Distance_LE	Distancia recorrida registrada por el Apple Watch, posiblemente ajustada para una estimación logarítmica.
EntropyApplewatchHeartPerDay_LE	Cálculo de la entropía de los datos diarios de frecuencia cardiaca del Apple Watch.
EntropyApplewatchStepsPerDay_LE	Cálculo de la entropía del conteo diario de pasos del Apple Watch.
RestingApplewatchHeartrate_LE	Frecuencia cardiaca en reposo registrada por el Apple Watch.
CorrelationApplewatchHeartrateSteps_LE	Correlación entre la frecuencia cardiaca y los pasos registrados por el Apple Watch.
NormalizedApplewatchHeartrate_LE	Frecuencia cardiaca normalizada a un rango específico para el análisis.
ApplewatchIntensity_LE	Intensidad del ejercicio estimada a partir de la frecuencia cardiaca y datos de movimiento del Apple Watch.
SDNormalizedApplewatchHR_LE	Desviación estándar de la frecuencia cardiaca normalizada del Apple Watch.
ApplewatchStepsXDistance_LE	Producto de los pasos y la distancia, representando el nivel de actividad del Apple Watch.
activity_trimmed	Estado de actividad categorizado del participante (por ejemplo, acostado, caminata a ritmo propio).

Tabla A.1.: Descripción de los datos de Apple Watch

Dato	Descripción
Unnamed: 0	Índice de la entrada de datos, que sirve como identificador único.
age	Edad del participante en años.
gender	Género del participante, representado numéricamente (por ejemplo, 1 para hombre, 2 para mujer).
height	Altura del participante en centímetros.
weight	Peso del participante en kilogramos.
Fitbit.Steps_LE	Pasos registrados por el Fitbit, posiblemente ajustados para una estimación logarítmica.
Fitbit.Heart_LE	Frecuencia cardiaca registrada por el Fitbit, posiblemente ajustada para una estimación logarítmica.
Fitbit.Calories_LE	Calorías quemadas según Fitbit, posiblemente ajustadas para una estimación logarítmica.
Fitbit.Distance_LE	Distancia recorrida registrada por el Fitbit, posiblemente ajustada para una estimación logarítmica.
EntropyFitbitHeartPerDay_LE	Cálculo de la entropía de los datos diarios de frecuencia cardiaca del Fitbit.
EntropyFitbitStepsPerDay_LE	Cálculo de la entropía del conteo diario de pasos del Fitbit.
RestingFitbitHeartrate_LE	Frecuencia cardiaca en reposo registrada por el Fitbit.
CorrelationFitbitHeartrateSteps_LE	Correlación entre la frecuencia cardiaca y los pasos registrados por el Fitbit.
NormalizedFitbitHeartrate_LE	Frecuencia cardiaca normalizada a un rango específico para el análisis.
FitbitIntensity_LE	Intensidad del ejercicio estimada a partir de la frecuencia cardiaca y datos de movimiento del Fitbit.
SDNormalizedFitbitHR_LE	Desviación estándar de la frecuencia cardiaca normalizada del Fitbit.
FitbitStepsXDistance_LE	Producto de los pasos y la distancia, representando el nivel de actividad del Fitbit.
activity_trimmed	Estado de actividad categorizado del participante (por ejemplo, acostado, caminata a ritmo propio).

Tabla A.2.: Descripción de los datos de Fitbit

Glosario

Monitoreo de la salud: Proceso continuo de recolección de datos relacionados con el bienestar físico y mental de un individuo, como el ritmo cardíaco, la actividad física, y la calidad del sueño.

Relojes de salud (Smartwatches): Dispositivos portátiles que integran funciones avanzadas para monitorear y analizar datos de salud.

Wearables: Dispositivos electrónicos que se llevan puestos en el cuerpo y que proporcionan funciones de monitoreo de la salud y otras capacidades tecnológicas.

Sensores de salud: Componentes electrónicos incorporados en los wearables que se utilizan para medir diferentes parámetros fisiológicos, como la frecuencia cardíaca o los niveles de oxígeno en sangre.

Calidad del sueño: Indicador del bienestar relacionado con el descanso, que puede medirse a través de parámetros como la duración del sueño y las fases de sueño.

Sistema SOA (Service-Oriented Architecture): Estilo de arquitectura de software que permite la creación de servicios web interactúan unos con otros.

Datos de salud: Información recolectada de los usuarios mediante wearables, relacionada con la actividad física, el estado fisiológico, y otros aspectos de la salud personal.

Electrocardiograma (ECG): Método de monitoreo eléctrico del corazón que registra las señales eléctricas para detectar arritmias u otras irregularidades en el ritmo cardíaco. Algunos smartwatches pueden realizar un ECG directamente desde la muñeca del usuario.

Variabilidad de la Frecuencia Cardíaca (HRV): Medida de las variaciones en los intervalos entre latidos consecutivos del corazón. Un alto valor de HRV indica una buena respuesta adaptativa del sistema nervioso autónomo, mientras que un valor bajo puede estar relacionado con estrés o fatiga.

Saturación de Oxígeno (SpO_2): Porcentaje de oxígeno presente en la sangre. Esta métrica se obtiene mediante sensores que miden la cantidad de oxígeno que se transporta a través de la hemoglobina. Un valor bajo de SpO_2 puede indicar problemas respiratorios o deficiencias de oxígeno.

Monitorización de Estrés: Técnica utilizada por algunos dispositivos wearables para estimar el nivel de estrés a partir de la variabilidad de la frecuencia cardíaca y otras métricas fisiológicas. El propósito es ayudar al usuario a controlar mejor su bienestar emocional.

Body Battery: Métrica utilizada por algunos dispositivos Garmin que estima los niveles de energía del cuerpo a lo largo del día, basada en factores como la calidad del sueño, la actividad física y el estrés. Ayuda al usuario a gestionar su energía y programar mejor sus actividades.

Bioimpedancia (BIA): Técnica para medir la composición corporal mediante la resistencia eléctrica. Se utiliza para obtener datos sobre la masa muscular, porcentaje de grasa y agua corporal. Algunos relojes inteligentes integran esta tecnología para proporcionar un análisis más profundo de la salud física.

SDK (Software Development Kit): Conjunto de herramientas y bibliotecas que permiten a los desarrolladores crear aplicaciones específicas para dispositivos, como los wearables. Los SDK permiten una mejor integración entre aplicaciones y dispositivos, facilitando la

personalización del monitoreo de la salud.

Medición de la Presión Arterial: Algunos dispositivos wearables tienen la capacidad de medir la presión arterial del usuario mediante el uso de sensores especiales. Este dato es importante para el monitoreo de la salud cardiovascular.

VO2 Max: Máxima cantidad de oxígeno que el cuerpo puede utilizar durante el ejercicio. Es un indicador clave del rendimiento cardiovascular y la capacidad aeróbica del usuario. Los relojes inteligentes pueden estimar este valor basándose en los datos de frecuencia cardíaca y la intensidad de la actividad.

API (Interfaz de Programación de Aplicaciones): Conjunto de definiciones y protocolos que permiten la comunicación entre diferentes aplicaciones y dispositivos. Las APIs de salud permiten a los desarrolladores acceder a datos biométricos de los wearables para crear soluciones más personalizadas y eficientes.

Sprint: Un periodo de tiempo definido, usualmente entre 1 a 4 semanas, durante el cual un equipo de desarrollo trabaja para completar un conjunto específico de tareas

Scrum: Un marco de trabajo ágil utilizado para la gestión de proyectos, que divide el desarrollo en iteraciones llamadas sprints. Incluye roles específicos como Product Owner y Scrum Master.

FastAPI: Un framework web rápido para crear APIs en Python, usado para desarrollar el backend del proyecto.

CSV (Comma-Separated Values): Un formato de archivo que permite almacenar datos tabulares en texto plano, donde los valores están separados por comas.

React: Una biblioteca de JavaScript utilizada para construir interfaces de usuario, conocida por su enfoque en el desarrollo de componentes reutilizables.

Índices de Base de Datos: Estructuras que mejoran la velocidad de recuperación de datos en una base de datos, especialmente en el caso de grandes volúmenes de información.

Bibliografía

[1] Fitbit-web-api-endpoints. https://dev.fitbit.com/build/reference/web-api/explore/, 2023. [Recurso online, accedido el 14 de octubre de 2024].

[2] Garmin sdk. https://developer.garmin.com/connect-iq/overview/, 2023. [Recurso online, accedido el 14 de octubre de 2024].

[3] Garmin sdk. https://developer.garmin.com/health-sdk/overview/, 2023. [Recurso online, accedido el 14 de octubre de 2024].

[4] Pydantic documentation. https://docs.pydantic.dev/latest/, 2023. [Recurso online, accedido el 14 de octubre de 2024].

[5] Fitbit-Developer. Fitbit web api reference. https://dev.fitbit.com/build/reference/web-api/, 2023. [Recurso online, accedido el 14 de octubre de 2024].

[6] Garmin-Developer-Program. Garmin connect developer program | health api. https://developer.garmin.com/gc-developer-program/health-api/, 2023. [Recurso online, accedido el 14 de octubre de 2024].

[7] HealthKit-Data-Types. Data types | apple developer documentation. https://developer.apple.com/documentation/healthkit/data_types, 2023. [Recurso online, accedido el 14 de octubre de 2024].