Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

Arquitectura avanzada con sensores vestibles y teléfonos inteligentes para la monitorización de múltiples usuarios

Alumno/a: David Díaz Jiménez

Tutor/a: Javier Medina Quero
Macarena Espinilla Estévez

Dpto.: Informática

Mayo, 2023

Agradecimientos

Es un honor para mí expresar mi profundo agradecimiento a mi familia, amigos y tutores, por su apoyo incansable durante el desarrollo de este proyecto. Su apoyo emocional y motivacional ha sido fundamental para lograr los objetivos establecidos y alcanzar el éxito. Sin su ayuda, este proyecto no hubiera sido posible. Gracias de corazón por su constante presencia y por ser parte de mi vida.

Tabla de contenidos

Lista de figuras

Lista de tablas

Lista de listados de código

Capítulo 1

Introducción

En este capítulo se presentará una introducción al trabajo fin de máster (TFM) junto con su motivación. Posteriormente, se indicará el propósito general y los objetivos específicos que persigue. Además, se indicará la relación de la propuesta con los Objetivos de Desarrollo Sostenible y, finalmente, se detallará la estructura que sigue el presente documento.

1.1. Introducción y motivación

En la actualidad, los avances en la informática han dado lugar a una gran cantidad de dispositivos vestibles que permiten recabar una amplia variedad y cantidad de datos. Estos dispositivos, que incluyen desde smartwatches hasta sensores de fitness, son capaces de monitorear aspectos importantes de la salud y el bienestar, como el sueño, la actividad física y los signos vitales. Además, la capacidad de almacenar y transmitir estos datos en tiempo real permite un análisis más profundo y detallado de la información, lo que a su vez permite una mayor comprensión y mejora de la salud y el bienestar [6, 7, 8].

Los dispositivos vestibles, también denominados dispositivos weareables, también ofrecen una gran cantidad de información sobre nuestro entorno y nuestras interacciones con él, lo que puede ser utilizado para mejorar la calidad de vida y la eficiencia en diferentes aspectos de la sociedad, como la salud, la seguridad y la productividad [9, 10]. Además, la combinación de tecnologías como la inteligencia artificial y el aprendizaje automático permite una mejor personalización y adaptación de los dispositivos a las necesidades individuales [11].

Figura 1.1: Dispositivos Weareables [1].

Para la gestión de datos de los dispositivos vestibles, existen soluciones en forma de API (Application Programming Interface) que ofrecen las empresas, como pueden ser Fitbit y Garmin.

Estas APIs se dividen de forma principal en API-REST cloud o SDK de desarrollo de aplicaciones nativas que pueden ser ejecutadas en los propios dispositivos. Dichas soluciones son herramientas útiles para que los desarrolladores puedan acceder a los datos de fitness y salud de los usuarios de sus plataformas y crear aplicaciones o integraciones que mejoren la experiencia del usuario. Sin embargo, estas API presentan algunas limitaciones que se mencionan a continuación:

1. Limitaciones en la personalización: Las API REST de Fitbit y Garmin están diseñadas para proporcionar acceso a un conjunto limitado de datos y funcionalidades. Esto significa que los desarrolladores pueden encontrar dificultades para personalizar la información y las funcionalidades para satisfacer sus necesidades específicas.

2. Requiere experiencia en programación: Para utilizar las API REST de Fitbit y Garmin, los desarrolladores necesitan tener conocimientos sólidos en programación y estar familiarizados con los protocolos de comunicación web y los lenguajes de programación comunes como JSON y XML.

3. Limitaciones en la velocidad de respuesta: Las API REST pueden tener limitaciones en la velocidad de respuesta debido a la carga de solicitudes que reciben. Si hay una gran cantidad de solicitudes simultáneas, las respuestas pueden retrasarse, lo que puede afectar la experiencia del usuario final.

4. Límites en el número de solicitudes: Las API REST tienen límites en el número de solicitudes que se pueden realizar en un período de tiempo determinado. Esto significa que los desarrolladores pueden tener dificultades para acceder a grandes cantidades de datos en poco tiempo y pueden necesitar implementar estrategias de almacenamiento en caché para minimizar la cantidad de solicitudes realizadas.

5. Tiempo de disponibilidad de los datos: Los datos en las API REST de Fitbit y Garmin no están disponibles en tiempo real. Esto significa que los desarrolladores pueden encontrar dificultades para acceder a los datos más recientes de los usuarios, lo que puede afectar la precisión de las aplicaciones y las integraciones que están creando.

6. Datos cuantitativos: Las API REST proporcionan únicamente información de forma cuantitativa a través de datos y resúmenes.

7. En API REST: no es posible crear procesos lógicos integrados que puedan ser computados en el propio dispositivo bajo una filosofía Fog o Edge computing. En API REST: no es posible crear procesos lógicos integrados que puedan ser computados en el propio dispositivo bajo una filosofía Fog o Edge computing.

Por lo tanto, es necesario disponer de arquitecturas que superen estas limitaciones y brinden una gestión eficiente y flexible de los datos generados por los dispositivos wearables.

Este trabajo fin de máster propone una arquitectura avanzada con una serie de beneficios significativos. En primer lugar, destaca su capacidad para incorporar datos de múltiples dispositivos, lo que permite a los usuarios tener acceso a una amplia variedad de información relevante en un solo lugar usando un sistema de persistencia y comunicación flexible.

Además, est modelo es capaz de generar resúmenes lingüísticos precisos y eficaces, lo que facilita la comprensión de la información presentada y agiliza el proceso análisis y toma de decisiones. En la literatura se evidencia como el uso de resúmenes lingüísticos mejora el entendimiento de los datos proporcionados con sensores [12, 13, 14, 15, 16, 17].

Finalmente, esta arquitectura ofrece una API intuitiva y fácil de usar que puede ser utilizada por personas sin conocimientos previos de programación, lo que amplía la accesibilidad y la utilidad de la plataforma en general. Es por ello que los beneficios de esta arquitectura son múltiples y significativos, y están diseñados para mejorar la eficiencia, la precisión y la accesibilidad en el manejo de información compleja.

1.2. Propósito general y objetivos

1.2.1. Propósito general

El objetivo del proyecto es el desarrollo de una arquitectura avanzada multiusuario que permita la recopilación de datos generados por los sensores de dispositivos wearables. Esta arquitectura será diseñada para ser escalable, flexible y segura, y para permitir la integración eficiente de los datos generados por los sensores de los dispositivos wearables en una plataforma centralizada, permitiendo resúmenes lingüísticos.

1.2.2. Objetivos específicos

Una vez se ha definido el propósito general del proyecto, a continuación, se definen los objetivos específicos:

- Analizar y estudiar los conceptos y tecnologías asociadas con el desarrollo del proyecto.
- Evaluar los dispositivos wearables que hay en el mercado y seleccionar el más adecuado para el propósito de este trabajo fin de máster.
- Crear un modelo de datos para sensores wearables heterogéneos y evaluar las distintas opciones de bases de datos disponibles con el fin de elegir la más apropiada para el proyecto.
- Diseñar una arquitectura escalable y flexible que permita la integración de una gran cantidad de dispositivos wearables y de una gran cantidad de datos generados por estos dispositivos con resúmenes lingüísticos.
- Desarrollar una aplicación para dispositivos móviles que permita gestionar y analizar los datos recogidos de manera sencilla y efectiva.

1.3. Jutificación con ODS

Figura 1.2: Objetivos de Desarrollo Sostenible [2].

Los Objetivos de Desarrollo Sostenible (ODS) [18], son unos acuerdos de objetivos globales adoptados por las Naciones Unidas en 2015 como parte de la Agenda 2030 para el Desarrollo Sostenible. Estos objetivos tienen como objetivo abordar los desafíos mundiales más urgentes, incluyendo la pobreza, la desigualdad, el cambio climático, la degradación ambiental y la falta de acceso a servicios básicos como la educación y la atención médica.

La consecución de los ODS es una tarea de todos. Cada persona, empresa, organización y gobierno tiene un papel que desempeñar en la construcción de un mundo más justo y sostenible. En este sentido, es importante destacar que los ODS no son una responsabilidad exclusiva de los gobiernos o de los países en desarrollo. Todos los sectores de la sociedad, incluyendo las universidades, tienen un papel importante que desempeñar en la consecución de los ODS.

En este Trabajo Final de Máster (TFM), se aborda la importancia de identificar aquellos objetivos y metas que están alineados con la consecución de los ODS. A continuación se detallan los 17 ODS y posteriormente se identifican aquellos con los que está más relacionado la propuesta de este TFM.

1. ODS 1 - Fin de la Pobreza: Erradicar la pobreza extrema y reducir la desigualdad

económica y social.

2. ODS 2 - Hambre Cero: Lograr la seguridad alimentaria y una agricultura sostenible para todos.
3. ODS 3 - Salud y Bienestar: Garantizar una vida saludable y promover el bienestar para todas las personas en todas las edades.
4. ODS 4 - Educación de Calidad: Asegurar una educación inclusiva, equitativa y de calidad para todos.
5. ODS 5 - Igualdad de Género: Lograr la igualdad de género y empoderar a todas las mujeres y niñas.
6. ODS 6 - Agua Limpia y Saneamiento: Garantizar la disponibilidad y la gestión sostenible del agua y el saneamiento para todos.
7. ODS 7 - Energía Asequible y no Contaminante: Garantizar el acceso a energías renovables y asequibles para todos, y promover la eficiencia energética.
8. ODS 8 - Trabajo Decente y Crecimiento Económico: Promover un crecimiento económico sostenible, inclusivo y sostenible, y garantizar el trabajo decente y la protección social para todos.
9. ODS 9 - Industria, Innovación e Infraestructura: Promover la innovación y el desarrollo de infraestructuras y sistemas de información para recopilar y analizar datos en tiempo real.
10. ODS 10 - Reducción de las Desigualdades: Reducir la desigualdad económica, social y territorial, y promover la inclusión social, la no discriminación y la justicia para todos.
11. ODS 11 - Ciudades y Comunidades Sostenibles: Lograr ciudades y comunidades más inclusivas, seguras, resilientes y sostenibles.
12. ODS 12 - Producción y Consumo Responsables: Promover el consumo y la producción responsables, reduciendo la huella ambiental y fomentando la eficiencia en el uso de los recursos naturales.
13. ODS 13 - Acción por el Clima: Adoptar medidas urgentes para combatir el cambio climático y sus efectos, y promover la adaptación y la resiliencia ante los impactos del cambio climático.
14. ODS 14 - Vida Submarina: Proteger y conservar la vida submarina y los ecosistemas marinos.

15. ODS 15 - Vida de Ecosistemas Terrestres: Proteger y restaurar los ecosistemas terrestres y promover la biodiversidad y los servicios ecosistémicos.
16. ODS 16 - Paz, Justicia e Instituciones Sólidas: Promover sociedades pacíficas, justas e inclusivas, y garantizar el acceso a la justicia y la igualdad ante la ley.
17. ODS 17 - Alianzas para lograr los objetivos: Fortalecer la cooperación internacional y la colaboración entre diferentes actores, como empresas, organizaciones y gobiernos, para lograr los objetivos globales de desarrollo sostenible.

En este sentido, el proyecto propuesto podría contribuir significativamente a algunos de los ODS.

El TFM propuesto tiene el potencial de contribuir significativamente a varios de los Objetivos de Desarrollo Sostenible (ODS). En primer lugar, el proyecto podría impactar el ODS 3: Salud y Bienestar, ya que la aplicación podría proporcionar información valiosa sobre la salud del usuario en tiempo real, lo que permitiría la detección temprana de posibles problemas y enfermedades. Esto tendría un impacto positivo en la calidad de vida del usuario y en la prevención de enfermedades graves.

Además, la arquitectura propuesta podría contribuir al ODS 9: Industria, Innovación e Infraestructura, al utilizar dispositivos wearables y generar descripciones lingüísticas de los flujos de datos. De esta manera, se promovería la innovación y el desarrollo de nuevas tecnologías en este campo, y se podría contribuir al desarrollo de infraestructuras y sistemas de información para recopilar y analizar datos.

Por último, la arquitectura también podría contribuir al ODS 17: Alianzas para lograr los objetivos, fomentando la colaboración entre diferentes actores para abordar los desafíos más urgentes. En este sentido, la aplicación podría promover la colaboración entre empresas de tecnología, instituciones de investigación, organizaciones sin fines de lucro y gobiernos.

En conclusión, el TFM propuesto tiene el potencial de contribuir significativamente a los ODS 3, 9 y 17, al promover la salud y el bienestar, fomentar la innovación y el desarrollo de infraestructuras y sistemas de información, y fomentar la colaboración entre diferentes actores para alcanzar los objetivos globales de desarrollo sostenible.

1.4. Estructura de la memoria

La memoria del proyecto está estructurada de la siguiente forma:

1. **Introducción y motivación** Se presenta el contexto y el objetivo del trabajo, y se justifica su importancia y relevancia.
2. **Planificación** Se establecen los objetivos a largo plazo y se desarrolla un plan detallado para alcanzarlos, incluyendo las tareas, recursos y fechas límite.
3. **Lógica difusa y protoformas lingüísticas** Se presentan los principales conceptos para elaborar resúmenes lingüísticos.
4. **Análisis.** Se determinan los requisitos funcionales y no funcionales del sistema a desarrollar.
5. **Diseño** Se explican los diferentes elementos y componentes de la arquitectura propuesta.
6. **Implementación** Se detalla la construcción de las aplicaciones y componentes de software, siguiendo los planes y diseños especificados en los capítulos previos.
7. **Caso de estudio** Se presenta el caso de estudio con los diferentes escenarios evaluados.
8. **Conclusiones** Se presentan los resultados y se destacan los logros del proyecto. También se identifican las áreas de mejora.
9. **Anexos** Se proporciona el manual de usuario para desplegar la arquitectura propuesta.

Capítulo 2

Planificación

En esta sección se discute la planificación del proyecto, la cual es esencial para garantizar que el proyecto se desarrolle de manera eficiente y efectiva. La planificación del proyecto incluye la identificación de los objetivos del proyecto, la definición de las tareas necesarias para alcanzar esos objetivos, la asignación de recursos y la estimación de los plazos de entrega.

También se describirán las metodologías utilizadas para la planificación del proyecto y se estimarán los costes asociados al proyecto. En resumen, esta sección proporcionará una visión general de la planificación del proyecto y su importancia en el desarrollo exitoso del mismo.

2.1. Metodología

La elección de una metodología de desarrollo de software es esencial para garantizar que el proyecto se implemente de manera eficiente y efectiva. Una metodología proporciona un marco estructurado para el desarrollo del proyecto, incluyendo una planificación detallada, la definición de tareas y responsabilidades, y un proceso de seguimiento y control.

Además, la metodología proporciona una serie de prácticas y procesos estandarizados que ayudan a garantizar la calidad del software y a reducir los riesgos asociados con el desarrollo del proyecto.

Existen varias metodologías de desarrollo de software, cada una con sus propias características y componentes. A continuación se describen algunas de las metodolo-

gías más comunes en desarrollo software.

2.1.1. Modelo en cascada

La metodología de cascada [19] es una metodología de desarrollo de software secuencial, en la que cada fase del desarrollo debe ser completada antes de pasar a la siguiente. Se caracteriza por su planificación detallada y su enfoque en la documentación.

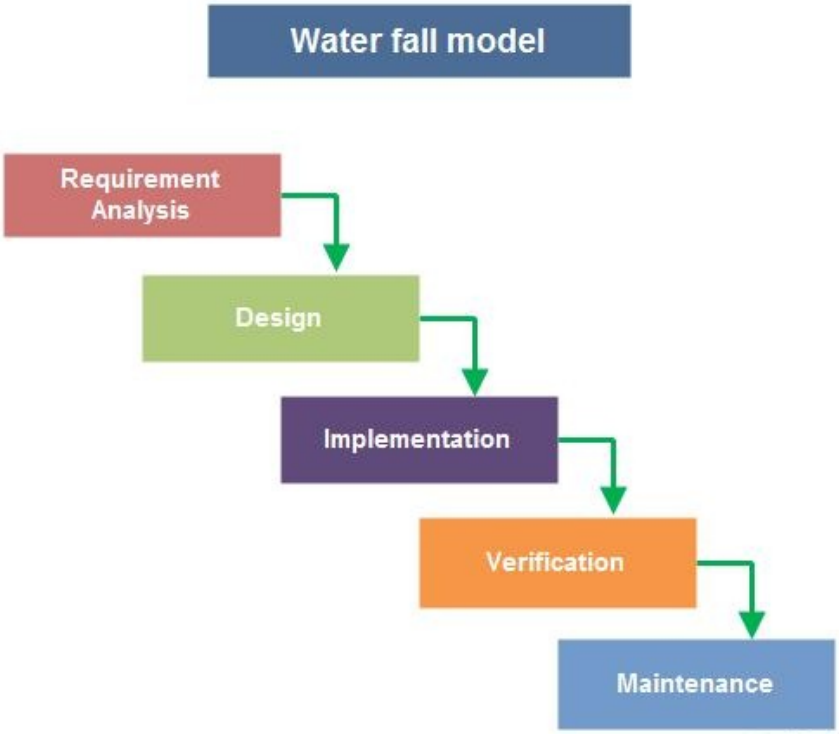

Figura 2.1: Fases del modelo en cascada [3].

La metodología de cascada puede dividirse en varias fases:

- Análisis: En esta fase, se identifican los requisitos del sistema y se estudian las necesidades del usuario. Se realiza un estudio de viabilidad para determinar si el proyecto es factible y se elabora un plan detallado del proyecto.
- Diseño: Se especifica el diseño detallado del sistema, incluyendo la arquitectura, la interfaz de usuario, y los componentes necesarios para el sistema. También se describen los procedimientos de prueba y se desarrollan los diagramas de flujo.
- Implementación: En esta fase se realiza el desarrollo.

- Pruebas: Se realizan pruebas unitarias y pruebas de integración para garantizar que el sistema cumpla con los requisitos definidos anteriormente.
- Mantenimiento: Una vez que el sistema ha sido liberado, se realiza el mantenimiento y se corrigen los errores encontrados.

Es importante señalar que esta metodología tiene sus limitaciones, ya que en proyectos con requisitos inestables supone realizar un gran cantidad de documentación, siendo menos flexible que otras metodologías. Aun así, es utilizada ampliamente en proyectos con requisitos claramente definidos y estables.

2.1.2. Scrum

Scrum es un marco de trabajo ágil [20, 21] para el desarrollo de proyectos y productos, especialmente software. Fue desarrollado por Ken Schwaber y Jeff Sutherland.

Scrum se basa en un enfoque iterativo e incremental, donde el equipo trabaja en sprints (períodos de tiempo, generalmente de 2 a 4 semanas) para completar un conjunto de tareas o entregables. Cada sprint comienza con una planificación, seguida de una ejecución y finaliza con una revisión y una retrospectiva.

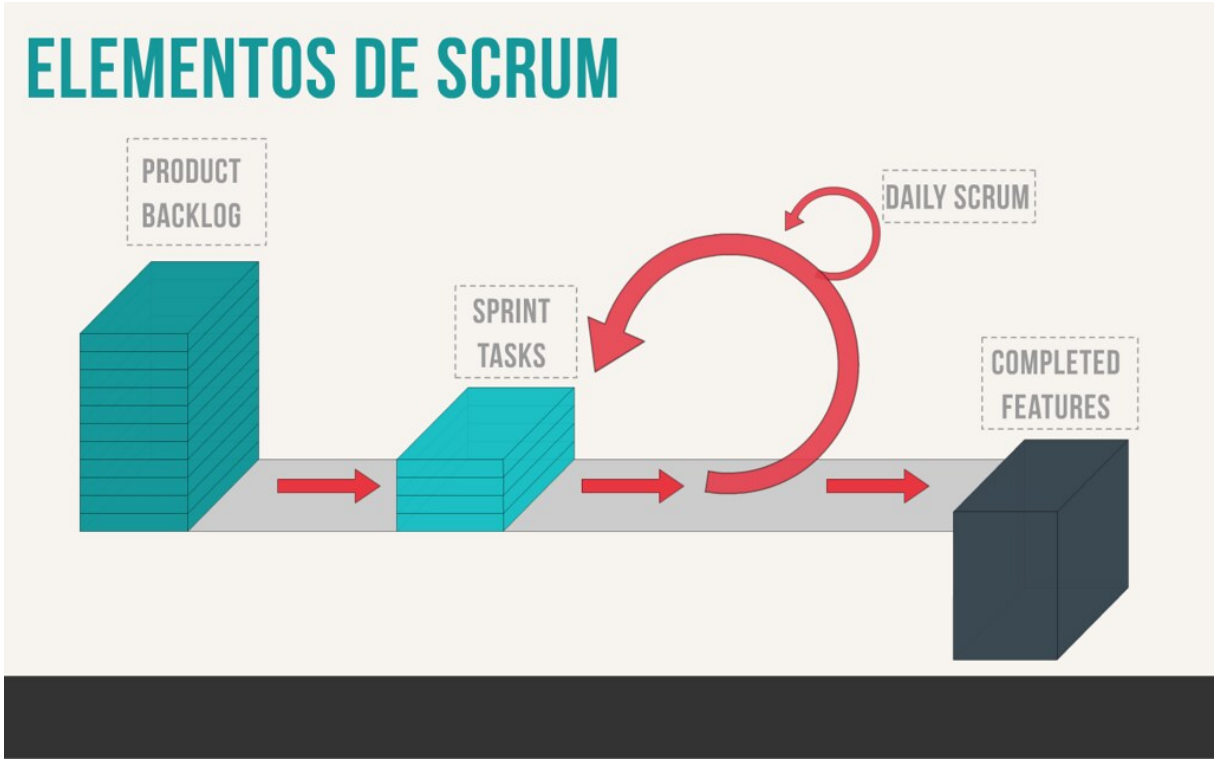

Figura 2.2: Elementos de Scrum [4].

Los roles clave en Scrum son el Product Owner, el Scrum Master y el equipo de desarrollo. El Product Owner es responsable de la estrategia y el alineamiento del producto con los objetivos del negocio. El Scrum Master es el facilitador y el guardián del proceso Scrum, asegurando que el equipo sigue las prácticas y los valores Scrum. El equipo de desarrollo es responsable de la planificación, la ejecución y la entrega del trabajo en cada sprint.

Scrum también define un conjunto de artefactos, incluyendo el product backlog, el sprint backlog y el incremento del producto, que son utilizados para planificar y monitorear el progreso del proyecto.

En resumen, Scrum es un marco ágil que se enfoca en la entrega continua de valor a través de sprints cortos, y fomenta la colaboración, la transparencia y la adaptabilidad a cambios en el proyecto.

2.1.3. Modelo incremental

El modelo incremental [22] es un enfoque de desarrollo de proyectos en el cual el proceso se divide en etapas o fases sucesivas, llamadas incrementos, cada una de las cuales añade una funcionalidad adicional al producto o proyecto. El objetivo es entregar un producto o proyecto completo a través de una serie de entregas parciales.

El modelo incremental se caracteriza por la entrega continua de valor al cliente o usuario y la adaptación al cambio. A medida que se completan los incrementos, el equipo recopila retroalimentación del cliente o usuario y la incorpora en el siguiente incremento, lo que permite mejorar y ajustar el producto o proyecto a medida que avanza.

En el modelo incremental, el equipo trabaja en un ciclo de planificación, diseño, construcción y evaluación. La planificación se enfoca en definir los objetivos y tareas para el incremento actual, el diseño se enfoca en crear un plan para cumplir esas tareas y construir el producto o proyecto, la construcción se enfoca en ejecutar ese plan y finalmente la evaluación se enfoca en revisar el trabajo realizado y recopilar retroalimentación.

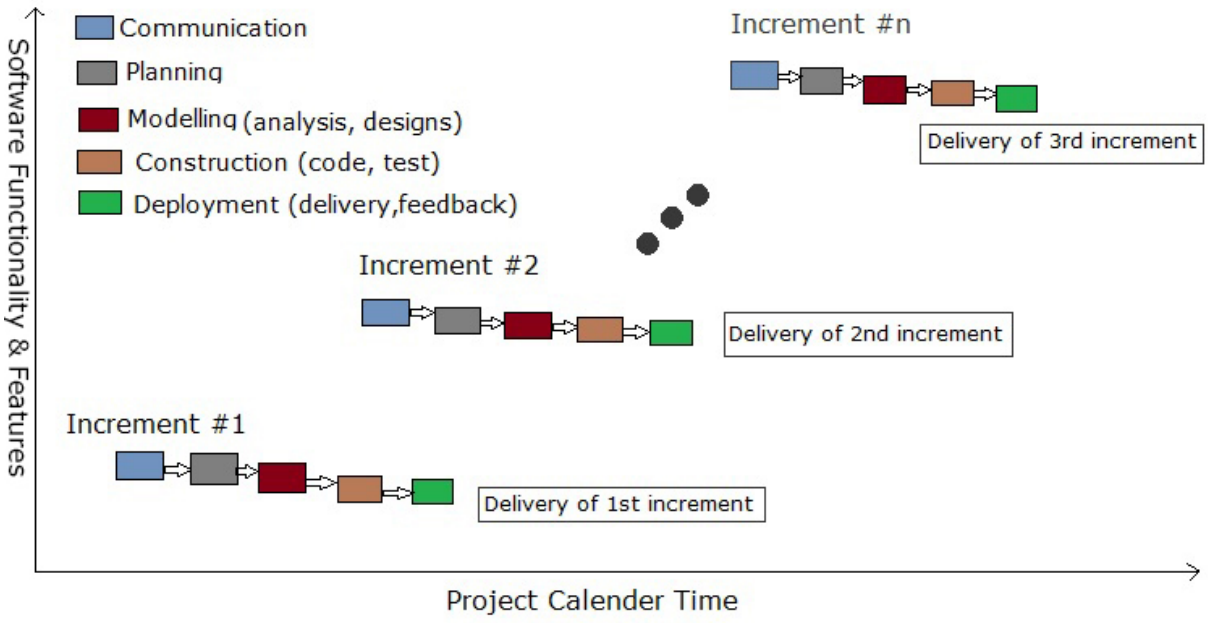

Figura 2.3: Modelo incremental [4].

2.1.4. Elección de la metodología

En base a lo explicado anteriormente, se ha seleccionado el *modelo incremental* por las siguientes razones:

Flexibilidad: El modelo incremental permite adaptarse a cambios en los requisitos del proyecto de manera eficiente. Permite planificar y entregar incrementos parciales, lo que permite ser más flexible y reaccionar rápidamente a los cambios en el proyecto.

Entrega continua de valor: El modelo incremental permite entregar versiones parciales del producto o proyecto, lo que permite obtener retroalimentación temprana del cliente o usuario y asegurar que el proyecto está en línea con sus necesidades.

Mayor control: Habilita un mayor control en proyectos con pocos desarrolladores, lo que permite tomar decisiones rápidas y adaptarse a los cambios en el proyecto.

Escalabilidad: El modelo incremental permite escalar el proyecto de manera gradual, lo que permite manejar un mayor volumen de trabajo y complejidad a medida que el proyecto avanza.

En resumen, se ha elegido el modelo incremental puesto que permite una mayor flexibilidad y adaptabilidad a los cambios en el proyecto, entregar valor continuamente, tener mayor control sobre el proyecto y escalar el proyecto de manera gradual.

2.2. Planificación temporal

Determinar las tareas principales dentro del proyecto es esencial para establecer la planificación temporal. En la Tabla 2.1, se pueden apreciar las tareas identificadas y la fecha de inicio y fin establecidas. Adicionalmente se hará uso de diagramas de Gantt.

Tarea	**Inicio - Finalización**
Documentación y revisión de trabajos	01/10/2022 - 09/10/2022
Análisis de requisitos	09/10/2022 - 12/10/2022
Diseño del modelo de datos	11/10/2022 - 14/10/2022
Diseño de la API	14/10/2022 - 20/10/2022
Diseño de las aplicaciones	20/10/2022 - 27/10/2022
Implementación de la API	27/10/2022 - 04/11/2022
Implementación de la base de datos	04/11/2022 - 07/11/2022
Implementación del proxy inverso	07/11/2022 - 09/11/2022
Implementación de la app del reloj	09/11/2022 - 30/11/2022
Implementación de la app de gestión	30/11/2022 - 26/12/2022
Pruebas	26/12/2022 - 24/01/2023
Redacción de la memoria	01/10/2022 - 29/01/2023

Tabla. 2.1: Fecha de inicio y de finalización de las principales tareas del proyecto.

Un diagrama de Gantt [23] es una herramienta de planificación y seguimiento de proyectos que muestra las tareas que deben ser realizadas y el tiempo previsto para su finalización. Se representa gráficamente mediante una barra horizontal que representa el tiempo, y otra vertical con las tareas. En la barra horizontal se marcan los plazos para cada tarea.

La necesidad de utilizar un diagrama de Gantt radica en que permite visualizar de manera clara y sencilla el estado de un proyecto, así como los plazos de finalización de las tareas, lo cual facilita la toma de decisiones y el seguimiento del mismo. Además, permite identificar posibles problemas y retrasos en el proyecto, y tomar medidas para solucionarlos.

A continuación, se presenta el diagrama de Gantt asociado a la planificación del proyecto, Figura 2.4, de tal manera que facilite entender la distribución temporal de las distintas tareas.

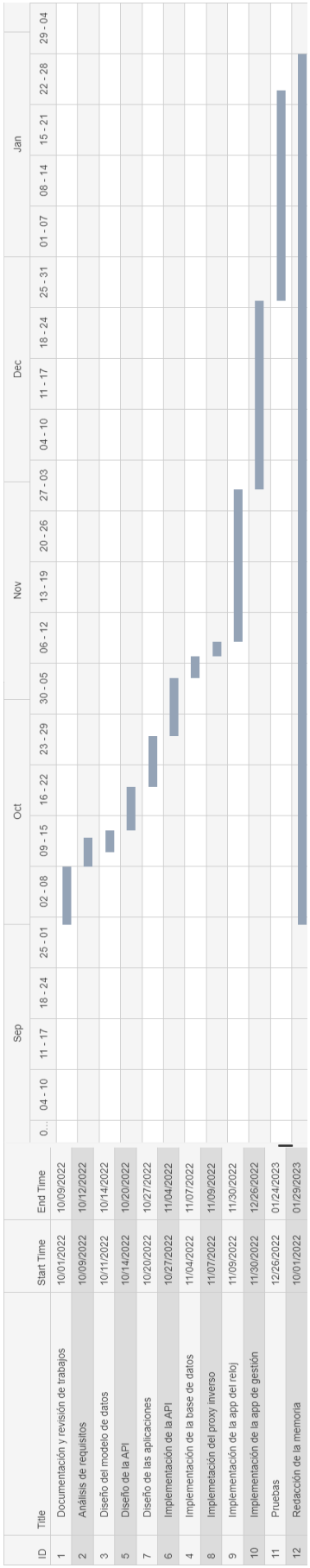

Figura 2.4: Diagrama de Gantt.

2.3. Estimación de recursos

En esta sección se hará una revisión de los costes y recursos asociados al desarrollo y ejecución del proyecto. Serán tenidos en cuenta costes asociados al proyecto de manera directa e indirecta. Las estimaciones aquí presentadas se han calculado en base a la duración estimada del proyecto.

2.3.1. Recursos materiales

- Equipamiento informático:
  - Procesador: AMD Ryzen 5600H
  - RAM: 16GB DDR4
  - GPU: Nvidia GTX 1650
  - Almacenamiento: 1TB SSD
  - Pantalla: 24"1920x1080 px
- Dispositivos:
  - FitBit Versa 3
- Software:
  - Visual Studio Code
  - Visual Paradigm Online
  - Overleaf
  - Fitbit Studio
  - Fitbit Simulator
- Salario:
  - 5 horas diarias
  - 80 días
- Alquiler de la oficina
- Servicios:
  - Electricidad

- Agua
- Internet: Fibra óptica 600 MB
- Material de oficina

2.3.2. Recursos humanos

Los recursos humanos son aquellos que se relacionan con el personal que trabaja en un proyecto. Incluyen costes como salarios, beneficios, capacitación y desarrollo del personal, entre otros. Es importante incluirlos en la planificación del proyecto porque el personal es un recurso vital para el éxito del proyecto. Si los costes humanos no se planifican adecuadamente, pueden generar problemas como falta de personal capacitado, sobre-costes y retrasos en la ejecución del proyecto. Por lo tanto, es esencial considerar los costes humanos en la planificación del proyecto para asegurar una gestión eficiente y efectiva del personal y garantizar el éxito del proyecto.

Una vez establecida la necesidad de la planificación de los costes humanos, se va a podrecer a su estimación. Para ello hay que tener en cuenta una serie condiciones previas, como pueden ser el número estimado de personas necesarias, o los días no laborables. En este último caso, el número de días hábiles del año queda reducido a 225 días. Respecto al personal necesario, se establece que sería contar con un analista programador, puesto que debería realizar el análisis de requisitos, diseño software e implementación.

2.3.3. Estimación de costes

Basándose en los recursos, tanto hardware como software (materiales) y humanos, descritos en secciones previas y en la estimación temporal, se ha elaborado una estimación de costos. Esta estimación tiene en cuenta los recursos necesarios para el correcto desarrollo del proyecto, incluyendo el equipamiento, los programas y herramientas necesarias, así como el personal y su capacitación.

La estimación de costos se ha realizado con el objetivo de proporcionar una visión general de los costos previstos y para poder llevar a cabo un seguimiento del presupuesto durante el desarrollo del proyecto.

Concepto	Importe unitario	Amortización	Total/mes
Ordenador	900€	4 años	8,33€
Versa 3	180€	4 años	1,82€
		Total	35,52€

Tabla. 2.2: Costes amortizables.

Concepto	Importe unitario	Amortización	Total/mes
Impuestos, salarios, seguro, etc.	1.026,68€/mes	3,5 meses	3593,7€
Alquiler Oficina	550€/mes	3,5 meses	1925€
Servicios (Agua, Electricidad, Internet)	160€/mes	3,5 meses	560€
		Total	4341,67€

Tabla. 2.3: Costes.

Este presupuesto se ha elaborado a partir de las estimaciones de tiempo previamente establecidas en secciones previas. Sin embargo, es importante tener en cuenta que estas estimaciones son solo eso, estimaciones, y que en un entorno real los resultados pueden variar significativamente.

Capítulo 3

Lógica difusa y protoformas lingüísticas

La lógica difusa [24] es una herramienta matemática que permite modelar y manipular la incertidumbre y la imprecisión inherentes a muchos problemas del mundo real. Esta lógica fue desarrollada por Lotfi A. Zadeh en la década de 1960 y desde entonces ha sido ampliamente utilizada en diversas aplicaciones como la ingeniería, la medicina, la economía, la robótica, entre otras.

La lógica difusa fue propuesta por Lotfi A. Zadeh [25] en 1965 como una alternativa a la lógica clásica de dos valores (verdadero o falso). La idea principal de la lógica difusa es que la verdad de una proposición no es binaria, sino que puede tener un grado de verdad que varía entre 0 y 1. Esta idea fue introducida por Zadeh en su artículo "Fuzzy sets", publicado en la revista Information and Control.

3.0.1. Protoformas lingüísticas difusas

Las protoformas lingüísticas difusas son herramientas que permiten describir de forma lingüística la información procesada mediante el lenguaje natural, a través del uso de variables difusas, cuantificadores y ventanas temporales difusas. Estas herramientas son útiles para representar la imprecisión e incertidumbre inherentes al lenguaje natural, y pueden ser utilizadas en diversas aplicaciones como sistemas de control, diagnóstico médico, análisis financiero, entre otros [26, 27, 28, 29].

En este trabajo, se utilizarán protoformas lingüísticas difusas para describir los flujos de datos de la frecuencia cardíaca y la aceleración. Estas variables pueden ser

descritas mediante conjuntos difusos que indican el grado de pertenencia de cada valor a una categoría específica. Además, se utilizarán cuantificadores difusos para expresar el grado de verdad de las proposiciones difusas y ventanas temporales difusas para describir la evolución temporal de las variables difusas.

3.0.2. Variables difusas

Las variables difusas son variables que pueden tomar valores en un conjunto difuso, es decir, un conjunto en el que los elementos tienen un grado de pertenencia que varía entre 0 y 1. Por ejemplo, la variable temperatura puede ser difusa si se considera que una temperatura de 20 grados Celsius tiene un grado de pertenencia de 1 en el conjunto templado, un grado de pertenencia de 0.5 en el conjunto caliente y un grado de pertenencia de 0 en el conjunto frío.

3.0.3. Ventanas temporales difusas

Las ventanas temporales difusas son herramientas que permiten describir la evolución temporal de una variable difusa. Por ejemplo, la temperatura puede ser descrita mediante una ventana temporal difusa que indica que en un cierto momento la temperatura es alta en un grado de pertenencia de 0.8 y que este grado de pertenencia disminuye o aumenta gradualmente a medida que es cierto en una ventana temporal difusa.

3.0.4. Cuantificadores difusos

Los cuantificadores difusos [30] son expresiones lingüísticas que permiten cuantificar la verdad de una proposición difusa. Por ejemplo, el cuantificador *muy* puede ser utilizado para indicar que una proposición es verdadera en un alto grado de pertenencia (por ejemplo, la temperatura está muy cerca de los 20 grados Celsius) o en un bajo grado (por ejemplo, la temperatura está muy lejos de los 20 grados Celsius).

3.0.5. Funciones de pertenencia

Una función de pertenencia es una función que asigna a cada valor de una variable el grado en el que ese valor pertenece a un conjunto difuso. En la lógica difusa, los conjuntos no son definidos por una lista de elementos, sino por una función de pertenencia que indica el grado de pertenencia de cada elemento en un conjunto.

Función trapezoidal

Una función de pertenencia trapezoidal es una de las funciones de pertenencia más comunes en la lógica difusa. Esta función se utiliza para definir conjuntos difusos que tienen una forma trapezoidal. Un conjunto difuso trapezoidal se caracteriza por cuatro parámetros: a, b, c y d, donde a y d son los extremos inferiores y superiores del intervalo de pertenencia, y b y c son los puntos donde la función alcanza el valor 1.

La función de pertenencia trapezoidal se define como:

$$\mu(x) = \begin{cases} 0 & si\ x \leq a \\ \frac{x-a}{b-a} & si\ a < x < b \\ 1 & si\ b \leq x \leq c \\ \frac{d-x}{d-c} & si\ c < x < d \\ 0 & si\ x \geq d \end{cases} \tag{3.1}$$

Esta ecuación indica que el grado de pertenencia $\mu(x)$ de un valor x en un conjunto difuso trapezoidal depende de la posición de x dentro del intervalo $[a, d]$ y de los valores de los parámetros a, b, c y d. En particular, la función de pertenencia trapezoidal toma el valor 1 en el intervalo $[b, c]$, el valor 0 fuera del intervalo $[a, d]$, y crece linealmente desde 0 a 1 en el intervalo $[a, b]$, y luego decrece linealmente desde 1 a 0 en el intervalo $[c, d]$.

Función triangular

Las funciones de pertenencia triangular son un tipo de función utilizada en lógica difusa para representar la incertidumbre o el conocimiento incompleto sobre una variable. Como su nombre indica, tienen forma de triángulo y se utilizan para modelar variables cuyo rango de valores aceptables es limitado.

La función de pertenencia triangular se define mediante la siguiente fórmula:

$$\mu(x) = \begin{cases} 0 & si\ x \leq a \\ \frac{x-a}{b-a} & si\ a < x < b \\ \frac{c-x}{c-b} & si\ b < x < c \\ 0 & si\ x \geq c \end{cases} \tag{3.2}$$

Donde a, b y c son los parámetros que definen los puntos en los que la función de pertenencia triangular cambia su valor. Estos parámetros representan el valor mínimo aceptable, el valor medio (o el punto más alto) y el valor máximo aceptable de la variable que se está modelando, respectivamente.

La función de pertenencia triangular se compone de dos partes triangulares unidas por un punto en común. El punto en común es el valor de x en el que la función alcanza su valor máximo, que es 1. La primera parte triangular se define por dos parámetros: el valor mínimo aceptable a (punto de inicio de la función) y el valor medio b (punto más alto de la función). La segunda parte triangular se define por otros dos parámetros: el valor medio b (punto más alto de la función) y el valor máximo aceptable c (punto final de la función).

La función de pertenencia triangular es simétrica si los parámetros b y c tienen el mismo valor. Si b es menor que c, la función de pertenencia triangular es asimétrica y puede utilizarse para modelar variables que tienen un rango de valores aceptables que no es simétrico.

Función sigmoidea

La función de pertenencia sigmoidea es un tipo de función utilizada en lógica difusa para modelar variables que tienen una transición gradual entre los valores de pertenencia. A diferencia de las funciones de pertenencia trapezoidales y triangulares, las funciones de pertenencia sigmoideas no tienen una forma geométrica simple y se representan mediante una curva suave.

La función de pertenencia sigmoidea se define mediante la siguiente fórmula:

$$\mu(x) = \frac{1}{1 + e^{-\alpha(x-\beta)}} \tag{3.3}$$

Donde α y β son los parámetros de la función que controlan la forma de la curva.

El parámetro β determina el valor de la variable en el que la función alcanza su valor medio (o el punto más alto), mientras que el parámetro α determina la pendiente de la curva en ese punto.

La función de pertenencia sigmoidea tiene una forma de S y su valor varía entre 0 y 1. En la práctica, se utiliza una versión escalada de la función de pertenencia sigmoidea para ajustarla a un rango específico de valores.

3.0.6. Función gaussiana

La función de pertenencia gaussiana es otro tipo de función utilizada en lógica difusa para modelar variables continuas. Se basa en la distribución normal o gaussiana y se utiliza para modelar variables que tienen un comportamiento más suave que las variables que se modelan con funciones de pertenencia trapezoidales, triangulares o sigmoideas.

La función de pertenencia gaussiana se define mediante la siguiente fórmula:

$$\mu(x) = e^{-\frac{(x-c)^2}{2\sigma^2}} \tag{3.4}$$

Donde c es el centro de la función (el valor donde alcanza su máximo) y σ es la desviación estándar de la distribución gaussiana. El parámetro σ controla la anchura de la función de pertenencia gaussiana y su forma es simétrica alrededor del centro c.

La función de pertenencia gaussiana tiene una forma de campana y su valor varía entre 0 y 1. La función alcanza su valor máximo en $x = c$ y disminuye a medida que nos alejamos del centro en ambas direcciones.

3.0.7. Aplicaciones de descripciones lingüísticas a otros contextos

Las descripciones lingüísticas difusas son una herramienta valiosa y versátil en diferentes áreas del conocimiento. Su capacidad para manejar la incertidumbre y la ambigüedad en la toma de decisiones, la evaluación del rendimiento y la modelización de sistemas complejos la convierten en una herramienta indispensable en la era de la inteligencia artificial y la automatización.

A continuación se realiza una revisión de diferentes contextos donde las descripciones lingüísticas difusas han proporcionado excelentes resultados:

- En [31], se introduce el concepto de protoform como representación lingüística de sensores locales y remotos. Estos configuran los antecedentes de las reglas en un Motor de Inferencia. Se presenta un caso de estudio donde dos habitantes con un dispositivo wearable realizan actividades en un Smart Lab. Cada dispositivo portátil infiere las actividades diarias dentro de los dispositivos portátiles por medio del motor de inferencia basado en reglas.
- En un artículo publicado por Albín Rodríguez et-al. [12] se propone una metodología difusa de localización en interiores para ayudar a cuidadores a monitorear a personas mayores con problemas físicos y cognitivos en un entorno inteligente. La metodología se basa en dispositivos móviles y balizas BLE, y utiliza una aproximación lingüística difusa para lidiar con la naturaleza imprecisa de los RSSI. El caso de estudio llevado a cabo en el Smart Lab de la Universidad de Jaén demostró que la precisión de la metodología propuesta fue aproximadamente un 10 % mayor que la del enfoque no difuso, lo que demuestra la eficacia de la utilización de la lógica difusa para tratar con la imprecisión de los datos RSSI en la localización en interiores.
- En este artículo [13], se presenta una metodología para la detección temprana y fiable del comportamiento hiperactivo mediante la propuesta de un nuevo sistema llamado Smart HyBeDe. Este sistema integra dispositivos portátiles comerciales no invasivos, como brazaletes de actividad, para capturar flujos de datos de unidades de medición inerciales y sensores ópticos de frecuencia cardíaca. La metodología propuesta utiliza la lógica difusa para procesar los datos y generar una protoforma lingüística difusa que describe lingüísticamente el comportamiento hiperactivo.
- Este trabajo [14] propone un enfoque difuso para la computación en la nube-niebla (fog computing) basado en ventanas temporales difusas y agregación difusa. El objetivo es modelar y calcular datos imprecisos presentados en el paradigma de fog computing, y aplicar este enfoque a la reconocimiento de actividades en hogares inteligentes. Para demostrar la eficacia de la propuesta, se realiza un caso de estudio en el laboratorio inteligente de la Universidad de Jaén, comparando los resultados con un enfoque no difuso y mostrando las ventajas de la metodología difusa.
- Este artículo publicado por Peláez-Aguilera [15], se presenta una metodología innovadora para evaluar los flujos de ritmo cardíaco de pacientes con enferme-

dades cardíacas isquémicas que participan en programas de rehabilitación al aire libre. Estos programas utilizan dispositivos de muñeca para monitorizar en tiempo real las sesiones de rehabilitación basadas en pautas clínicas. La metodología propuesta utiliza descripciones lingüísticas relevantes (protoformas) para que el equipo de rehabilitación cardíaca pueda identificar sesiones con indicadores de interés mediante resúmenes lingüísticos. El proceso de análisis se automatiza de manera comprensible y se ofrecen descripciones cortas al equipo de rehabilitación cardíaca, que brinda comentarios a sus pacientes. La metodología se ha aplicado a datos reales proporcionados por pacientes de un programa de rehabilitación cardíaca al aire libre dirigido por el Consejo de Salud del Servicio de Salud de Andalucía (España).

- Este artículo [16], presenta un mecanismo novedoso para generar descripciones lingüísticas de series de tiempo mediante técnicas de lógica difusa y lenguaje natural. La generación de resúmenes de calidad captura las características de las series de tiempo relevantes para un usuario en una aplicación particular, y puede ser fácilmente personalizada para diferentes dominios. El enfoque se ha aplicado con éxito a la generación de descripciones lingüísticas de la inquietud en la cama de los residentes de TigerPlace, utilizado como caso de estudio para ilustrar el proceso de modelado y mostrar la calidad de las descripciones obtenidas.

- Este trabajo [32] aborda el problema de la pobreza energética, que afecta a millones de personas en la Unión Europea. Se propone el uso del paradigma Internet de las cosas (IoT) para monitorear los indicadores de pobreza energética en viviendas, como la temperatura y humedad. En lugar de soluciones basadas en datos cuantitativos y gráficos, se presenta una arquitectura de IoT basada en lógica difusa, que permite generar resúmenes lingüísticos intuitivos para que los expertos puedan tomar decisiones más eficaces.

Capítulo 4

Análisis

Durante el desarrollo de este capítulo se identifican y analizan los diferentes requisitos funcionales y no funcionales que la arquitectura planteada debe cumplir.

4.1. Requisitos funcionales

Los requisitos funcionales [33], son una lista detallada de las características o funciones que un sistema debe cumplir para satisfacer las necesidades del usuario. Estos requisitos se relacionan directamente con el comportamiento del sistema y su capacidad para realizar tareas específicas.

Se han identificado los siguientes requisitos funcionales.

Código	**Casos de uso**
CU-01	Registro de usuario
CU-02	Inicio de sesión
CU-03	Recogida de datos
CU-04	Sincronización de datos
CU-05	Eliminación de datos
CU-06	Carga de ficheros
CU-07	Descarga de datos
CU-08	Visualización de los datos
CU-09	Generación de protoformas

Tabla. 4.1: Requisitos funcionales.

CU-01. Registro de usuario

Este caso de uso se enfoca en el registro del usuario en la aplicación. La tarea esencial es proporcionar un sistema que permita la creación de una cuenta de usuario válida, que garantice la seguridad de los datos y la privacidad del usuario. Además, el sistema debe informar al usuario sobre el resultado de la operación de registro, indicando si ha sido exitosa o no.

CU-01	**Registro de usuario**
Actores	Usuario, Sistema
Pre condiciones	Ninguna
Pos condiciones	Usuario listo para utilizarse
Escenario principal	1. El usuario abre la aplicación 2. Pulsa sobre el botón de registrarse 3. Rellena los campos 4. Pulsa sobre el botón registrarse 5. El sistema envía la petición y recibe la respuesta
Escenarios alternativo	5. a.1 El usuario ya se encuentra registrado 5. a.2 El sistema informa al usuario de que ya hay una cuenta registrada con esos datos 5. b.1 La aplicación no puede comunicarse con la API 5. b.2 El sistema informa al usuario de que no ha podido completar el registro

Tabla. 4.2: CU-01 Registro de usuario.

CU-02. Inicio de sesión

Este caso de uso se corresponde con la funcionalidad de Inicio de Sesión, en la que el sistema deberá autenticar al usuario y proporcionar una respuesta en caso de éxito o fracaso en la autenticación. Es un aspecto fundamental de la arquitectura, ya que permite garantizar la seguridad de los datos y la privacidad de los usuarios. El sistema deberá validar las credenciales del usuario y permitir el acceso a la aplicación

si la autenticación es satisfactoria, de lo contrario deberá informar al usuario del error.

CU-02	**Inicio de sesión**
Actores	Usuario, Sistema
Pre condiciones	Ninguna
Pos condiciones	El usuario inicia sesión
Escenario principal	1. El usuario abre la aplicación 2. Rellena los campos 3. Pulsa sobre el botón iniciar sesión 4. El sistema envía la petición y recibe la respuesta
Escenarios alternativo	4. a.1 El usuario no se encuentra registrado 4. a.2 El sistema informa al usuario de que no hay una cuenta registrada con esos datos 4. b.1 La aplicación no puede comunicarse con la API 4. b.2 El sistema informa al usuario de que no ha podido completar el inicio de sesión

Tabla. 4.3: CU-02 Inicio de sesión.

CU-03. Recogida de datos

Este caso de uso se refiere a la captura de información generada por los dispositivos wearables y su posterior transmisión a la arquitectura del sistema. La ejecución exitosa de este proceso es esencial para el funcionamiento óptimo del sistema y su capacidad para gestionar y analizar los datos de los dispositivos wearables.

CU-04. Sincronización de datos

Este caso de uso se corresponde con la recopilación de datos generados por los dispositivos wearables. El sistema debe ser capaz de recibir y almacenar de manera eficiente los datos provenientes de los dispositivos, garantizando su integridad y exactitud. Además, es importante que el sistema informe de manera oportuna sobre

CU-03	**Recogida de datos**
Actores	Usuario, Sistema
Pre condiciones	Ninguna
Pos condiciones	Datos registrados en el servidor
Escenario principal	1. El usuario abre la aplicación del dispositivo 2. El sistema realiza la petición para obtener la clave 3. El usuario pulsa sobre el botón play para comenzar 4. El sistema empieza a recoger los datos 5. Pulsa sobre el botón stop para finalizar 6. El sistema envía la petición y recibe la respuesta
Escenarios alternativo	6. a.1 La aplicación no puede comunicarse con la API 6. a.2 El sistema informa al usuario de que no ha podido completar el inicio de sesión

Tabla. 4.4: CU-03 Recogida de datos.

el estado de la recopilación de datos, para garantizar la correcta funcionalidad de la arquitectura.

CU-04	**Sincronización de datos**
Actores	Usuario, Sistema
Pre condiciones	Token obtenido
Pos condiciones	Datos registrados en el servidor
Escenario principal	1. El sistema envía los datos al servidor 2. Se guardan los datos en el servidor
Escenarios alternativo	1. a.1 No se puede verificar el token 1. a.2 El sistema informa de que el token no es válido o no consta

Tabla. 4.5: CU-04 Sincronización de datos.

CU-05. Eliminación de datos

Este caso de uso se corresponde con la funcionalidad de eliminación de datos, donde el sistema está diseñado para permitir a los usuarios eliminar la información previamente registrada y almacenada en el sistema. Es importante destacar que, después de realizar esta acción, el sistema deberá informar al usuario sobre el éxito o fracaso de la eliminación de los datos.

CU-05	**Eliminación de datos**
Actores	Usuario, Sistema
Pre condiciones	Token obtenido
Pos condiciones	Datos eliminados del servidor
Escenario principal	1. El sistema envía la petición de eliminación 2. El sistema elimina la copia local 3. El servidor elimina los datos
Escenarios alternativo	1. a.1 No se puede verificar el token 1. a.2 El sistema informa de que el token no es válido o no consta 3. b.1 No existen los datos 3. b.2 El sistema informa de que no ha podio completar la petición

Tabla. 4.6: CU-05 Eliminación de datos.

CU-06. Carga de ficheros

Este caso de uso se corresponde con el proceso de cargar los datos en el sistema. Se espera que el sistema sea capaz de recibir y almacenar los datos en un formato adecuado para su posterior procesamiento y análisis.

CU-07. Descarga de datos

Este caso de uso se refiere al proceso de descargar los datos almacenados en el sistema. El objetivo es permitir a los usuarios acceder a los datos de forma local en

CU-06	**Carga de ficheros**
Actores	Usuario, Sistema
Pre condiciones	Ficheros disponibles
Pos condiciones	Fichero cargado
Escenario principal	1. Se selecciona el archivo a cargar 2. El sistema carga el archivo 3. El sistema comprueba el archivo
Escenarios alternativo	1. a.1 No se puede verificar el token 1. a.2 El sistema informa de que el token no es válido o no consta 2. b.1 No existen los datos 2. b.2 El sistema informa de que no ha podio completar la petición

Tabla. 4.7: CU-06 Carga de ficheros.

un formato que les permita manipularlos o analizarlos de manera adecuada para sus necesidades específicas.

CU-08. Visualización de los datos

Este caso de uso se corresponde con la representación gráfica de los datos recogidos por los dispositivos wearables. Esto permitirá a los usuarios tener una visualización clara y concisa de la información, facilitando su análisis y comprensión.

CU-09. Generación de protoformas

Este caso de uso se corresponde con la generación de las protoformas y su posterior visualización en la aplicación.

CU-07	**Descarga de datos**
Actores	Usuario, Sistema
Pre condiciones	Token obtenido
Pos condiciones	Datos descargados del servidor
Escenario principal	1. El sistema envía una petición de descarga de datos 2. El sistema espera la respuesta y obtiene los datos
Escenarios alternativo	1. a.1 No se puede verificar el token 1. a.2 El sistema informa de que el token no es válido o no consta 2. a.1 Los datos solicitados no existen 2. a.2 El sistema informa de que los datos no existen

Tabla. 4.8: CU-07 Descarga de datos.

CU-08	**Visualización de los datos**
Actores	Usuario, Sistema
Pre condiciones	Token obtenido
Pos condiciones	Datos representados
Escenario principal	1. El usuario selecciona el archivo 2. El sistema carga el archivo 3. El sistema representa los datos mediante gráficas
Escenarios alternativo	3. a.1 El archivo no contiene datos interpretables 3. a.2 El sistema informa al usuario de que no ha podido representar los datos

Tabla. 4.9: CU-08 Visualización de los datos.

CU-98	**Generación de protoformas**
Actores	Sistema
Pre condiciones	Token obtenido, Archivo cargado
Pos condiciones	Protoforma visualizada
Escenario principal	1. El sistema envía una petición para calcular la protoforma 2. El sistema espera la respuesta y representa el resultado
Escenarios alternativo	No tiene

Tabla. 4.10: CU-09 Generación de protoformas.

4.2. Requisitos no funcionales

Los requisitos no funcionales [33], describen las características del sistema en cuanto a su rendimiento, escalabilidad, seguridad, disponibilidad, usabilidad, portabilidad, compatibilidad, entre otros. Estos requisitos son esenciales para garantizar que el sistema cumpla con las necesidades del usuario de manera adecuada.

Los requisitos no funcionales identificados son los siguientes.

Código	**Casos de uso**
RNF-01	La API debe de estar documentada
RNF-02	Se debe implementar comunicaciones seguras entre las distintas partes de la arquitectura
RNF-03	Se debe de dotar a la API de protección contra los ataques más comunes
RNF-04	Se debe desplegar de manera automática y auto gestionando las dependencias
RNF-05	La API debe de tener un rendimiento aceptable

Tabla. 4.11: Requisitos no funcionales.

Capítulo 5

Diseño

En este capítulo se explicará las distintas tareas enmarcadas dentro del diseño del sistema, entre las que se encuentran la selección del tipo de base de datos y la definición de las entidades, la definición de los distintos métodos y funciones de la API REST, el sistema de archivos o las distintas vistas de la aplicación.

5.1. Arquitectura del sistema

En esta sección, se presenta un análisis de la arquitectura del sistema, incluyendo una descripción de los componentes clave, las interacciones entre ellos, y las decisiones de diseño importantes. Además, se discuten las consideraciones de rendimiento, escalabilidad, seguridad, y mantenibilidad para garantizar el éxito del sistema a largo plazo.

Elementos de la arquitectura

- **Reloj inteligente.** Este dispositivo contiene la aplicación que se encarga de registrar los datos de los diferentes sensores, para posteriormente enviarlos a un servicio ubicado en el dispositivo móvil, mediante bluetooth. Este servicio se encarga de enviar los datos a la API ubicada en el servidor. El servicio se ejecuta en la propia aplicación de Fitbit, y es debido a esto que no se pueden almacenar los datos en el dispositivo, ya que el SDK no lo permite.
- **Dispositivo móvil.** El dispositivo móvil contiene dos elementos esenciales, por un lado el servicio que se encarga de subir las datos al servidor como ya hemos comentado, y por otro lado la aplicación de gestión y visualización de los datos.

Desde esta última aplicación es desde donde el usuario puede iniciar sesión y visualizar los datos recogidos durante la sesión con el dispositivo.

- **Proxy inverso.** El proxy inverso es un elemento clave de la arquitectura. Brinda la posibilidad de establecer conexiones seguras con los clientes de la API mediante HTTPS. Además, incorpora una serie de protecciones y mitigaciones contra los principales tipos de ataques, lo cual no sería posible sin este elemento.
- **API-REST.** La API contiene los diferentes métodos necesarios para que los elementos de la arquitectura intercambien datos de forma remota entre sí.
- **Base de datos.** La base de datos se encarga de almacenar los usuarios registrados en el sistema.

Se ha realizado un esquema, Figura 5.1, que contiene los elementos de la arquitectura y su interacción.

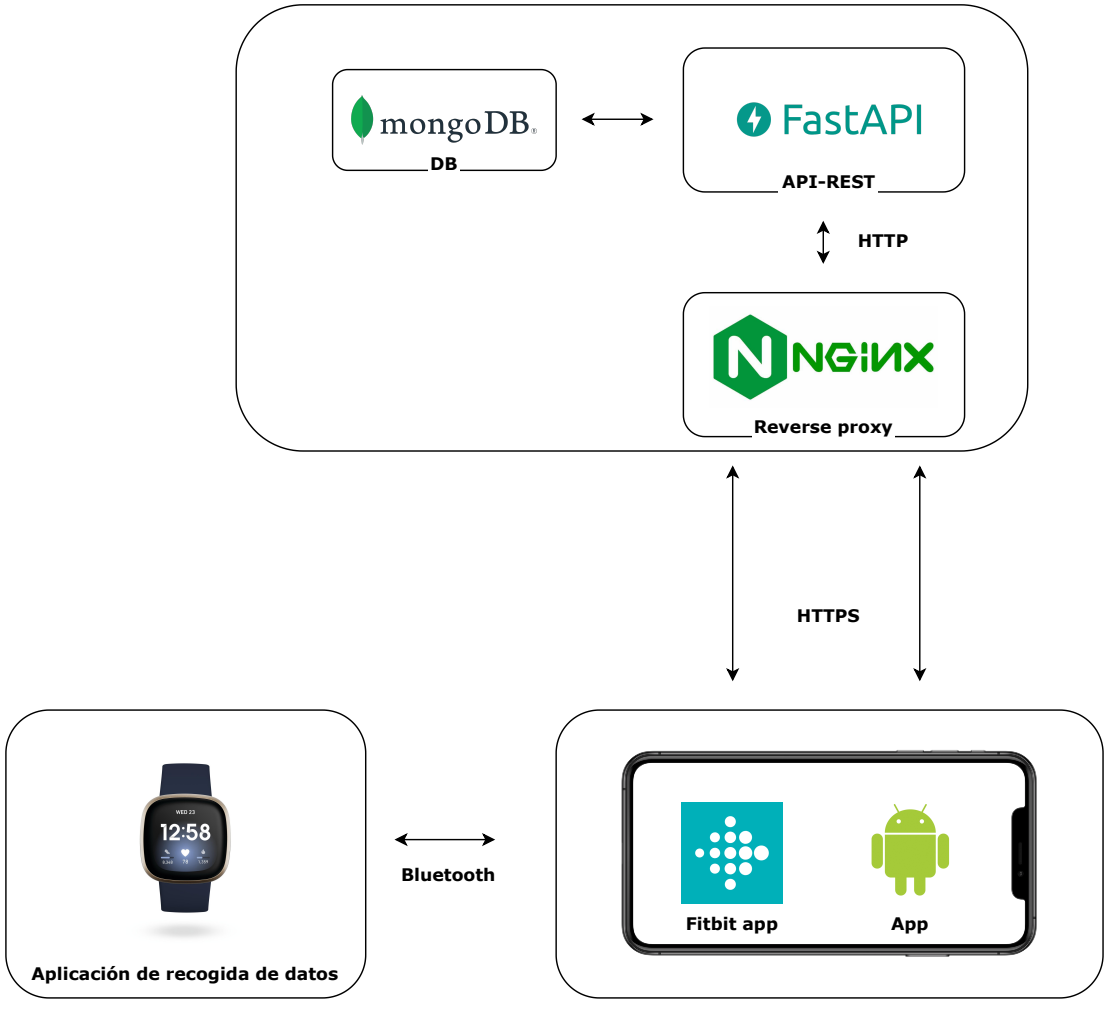

Figura 5.1: Esquema del sistema.

5.2. Diseño del modelo de datos

En esta sección se determinarán los motivos que han llevado a la elección de la base de datos. Adicionalmente, se determinarán las entidades necesarias.

En primer lugar, las entidades necesarias se corresponderán con los objetos propios de la aplicación. En este caso, se trabaja con usuarios y datos de los diferentes sensores de los dispositivos weareables. Sin embargo, y como se explicará de manera más detallada posteriormente, los datos correspondientes a los sensores de los dispositivos no son susceptibles de ser almacenados en bases de datos, puesto que son datos de alta frecuencia que no se van a aprovechar de los beneficios que implica su almacenamiento en estructuras indexadas y rígidas. Esto conlleva que los objetos que se almacenarán en la base de datos se correspondan con los datos sobre los usuarios. En este caso las operaciones que se aplicarán sobre estos datos se corresponden con CRUD, es decir, inserción, modificación, borrado y selección.

Una vez determinadas las entidades de nuestra base de datos, se va a proceder a explicar el tipo de base de datos seleccionada y las razones subyacentes a dicha elección.

Actualmente, dentro de lo que son las bases de datos, podemos diferenciar dos grandes bloques, las tipo SQL (Structured Query Language) y las NoSQL (Not only SQL). Una de las principales diferencias entre estos dos bloques son el cumplimiento de las propiedades ACID (Atomicity, Consistency, Isolation, Durability). ACID garantiza que las transacciones que se realizan serán siempre consistentes, dotando a la base de datos de una mayor robustez. Sin embargo, el cumplimiento de ACID también conlleva una perdida de rendimiento, escalabilidad y flexibilidad, que en determinadas situaciones puede ser contraproducente. El cumplimiento de todas las propiedades está garantizado en las bases de datos relacionales, o SQL. En las de tipo NoSQL, no se siempre se cumplen todas las propiedades de ACID, por lo que se suele usar otro tipo de propiedades denominadas BASE (Basically Available, Soft State, Eventually Consistent). Al ser BASE un conjunto de propiedades menos estricto que ACID, permiten que las bases de datos sean más escalables y flexibles, a costa de que las transacciones que se realicen no sean tan consistentes en comparación con ACID.

5.3. Sistema de ficheros

Como se ha comentado en la sección anterior, los datos procedentes de los distintos sensores de los dispositivos no se van a almacenar en bases de datos. Los datos se van a agrupar en sesiones, es decir, la aplicación va a generar un volumen de datos durante un determinado tiempo, que posteriormente se subirá al servidor. Esto es debido a que se genera un gran volumen de datos al utilizar un frecuencia de muestro muy alta, puesto que los datos inerciales, para que sean determinantes requieren de un gran número de muestras por segundo. Insertar cada dato generado en la base de datos, saturaría rápidamente la API y la propia base de datos, debido al elevado número de peticiones en un periodo corto de tiempo.

Un enfoque que podría ser adecuado, sería generar documentos en formato JSON, donde cada documento constaría de todos los datos de la sesión, de tal manera que las claves serían los timestamp de dichos datos. Este enfoque, si bien es cierto que reduce el número de peticiones a la base de datos, tiene asociado un problema. Dependiendo de la duración de las sesiones, los documentos generados pueden ocupar varios megas. Almacenar ese volumen de documentos en una base de datos no nos aporta nada, debido a que no se van a realizar búsquedas dentro de los documentos. Al no realizarse búsqueda dentro de las sesiones, podemos optar por otro tipo de enfoque, que consiste en guardar los datos de cada sesión en ficheros. Cuando la API recibe los datos de la sesión, se genera un fichero CSV (Comma-separated values) que se almacena en el directorio correspondiente a cada usuario en el servidor, y se clasifica según el tipo de dato, HR o inercial. Esto conlleva un tiempo de procesamiento de los datos mínimo, y puesto que se recuperan los archivos de las sesiones completos, es ideal para el sistema planteado.

5.4. API

Un punto esencial en la arquitectura planteada es el nexo de unión entre los distintos servicios y aplicaciones, para que puedan comunicarse entre si. Dentro de las diferentes pasibilidades que existen, podemos encontrar dos que destacan sobre el resto, las APIs SOAP (Simple Object Access Protocol) Y REST. Por ello, antes de determinar que tipo de servicio se ha seleccionado, se va a realizar una revisión de las principales características de SOAP y REST.

SOAP es un protocolo, por lo que se rige por unas estrictas reglas de comunicación

entre el cliente y el servidor. Utiliza únicamente XML (Extensible Markup Language) como formato de la información contenida en los mensajes. SOAP cumple las propiedades definidas mediante ACID. Además, está concebido para aplicaciones empresariales, donde la seguridad y la robustez son dos puntos clave, posicionándose en estos ámbitos por encima de REST. Para el cifrado de las comunicaciones utiliza SSL/TLS (Secure Socket Layer/Transport Layer Security) y WS-Security (Web Services Security).

REST es un estilo de arquitectura que no sigue ningún estándar, pero que debe de cumplir con una serie de propiedades. REST se caracteriza por ser establecer conexiones sin estado, es decir no almacena ninguna información relacionada con las sesiones anteriores, tratando cada una de las solicitudes de forma independiente. Es debido a esto que todos los datos necesarios para realizar las solicitudes deben estar contenidos en la propia solicitud. REST puede trabajar con más tipos de formatos en comparación con SOAP, como puede ser JSON (JavaScript Object Notation), XML o texto plano, siendo el preferido JSON. En este sentido nos proporciona una mayor flexibilidad a la hora de definir la froma de interactuar con la API. El cifrado de las comunicaciones se basa en SSL/TLS Y HTTPS (Hypertext Transfer Protocol Secure). REST está especialmente indicado para ser utilizado por dispositivos móviles y otros tipos de dispositivos de baja potencia.

En base a lo anterior, se ha seleccionado REST, ya que es la que más se adecua a las necesidades del sistema planteado. A continuación, se especifican los distintos recursos que se han definido.

En primer lugar, veamos los recursos que se han definido para la gestión de los usuarios y de los dispositivos wearables.

Método	**URL**	**Descripción**
POST	/token	Permite obtener su token Bearer a los clientes
GET	/users/me	Devuelve la información de los clientes
POST	/newUser	Permite añadir nuevos usuarios al sistema
GET	/tokenWatch	Permite obtener tokens a los dispositivos wearables

Tabla. 5.1: Recursos definidos para la gestión de los usuarios.

A continuación, los recursos que se han definido para los datos de los sensores.

Cabe destacar que la tabla superior se corresponde con los recursos definidos para un único tipo de sensor, pero en la API se han definido para el resto de sensores.

Método	URL	Descripción
POST	/hrData	Permite subir documentos en formato JSON
DELETE	/hrData	Elimina todos los datos correspondientes al ritmo cardíaco del usuario
GET	/hrDataWatch	Devuelve la información de los clientes
POST	/hrDataCSV	Permite subir documentos cuyo contenido está en formato CSV
GET	/listHrDataFiles	Permite obtener la lista de documentos asociados al usuario
GET	/hrData/file_name	Permite obtener el contenido de un documento en formato CSV
GET	/hrFileDownload/file_name	Permite descargar los ficheros en formato CSV
GET	/protofromHr/file_name	Permite calcular la protoforma asociada

Tabla. 5.2: Recursos definidos para la gestión de los datos.

5.5. Diseño de la aplicación

Las GUI (Graphical user interface) son un tipo de interfaz de usuario que utiliza elementos visuales para permitir que el usuario interactúe con una aplicación o sistema. Estos elementos visuales pueden incluir ventanas, iconos, menús, botones y otros elementos gráficos. La ventaja de las GUI es que son más fáciles de usar y entender para los usuarios, ya que proporcionan una forma visual de interactuar con la aplicación. Además, las GUI suelen ser más atractivas y agradables de usar, lo que puede mejorar la experiencia del usuario. Esto las hace ideal para la arquitectura planteada, ya que la aplicación está destinada al usuario promedio.

Para planificar y diseñar la estructura y el contenido existen una serie de herramientas y técnicas que nos permiten definirlas con distintos grados de detalle antes de proceder a su implemenetación en las aplicaciones finales. Las más extendidas son Wireframe, Mockup y prototipos.

Un wireframe es una representación gráfica de una página web, aplicación o sistema que se utiliza para planificar y diseñar la estructura y el contenido de un producto digital. Los wireframes se utilizan para visualizar el contenido y la disposición de los elementos en una página o pantalla, y suelen ser utilizados por diseñadores y desarrolladores para probar y validar conceptos de usuario y navegación antes de comenzar a trabajar en la interfaz de usuario final.

Los wireframes son generalmente bocetos básicos que se utilizan para planificar la disposición de los elementos en una página o pantalla, y pueden ser creados a mano o utilizando herramientas de diseño gráfico. A menudo se utilizan para probar distintos diseños y disposiciones antes de comenzar a trabajar en el diseño final, y pueden ser modificados fácilmente para hacer cambios en la estructura o el contenido de la página o pantalla. Los wireframes suelen ser utilizados en conjunción con otros tipos de diseño, como prototipos y maquetas, para planificar y probar la interfaz de usuario de un producto digital.

Un mockup es una representación gráfica detallada de un producto o interfaz de usuario, que se utiliza para probar y validar el diseño y la funcionalidad de un producto digital antes de comenzar su desarrollo. Los mockups suelen ser utilizados para mostrar cómo se verá y cómo funcionará un producto en su forma final, y pueden ser creados a mano o utilizando herramientas de diseño gráfico.

Los mockups suelen ser más detallados que los wireframes, ya que incluyen información adicional sobre el diseño y la apariencia final del producto. Pueden incluir elementos como colores, fuentes, imágenes y otros detalles de diseño, y suelen ser utilizados para probar la usabilidad y la experiencia del usuario. Los mockups también pueden ser utilizados para probar distintos diseños y disposiciones antes de comenzar a trabajar en el diseño final, y pueden ser modificados fácilmente para hacer cambios en el diseño o la funcionalidad del producto.

Un prototipo es una versión preliminar de un producto o sistema, que se utiliza para probar y validar el diseño y la funcionalidad antes de su lanzamiento. Los prototipos suelen ser utilizados para probar la usabilidad y la experiencia del usuario, y pueden ser creados a mano o utilizando herramientas de diseño y desarrollo.

Hay varios tipos de prototipos, que pueden variar en términos de su nivel de detalle y complejidad. Los prototipos más simples pueden ser simplemente bocetos o maquetas de papel que muestren la disposición y el diseño de los elementos en una página o pantalla. Otros prototipos pueden ser más detallados y pueden incluir funcionalidades básicas, como enlaces y transiciones entre páginas o pantallas. Los prototipos más avanzados pueden incluir funcionalidades completas y pueden ser utilizados para probar el rendimiento y la estabilidad de un producto o sistema.

Los prototipos suelen ser utilizados en conjunción con otros tipos de diseño, como wireframes y mockups, para probar y validar el diseño y la funcionalidad de un producto digital antes de comenzar su desarrollo completo.

En base a lo expuesto con anterioridad y dado que la aplicación no está sujeta a la

aprobación por parte de un cliente, se ha optado por el uso de wireframe para diseñar la estructura general de la aplicación y del contenido ubicada en ella. Esta elección reduce el tiempo invertido en la realización de mockups y de prototipos, que dependiendo del nivel de detalle pueden ser tareas que consuman una parte importante del tiempo total disponible para el proyecto. Si el proyecto estuviera sujeto a la aprobación por parte de un cliente, sería recomendable, e incluso necesario, la realización de prototipos, que reflejen como se comportará la aplicación respecto a las funcionales que ofrece y el estilo de diseño empleado.

Como se puede apreciar en las Figuras 5.2 y 5.3, se muestra el wireframe de la pantalla de inicio de sesión, registro, principal y gráficos. Este primer diseño sirve para establecer la ubicación de los principales elementos dentro de la aplicación.

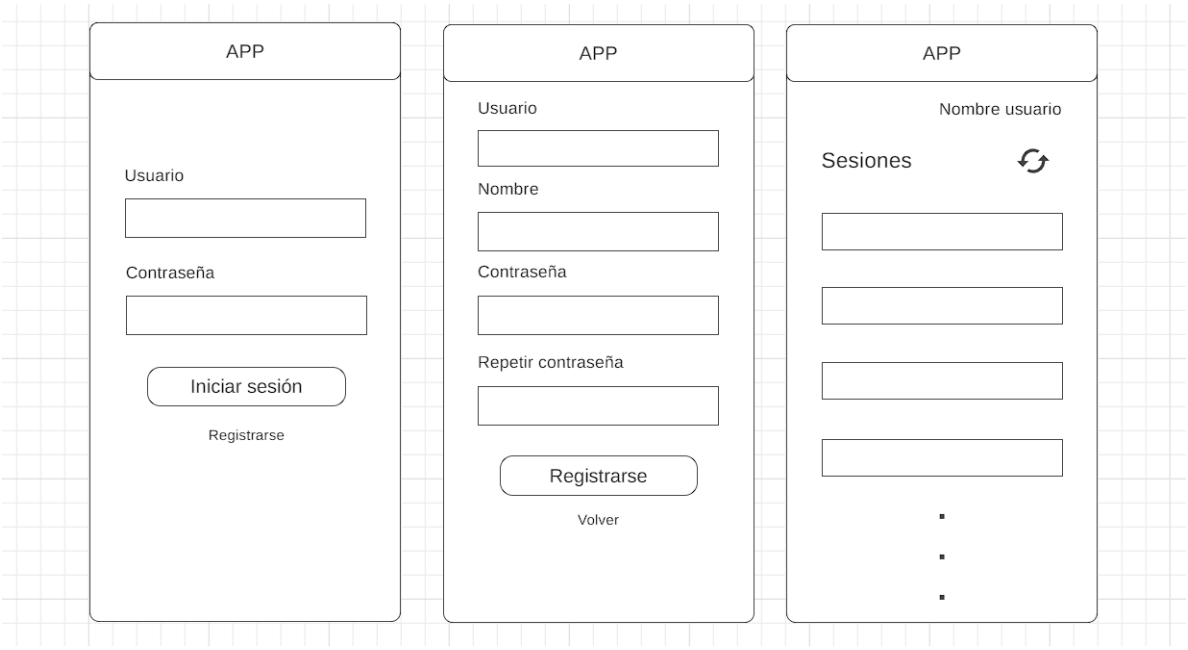

Figura 5.2: Wireframe de las pantallas de login, registro y principal.

5.6. Protoformas lingüísticas

Esta sección se enfoca en las proformas lingüísticas difusas, una herramienta utilizada en la teoría de conjuntos difusos para representar la incertidumbre o imprecisión en el lenguaje natural. Las proformas lingüísticas difusas permiten la expresión de términos vagos o imprecisos de manera más precisa y cuantificable, lo que las hace útiles en diversos campos, como la inteligencia artificial, la toma de decisiones, la clasificación de datos y la modelización de sistemas complejos.

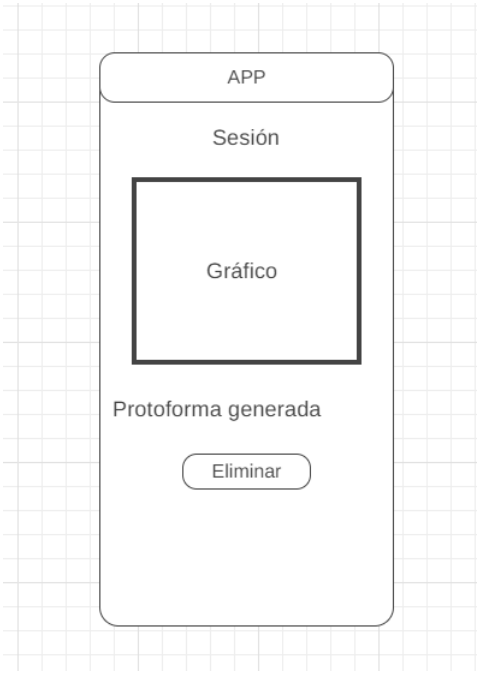

Figura 5.3: Wireframe de las pantalla de gráficos.

5.6.1. Variables lingüísticas difusas

En este trabajo, se utilizarán tres variables lingüísticas difusas para describir los flujos de datos del ritmo cardíaco y la aceleración. Cada una de estas variables se describirá mediante conjuntos difusos que indican el grado de pertenencia de cada valor a una categoría específica.

Para la variable del **ritmo cardíaco** (HR, por sus siglas en inglés), se utilizarán los valores lingüísticos bajo, moderado y alto para describir los diferentes niveles de frecuencia cardíaca. El conjunto difuso bajo incluirá valores de 0 hasta 60 latidos por minuto, el conjunto moderado incluirá valores entre 60 y 110 latidos por minuto, el conjunto alto incluirá valores entre 90 y 150 latidos por minuto y el conjunto muy alto incluirá valores de más de 130 latidos por minuto. Estos rangos se basan en los valores normales de frecuencia cardíaca para adultos en reposo y durante el ejercicio físico intenso.

Los siguientes conjuntos difusos son definidos mediante funciones de pertenencia trapezoidales Y triangulares sobre el indicador de la frecuencia cardíaca, expresado en pulsaciones por minuto.

- µHR frecuencia_cardiaca[’baja’]: TS(0,0,50,60)
- µHR frecuencia_cardiaca[’moderada’]: TS(50,80,100)

- µHR frecuencia_cardiaca['alta']: TS(80,120,140)
- µHR frecuencia_cardiaca['muy_alta']: TS(120,140,200,201)

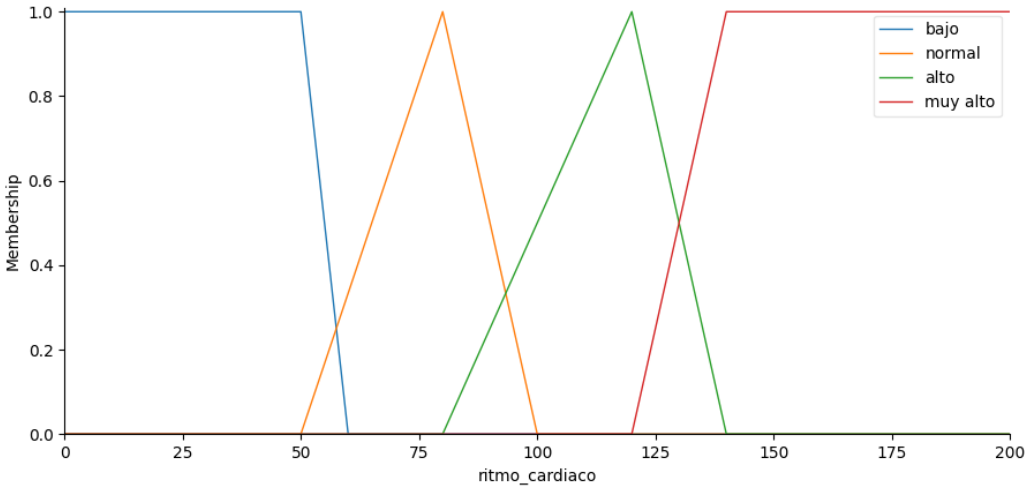

Figura 5.4: Funciones de pertenencia de los términos difusos de la variable frecuencia cardíaca.

Para la **variable de aceleración**, se utilizarán los valores lingüísticos bajo, moderado y alto para describir los diferentes niveles de aceleración. El conjunto difuso bajo incluirá valores entre $0\ y\ 14\,m/s^2$, el conjunto moderado incluirá valores entre $11\ y\ 20\,m/s^2$, y el conjunto alto incluirá valores de más de $16\,m/s^2$. Estos rangos se basan en los valores típicos de aceleración para actividades cotidianas como caminar, correr y saltar, teniendo en cuenta la fuerza ejercida por la aceleración de la Tierra.

Los siguientes conjuntos difusos son definidos mediante funciones de pertenencia trapezoidales sobre el indicador de la aceleración, expresado en metros por segundo.

- µA aceleracion['bajo']: TS(0,0,12,18)
- µA aceleracion['moderado']: TS(12,18,25,28)
- µA aceleracion['alto']: TS(25,28,50,51)

Se propone el uso de varios cuantificadores, que se definen mediante funciones sigmoidales y gausianas.

- µQ MOT(50, 0.1)
- µQ HOT(50, 0.1)

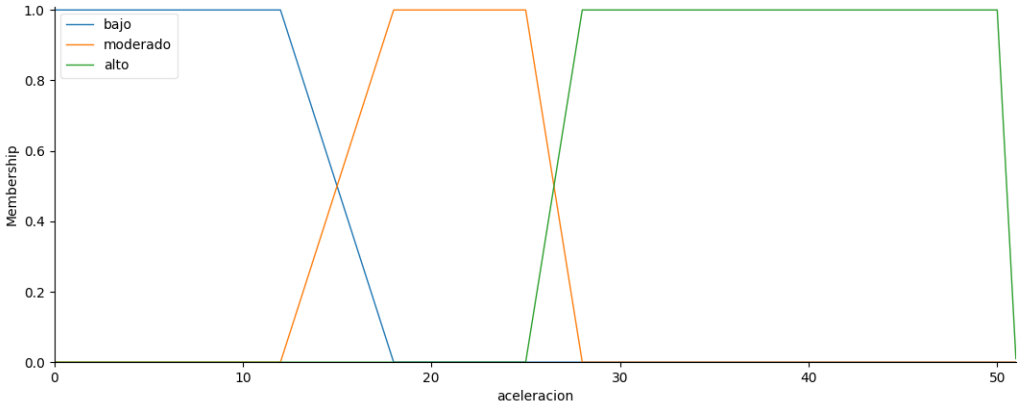

Figura 5.5: Funciones de pertenencia de los términos difusos de la variable aceleración.

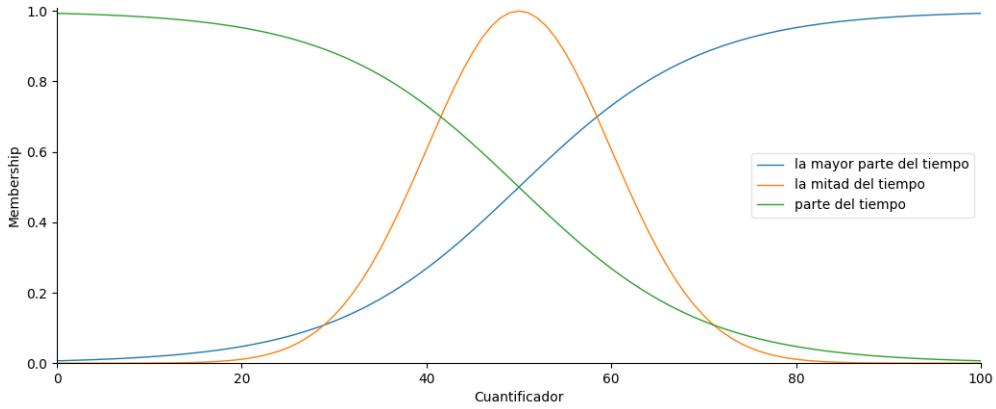

Figura 5.6: Funciones de pertenencia de los cuantificadores difusos.

- µQ POT(50, 0.1)

De acuerdo con los conjuntos difusos definidos anteriormente, se pueden definir protoformas lingüísticas como *La mayor parte del tiempo la frecuencia cardiaca es normal*.

Capítulo 6

Implementación

La implementación es una etapa crucial en cualquier proyecto, ya que implica la puesta en práctica de las ideas y los conceptos que se han desarrollado en las fases previas. En este sentido, esta sección se centrará en la implementación de la arquitectura propuesta, basada en dispositivos fitbit y dispositivos móviles. Se abordarán aspectos como el estudio de alternativas para la selección de los dispositivos y plataformas más adecuados, la utilización de la API-REST para la comunicación entre el dispositivo y la aplicación, el uso de un proxy inverso para mejorar la seguridad y la eficiencia del sistema, la utilización de contenedores para facilitar el despliegue y la gestión del sistema y el uso de protoformas lingüísticas para describir los flujos de datos.

6.1. Estudio de alternativas y viabilidad

En está sección se exponen algunas alternativas de lenguajes de programación, plataformas y frameworks que se han considerado para el desarrollo de la arquitectura.

Durante la fase inicial de diseño de la arquitectura se planteó la posibilidad de desarrollar una aplicación híbrida mediante el uso de frameworks específicos para tal fin, en lugar de realizar una aplicación nativa.

Por un lado, las aplicaciones híbridas son aquellas que combinan características de las aplicaciones nativas y las aplicaciones web. Estas aplicaciones suelen estar desarrolladas con tecnologías web como HTML, CSS y JavaScript, y se ejecutan en un navegador web dentro de un contenedor nativo en el dispositivo. Las aplicaciones híbridas tienen la ventaja de poder ser desarrolladas una sola vez y ejecutarse

en múltiples plataformas, pero pueden tener un rendimiento ligeramente inferior a las aplicaciones nativas.

Característica	**Aplicaciones nativas**	**Aplicaciones híbridas**
Desarrollo	Requiere lenguaje de programación específico para cada plataforma (por ejemplo, Swift para iOS y Java para Android)	Se puede utilizar un solo lenguaje de programación (como HTML, CSS y JavaScript) para múltiples plataformas
Rendimiento	Mayor rendimiento debido a que están diseñadas específicamente para la plataforma	Menor rendimiento en comparación con las aplicaciones nativas debido a que están en un entorno web y se ejecutan dentro de un contenedor
Experiencia de usuario	Mayor nivel de interactividad y personalización debido a la capacidad de acceder a las funciones del hardware y software del dispositivo	Pueden parecer menos nativas y tener una experiencia de usuario menos personalizada
Actualizaciones	Las actualizaciones se pueden implementar más rápido y sin depender de una tienda de aplicaciones	Las actualizaciones pueden requerir la aprobación de una tienda de aplicaciones, lo que puede retrasar su lanzamiento
Costo de desarrollo	Mayor costo debido a la necesidad de programar para múltiples plataformas	Menor costo debido a que se puede utilizar un solo lenguaje de programación y una base de código compartida para múltiples plataformas
Tiempo de desarrollo	Mayor tiempo debido a la necesidad de programar para múltiples plataformas y personalizar la experiencia del usuario	Menor tiempo debido a que se puede utilizar un solo lenguaje de programación y una base de código compartida para múltiples plataformas
Acceso a características del dispositivo	Mayor acceso a las funciones del hardware y software del dispositivo, como cámara, GPS y notificaciones	Menor acceso a las funciones del hardware y software del dispositivo en comparación con las aplicaciones nativas

Tabla. 6.1: Comparación de aplicaciones nativas vs híbridas.

Por otro lado, las aplicaciones nativas son aquellas que están desarrolladas para un sistema operativo específico y se ejecutan directamente en el dispositivo. Estas aplicaciones suelen tener un rendimiento más alto y una mayor integración con el sistema operativo y el hardware del dispositivo, ya que están diseñadas específicamente para él.

Dado la disponibilidad de medios, conocimientos previos en el desarrollo de aplicaciones nativas, y la alta tasa de penetración en el mercado de dispositivos basados en Android, se optó por desarrollar una aplicación nativa. Adicionalmente, dada la coexistencia de dos lenguajes de programación para el desarrollo de aplicaciones nativas, en este caso Java y Kotlin, se optó por este último. Esta elección se debe a que Kotlin es el lenguaje que Google indica para nuevos desarrollos, ya que aunque es posible desarrollar aplicaciones con Java todavía, Kotlin viene a reemplazar a Java, manteniendo Java únicamente para brindar compatibilidad con aplicaciones ya existentes.

En lo referente a la construcción de la API-REST, existen múltiples lenguajes de programación que permiten su creación, si bien el nivel de dificultad de dicha tarea puede verse enormemente afectado por dicha elección. Dentro de las distintas alternativas encontramos Python. Python es un lenguaje de programación de propósito general de alto nivel. Fue creado con el objetivo de proporcionar un lenguaje de programación fácil de leer y escribir.

Una de las principales ventajas de Python es su sintaxis clara y concisa, que permite a los programadores escribir código de manera rápida y eficiente. Python también cuenta con una gran cantidad de bibliotecas y módulos preconstruidos que pueden utilizarse para realizar tareas comunes, como la manipulación de datos, el procesamiento de texto y la conexión con bases de datos.

Python es utilizado ampliamente en aplicaciones de desarrollo web, ciencia de datos, análisis de datos y automatización de tareas. Es un lenguaje de programación muy popular en la comunidad de desarrolladores y tiene una gran cantidad de recursos y documentación disponibles en línea.

Una de las principales deficiencias de Python reside en que es menos eficiente que otros lenguajes de programación, como pueden ser C o Java.

Dado que la API-REST planteada para la arquitectura no va a realizar uso de operaciones altamente costosas a nivel computacional, Python es una opción ideal, al existir una gran cantidad de frameworks para el desarrollo de API RESTs. Algunos de los más populares incluyen:

- **Flask [34]**. Es un framework ligero y extensible que se enfoca en la simplicidad. Es fácil de configurar y utilizar, y es ideal para aplicaciones pequeñas y medianas.
- **Django REST framework [35]**. Es una extensión del popular framework Django para el desarrollo de aplicaciones web. Proporciona un conjunto completo de

herramientas para el desarrollo de API REST, incluyendo serializadores, autenticación y permisos.

- **Falcon [36]**. Es un framework de alto rendimiento y diseñado para aplicaciones REST y HTTP en general. Ofrece una gran flexibilidad y es ideal para aplicaciones con altas demandas de rendimiento.

- **Pyramid [37]**. Es un framework versátil y extensible que se puede utilizar para el desarrollo de aplicaciones de todos los tamaños. Proporciona una amplia gama de herramientas y opciones de configuración para adaptarse a diferentes requisitos de desarrollo.

- **FastAPI [38]**. Es un framework de alto rendimiento y fácil de usar que se enfoca en la simplicidad y la velocidad. Utiliza la especificación OpenAPI para generar la documentación de la API de manera automática y proporciona integración con diferentes librerías de validación y serialización.

En este caso se ha seleccionado FastAPI, ya que proporciona una serie de herramientas muy útiles como puede ser Pydantic, para la validación de tipos o Swagger, para la generación de la documentación de la API.

6.2. Cliente Fitbit

En esta sección se hará una revisión de como se ha desarrollado el cliente ubicado en los dispositivos Fitbit, y las diferentes partes que lo conforman.

En primer lugar se va a exponer el conjunto de herramientas que nos proporciona Fitbit para el desarrollo en su plataforma.

- El SDK (Software Development Kit) de Fitbit es un conjunto de herramientas y recursos que permite a los desarrolladores crear aplicaciones y servicios para dispositivos Fitbit. El SDK incluye una API (Application Programming Interface) que proporciona acceso a datos y funcionalidades de los dispositivos Fitbit, así como documentación y ejemplos de código para ayudar a los desarrolladores a integrar sus aplicaciones con Fitbit.

- Fitbit Studio, Figura 6.1, es un entorno de desarrollo en línea (IDE) para el desarrollo de aplicaciones para dispositivos Fitbit. Permite a los desarrolladores crear

y probar aplicaciones de manera rápida y sencilla, utilizando herramientas visuales y una interfaz intuitiva. Fitbit Studio también proporciona acceso a la API de Fitbit y a la documentación del SDK para facilitar el desarrollo de aplicaciones.

- Fitbit OS Simulator, Figura 6.2, es un programa que simula el sistema operativo de un reloj inteligente Fitbit. El simulador se utiliza para desarrollar, probar y depurar aplicaciones para relojes Fitbit sin la necesidad de un dispositivo físico.

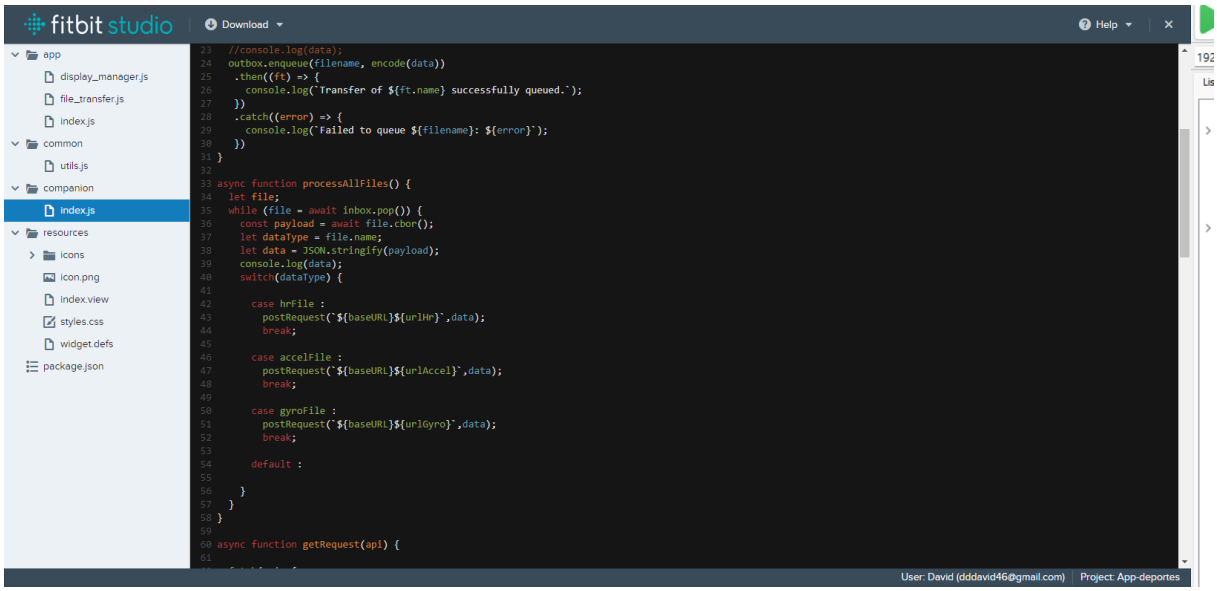

Figura 6.1: Fitbit Studio.

Una vez revisado el conjunto de herramientas que nos proporciona Fitbit, se va a proceder a explicar como se organizan los proyectos en la plataforma.

Cada proyecto cuenta con una serie de carpetas, cuya finalidad es la siguiente.

- App. Contiene el código fuente de la aplicación que se ejecuta en el reloj o pulsera inteligente.
- Companion. Contiene la lógica que se ejecuta en el teléfono móvil.
- Resources. Contiene los recursos necesarios para la aplicación, como imágenes, sonidos o la definición de la interfaz.
- Common. Contiene el código común para la aplicación del dispositivo como para la pulsera o reloj.
- Settings. Contiene la configuración de la aplicación.

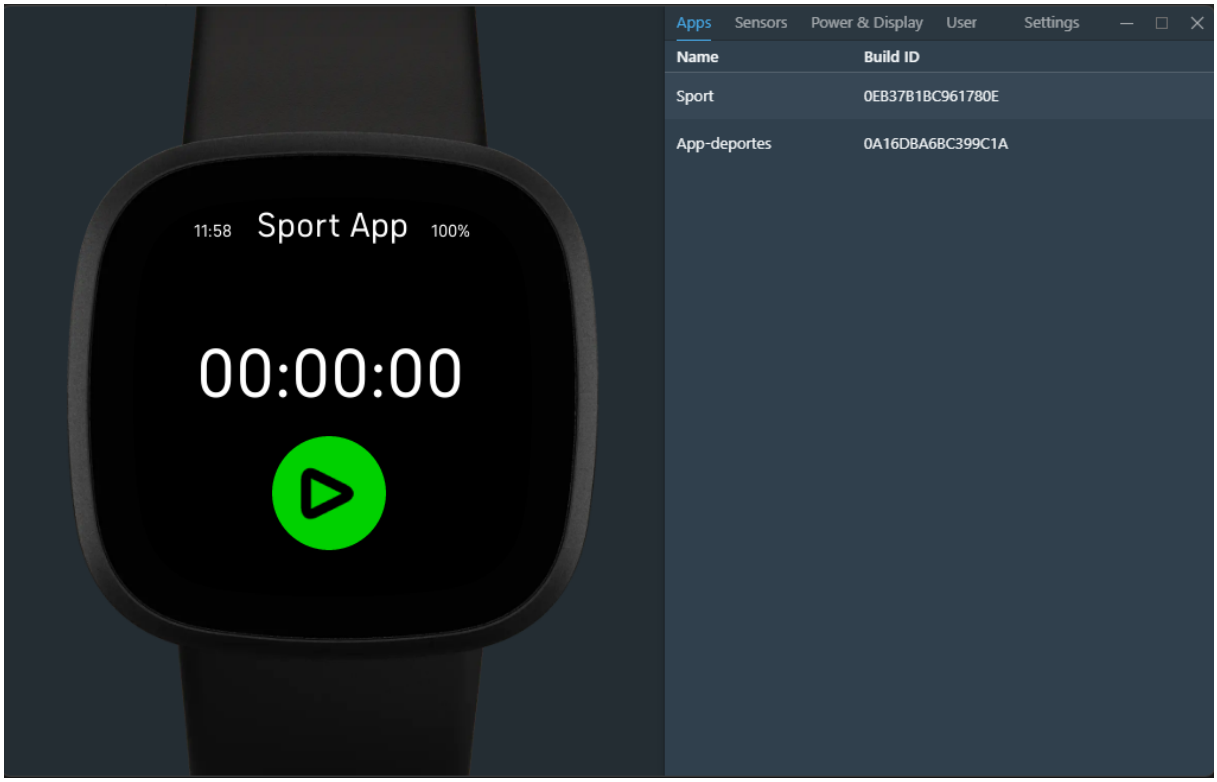

Figura 6.2: Fitbit OS Simulator.

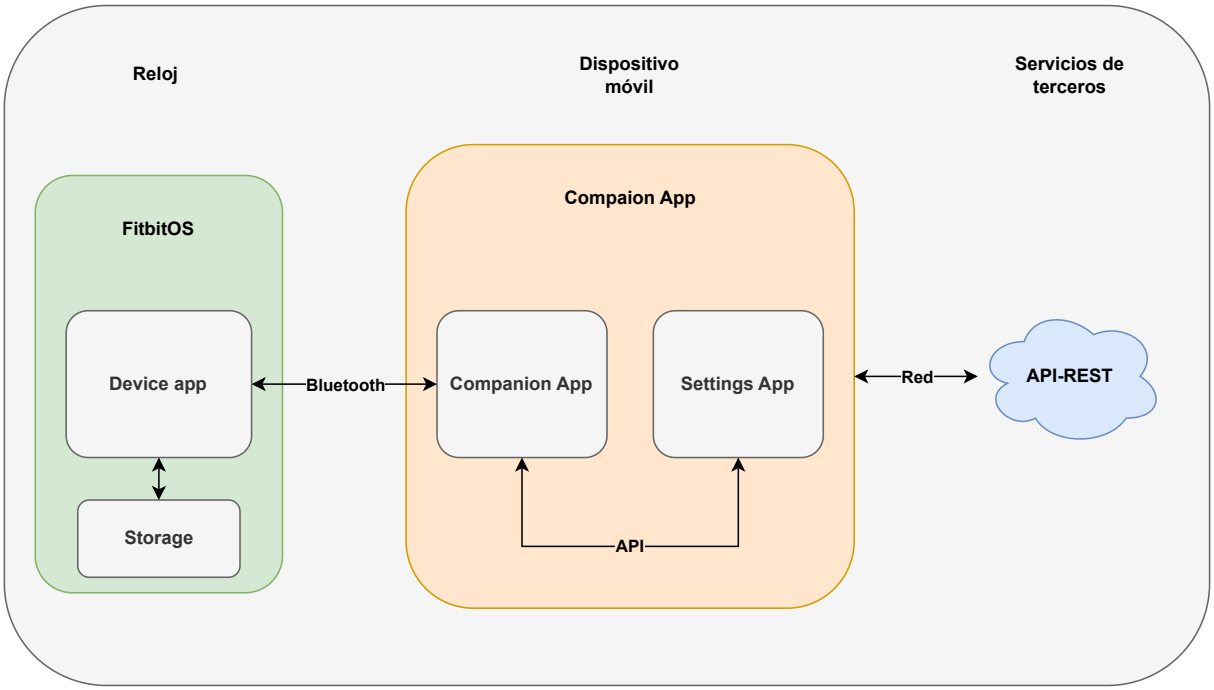

Figura 6.3: Arquitectura de Fitbit.

Como podemos observar, tenemos una carpeta principal que se corresponde con el la aplicación que se ejecutará en la pulsera o reloj, pero adicionalmente también podemos añadir lógica en la carpeta companion, que se ejecutará en el teléfono móvil. Esto es debido a que si bien se puede desarrollar una aplicación independiente del teléfono, está no tendrá la capacidad de conectarse con servicios externos, como puede ser nuestra API. Esto es debido a que si bien el reloj dispone de capacidades inalámbricas Wi-Fi y Bluetooth, carece de GSM, siendo incapaz de comunicarse con servicios externos si no dispone de señal Wi-Fi. Es debido a esto, que para el uso planteado, es necesario desarrollar cierta lógica a emplazar en el dispositivo, para que pueda conectarse con nuestra API. Esto se puede apreciar en la Figura 6.3.

Una vez explicada la necesidad de porque se debe desarrollar una aplicación ubicada en el reloj, y otra parte ubicada en teléfono, se va a proceder a explicar las principales características de ambas aplicaciones.

En primer lugar comenzamos con la aplicación propia del reloj. Esta cuenta con una serie de características principales.

- Botones para iniciar, pausar y finalizar la sesión.
- Indicador del estado de la batería.
- Indicador de la hora actual.
- Recogida de datos de ritmo cardíaco, aceleración y rotación.
- Escritura de la información en ficheros y envío a la aplicación en el teléfono.

Como se puede apreciar, la aplicación está concebida para resultar intuitiva, facilitando su uso por parte de los usuarios. La pantalla principal está formada por el botón de inicio, la hora y el indicador de la batería 6.4. La aplicación empieza a recoger los datos cuando el usuario presiona el botón para comenzar la sesión. En ese momento, empieza la recogida de datos del ritmo cardíaco, la aceleración y la rotación, mientras que la aplicación muestra dos botones nuevos, una para pausar la sesión y otro para finalizarla 6.5. La frecuencia a la que se recogen los datos del ritmo cardíaco se han fijado en una muestra por segundo, mientras que para los datos de los otros sensores se ha optado por recoger 30 muestras por segundo, ya que este tipo de datos están sujetos a una gran cantidad de variaciones en cortos periodos de tiempo.

```
1
2 if (Gyroscope) {
3   const gyro = new Gyroscope({ frequency: 30, batch: 1800 });
4   gyro.addEventListener("reading", () => {
```

```
    var item;
    if(fs.existsSync(gyroFile)){
      let content = fs.readFileSync(gyroFile,"cbor");
      item = content;
    }else{
      item = {};
      item.data = [];
    }
    let size = gyro.readings.timestamp.length, index = 0;
    while( size-- ) {
      let dataGyro = {};
      dataGyro.time = gyro.readings.timestamp[index];
      dataGyro.x = gyro.readings.x[index];
      dataGyro.y = gyro.readings.y[index];
      dataGyro.z = gyro.readings.z[index];
      item.data.push(dataGyro);
      index ++;
    }
    fs.writeFileSync(gyroFile,item,"cbor");
  })
  sensors.push(gyro);
}
```

Listado 6.1: Recogida de datos del giroscopio.

```
async function postRequest(api,jsonData) {
    fetch(api, {
      method : "POST",
      headers : myHeadersAuth,
      body: jsonData
    })
    .then( response => {
        return response.json();
    })
    .then( data => {
        console.log(data);
    })
    .catch ((err) => {
        console.error(err);
    })
}
```

Listado 6.2: Función para enviar los datos a la API.

6.3. Cliente Android

Una vez explicado en que consiste la aplicación que se encarga de recoger los datos, vamos a proceder con la aplicación para visualizar y gestionar dichos datos.

Como ya se ha explicado anteriormente, se decidió realizar un desarrollo nativo, en este caso para la plataforma de Android. Esta aplicación le permite al usuario visualizar

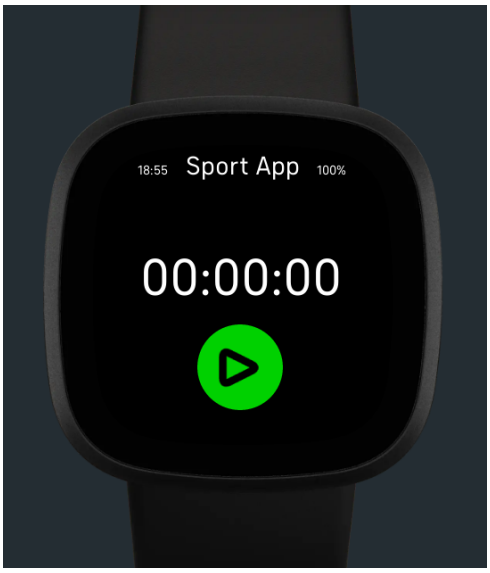

Figura 6.4: Pantalla principal de la aplicación.

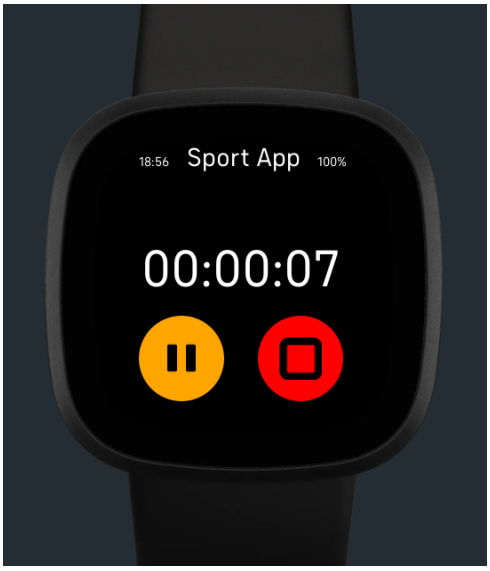

Figura 6.5: Pantalla secundaria de la aplicación.

y modificar los datos recogidos mediante el reloj inteligente.

La aplicación consta de varias funcionalidades, como pueden ser el inicio y registro de usuarios, sincronización, consulta y eliminación de sesiones.

6.3.1. Inicio de sesión

El proceso de inicio de sesión se lleva a cabo mediante peticiones realizadas con la librería OkHttp [39], utilizando las credenciales proporcionadas por el usuario. Una vez que las credenciales son verificadas correctamente por el servidor, se recibirá

un JSON Web Token (JWT) [40] como respuesta. Este token será utilizado en las siguientes peticiones para autenticar al usuario y proporcionar acceso a los recursos solicitados.

Para garantizar un rendimiento óptimo y evitar bloqueos en el hilo principal, las peticiones se realizarán mediante corrutinas en un contexto especial para operaciones de tipo IO (Input/Output). Esto significa que las peticiones se ejecutarán de manera asíncrona y en un hilo separado, lo que permitirá a la aplicación continuar funcionando sin interrupciones mientras se realiza la petición.

Es importante destacar que las peticiones realizadas con JWT en la cabecera proporcionan una capa adicional de seguridad, ya que el token es un identificador único y seguro que solo puede ser utilizado por el usuario autenticado. De esta manera, se asegura que solo el usuario autorizado tenga acceso a los recursos protegidos.

En la Figura 6.6 se puede apreciar el diseño final de la aplicación. Como se puede observar está basado en el wireframe presentado con anterioridad. En caso de que se produzca un error

En caso de ocurrir un error durante el proceso de inicio de sesión, ya sea por credenciales incorrectas o un problema de conexión con el servidor, la aplicación informará al usuario mediante un *popup*. Este mensaje de error permitirá al usuario saber que ha ocurrido un problema y, en caso de ser necesario, corregir las credenciales proporcionadas o solucionar el problema de conexión antes de intentar iniciar sesión de nuevo.

Esta medida es importante para garantizar una experiencia de usuario adecuada y para informar al usuario sobre el estado de la aplicación y la operación que está realizando.

6.3.2. Registro

La sección de registro es un proceso que permite a los usuarios crear una cuenta en la aplicación, para poder acceder a sus funcionalidades y recursos. Este proceso consiste en la captura de información esencial del usuario, como su nombre, correo electrónico y contraseña, y la verificación de dicha información antes de crear la cuenta.

La Figura 6.7 presenta la pantalla de registro accesible desde la aplicación.

Figura 6.6: Pantalla de inicio de sesión.

6.3.3. Principal

La pantalla principal de la aplicación es una interfaz intuitiva y fácil de usar que permite a los usuarios visualizar los archivos generados debido al uso de la aplicación del dispositivo wearable.

Además, la pantalla principal incluye un botón de sincronización que permite al usuario mantener los archivos actualizados y sincronizados con el servidor. Esta función permite descargar o subir los archivos que aún no hayan sido sincronizados con el servidor.

Las tareas de sincronización se realizarán mediante corrutinas, lo que garantiza una ejecución eficiente y sin interrupciones en la interfaz de usuario. Además, las corrutinas permiten manejar de manera efectiva los errores y las situaciones de fallo, informando al usuario de manera clara y precisa sobre el estado de la sincronización.

Como se puede observar el la Figura 6.8, los diferentes tipos de datos que se ge-

Figura 6.7: Pantalla de registro.

neran can acompañados de un icono distintivo, lo que permite a los usuarios identificar de manera rápida los datos contenidos en el fichero.

6.3.4. Gráficas

La pantalla asociada a cada archivo de datos que se carga es una interfaz intuitiva que permite a los usuarios visualizar los datos en una forma clara y fácil de entender, mediante el uso de gráficas.

El proceso de carga de los datos se realiza mediante corrutinas en el contexto para IO, lo que garantiza una ejecución eficiente y sin interrupciones en la interfaz de usuario. Esto permite a los usuarios ver los datos de forma rápida y sin tener que esperar a que se completen las tareas de procesamiento.

Una vez que los datos han sido cargados, son procesados y representados en la gráfica mediante corrutinas en el contexto para tareas de alto impacto. Posteriormente se envía una petición a la API para que calcule la protoforma lingüística asociada. Esto permite que la aplicación maneje de manera efectiva los procesos de alto impacto, sin

Figura 6.8: Pantalla principal.

interrumpir la experiencia del usuario ni afectar el rendimiento de la aplicación.

6.4. API-REST

Como ya se ha introducido con anterioridad, para la construcción de la API se ha utilizado el framework web FASTAPI [38].

La API-REST que se ha construido cuenta con una serie de características inherentes al propio framework utilizado, como puede ser la generación de la documentación de la API o la validación a automática de de tipos. A continuación, se van exponer los principales puntos.

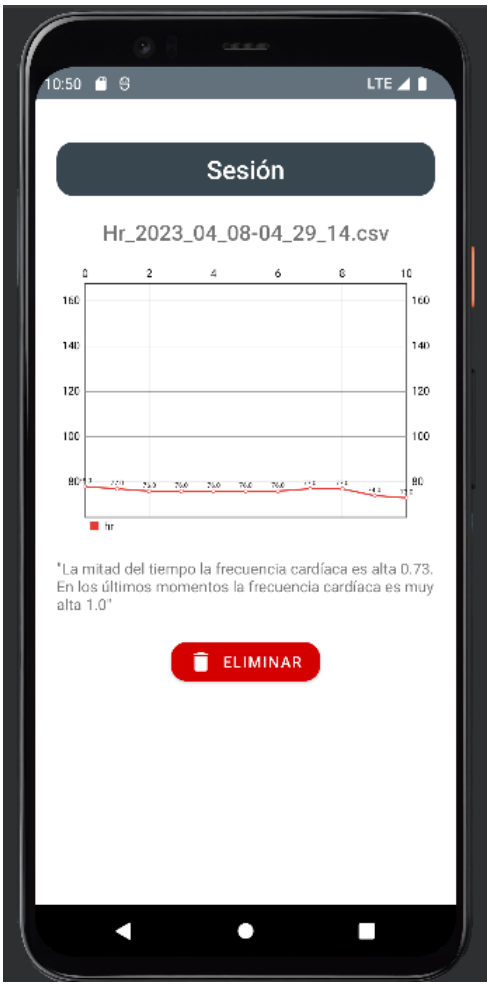

Figura 6.9: Pantalla de Gráficas.

6.4.1. Arquitectura de FASTAPI

En FASTAPI se pueden crear clases mediante Pydantic para representar los datos que recibirán las API. Pydantic es una librería de validación y serialización de datos que permite definir modelos de datos con tipos y validaciones. La utilización de estos modelos en FASTAPI hace que la validación de datos y la transformación de estos se realicen automáticamente.

La definición de rutas en FASTAPI se hace a través de funciones metadefinidas con métodos HTTP, por ejemplo, @app.get('/ruta'). Dentro de estas funciones, se puede especificar el tipo de datos que se van a recibir y los datos que se van a devolver, utilizando los modelos Pydantic definidos previamente. Además, se puede incluir lógica de negocio y operaciones con bases de datos u otros servicios.

En cuanto a las rutas, se pueden definir de forma individual o agruparlas mediante un objeto APIRouter. Esta herramienta permite organizar la API en módulos o grupos

de rutas para tener una estructura más clara y mantenible. Además, se puede utilizar el objeto APIRouter para definir middleware específico para cada grupo de rutas.

Además de los datos que se van a recibir y devolver en cada ruta, también se puede especificar en FASTAPI el tipo de respuesta HTTP que se debe enviar. Esto se hace a través de los parámetros de la función decorada, como por ejemplo response_class, que indica el tipo de respuesta a enviar.

Entre los tipos de respuesta que se pueden enviar, se encuentran:

- JSONResponse: para enviar una respuesta en formato JSON.
- FileResponse: para enviar un archivo como respuesta.
- PlainTextResponse: para enviar una respuesta en formato de texto plano.
- HTMLResponse: para enviar una respuesta en formato HTML.

Además, se puede especificar el código de respuesta que se debe enviar, como por ejemplo 200 para indicar una respuesta exitosa, o 404 para indicar que no se encontró el recurso solicitado.

```python
class HeartRate(BaseModel):
    hr: int
    time : int

    class Config:
        schema_extra = {
            "example": {
                "hr": 120,
                "time": 1655454149
            }
        }

class ItemHR(BaseModel):
    idHardware: str
    data: list[HeartRate]
```

Listado 6.3: Modelo de datos.

```python
@router.get("/hrFileDownload/{file_name}",tags=["Heart Rate"], status_code=200)
async def download_hr_file(file_name: str = Path(title="The ID of the item to get"),
    current_user: User = Depends(get_current_active_user)):
    return FileResponse(path=os.path.join(os.getcwd(),"FilesUsers",current_user.username,
        "Hr",file_name), filename=file_name, media_type='text/csv')
```

Listado 6.4: Función para descargar archivos.

6.4.2. Documentación

La documentación de una API es una parte esencial del proceso de desarrollo, ya que permite a los desarrolladores entender cómo usar y integrar la API con otros sistemas.

En este caso dada las utilidades que proporciona FastAPI, se ha generado la documentación con Sawgger [41].

Swagger es un framework de especificación de API que se utiliza para describir, producir, consumir y visualizar APIs RESTful. Permite especificar el comportamiento y la estructura de una API en un formato estándar y fácil de leer, lo que facilita la colaboración y el intercambio de información entre los equipos de desarrollo.

Swagger ofrece una amplia variedad de herramientas para trabajar con APIs, incluyendo un editor en línea que permite a los desarrolladores crear y probar fácilmente las API, así como una biblioteca de clientes que permite a los desarrolladores generar código en diferentes lenguajes de programación. Además, Swagger proporciona una interfaz de usuario visual y fácil de usar para explorar y probar las API.

Dado que Swagger está integrado con FastAPI, la generación de documentación es automática, pudiendo generar unos resultados profesionales. Como se puede ver en la Figura 6.10, la documentación generada contiene los diferentes métodos que se han implementado en la API, pudiendo interactuar directamente con la API sin la necesidad de desarrollar un cliente.

6.4.3. Seguridad

En lo referente a la seguridad, la API gestiona el acceso a determinados métodos de la misma mediante el uso de tokens. Cuando un usuario inicia sesión con sus datos en la API, esta la devuelve un JSON Web Token (JWT), que deberá enviar en la cabecera de las peticiones que realice a los métodos restringidos de la API, con el fin de verificar la identidad del usuario. Se ha elegido JWT puesto que es un estándar de seguridad abierto que se utiliza para transmitir información confidencial de manera segura entre dos partes.

Un JWT, es un documento en formato json que se compone de tres partes: el encabezado (header), el cuerpo (payload) y la firma (signature), Figura 6.11. La parte del encabezado define el tipo de token y el algoritmo de codificación utilizado. La parte

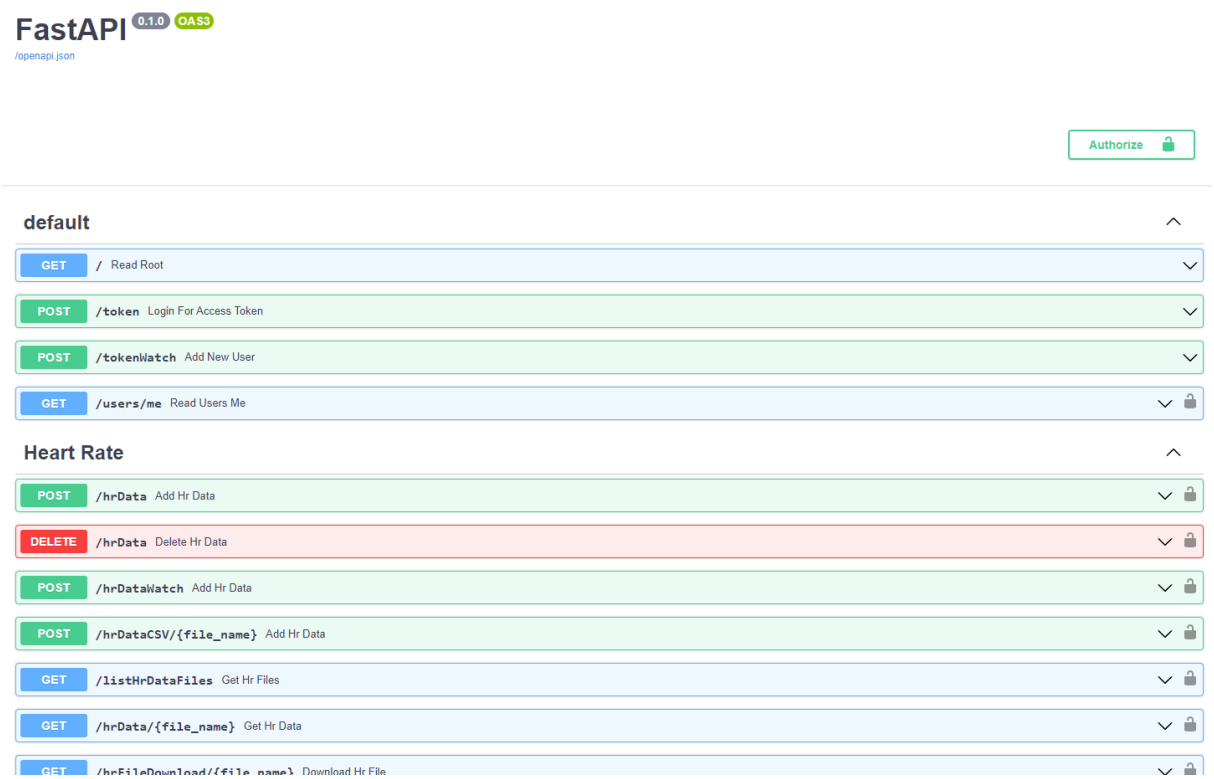

Figura 6.10: Documentación de la API.

del cuerpo contiene la información relevante que se quiere transmitir, en este caso el identificador del usuario y la fecha de expiración del token. La firma es un hash cifrado que se utiliza para verificar la integridad de la información transmitida. Este documento json es codificado antes de enviarlo, generando el JWT.

Dentro de los algoritmos de codificación más utilizados en JWT podemos encontrar HS256 Y RS265 [42], Figura 6.12. Las principales diferencias residen en que HS256 utiliza una clave compartida simétrica, es decir, la misma clave se utiliza para codificar y descodificar la información. Por otro lado, RS256 utiliza una clave asimétrica, es decir, un par de claves pública y privada. Esto implica HS256 es más rápido que RS256 debido a que utiliza una clave simétrica. Sin embargo, la seguridad en RS256 es mayor debido a que utiliza una clave asimétrica.

Aunque HS265 no proporcione el mismo nivel de seguridad que RS265, por el tipo de datos y el volumen que debe gestionar la API, se ha seleccionado HS265.

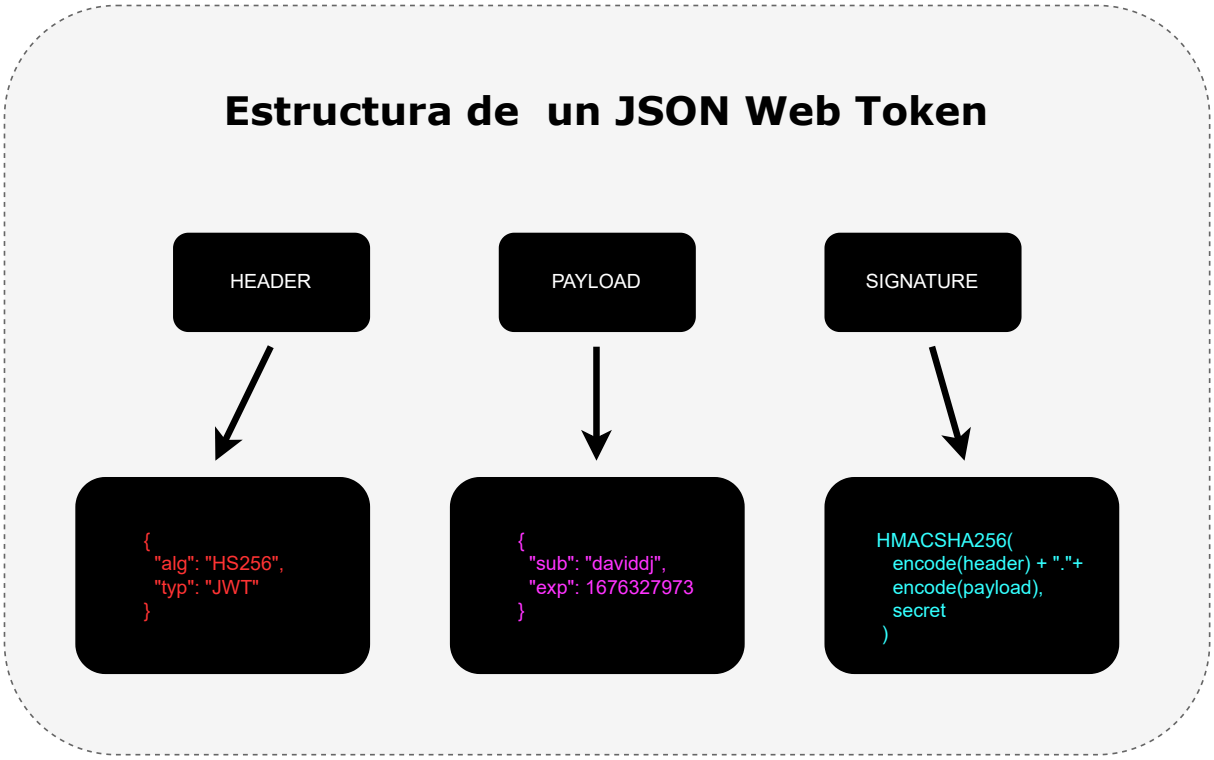

Figura 6.11: Estructura de un JWT.

6.5. Proxy inverso

Como se ha explicado en secciones anteriores, la API que se ha implementado requiere de algún servicio o servidor que gestione las conexiones mediante https. En este sentido y en base a los recursos disponibles se ha optado por implementar un proxy inverso.

Un proxy inverso es un servidor que actúa como intermediario entre los clientes y una aplicación web. La integración de un proxy inverso conlleva una serie de ventajas:

- Mejora de la escalabilidad y rendimiento: Mediante el balanceo de carga entre distintos servidores, se mejora la escalabilidad y el rendimiento.
- Seguridad: Puede actuar como un filtro de seguridad para proteger la aplicación web contra ataques externos, como ataques DDoS (Distributed Denial of Service).
- Caching y aceleración de contenido: El proxy inverso puede almacenar en caché los recursos de la aplicación web para mejorar la velocidad de acceso a ellos por parte de los clientes. Dentro de estos recursos encontramos todo el contenido estático.

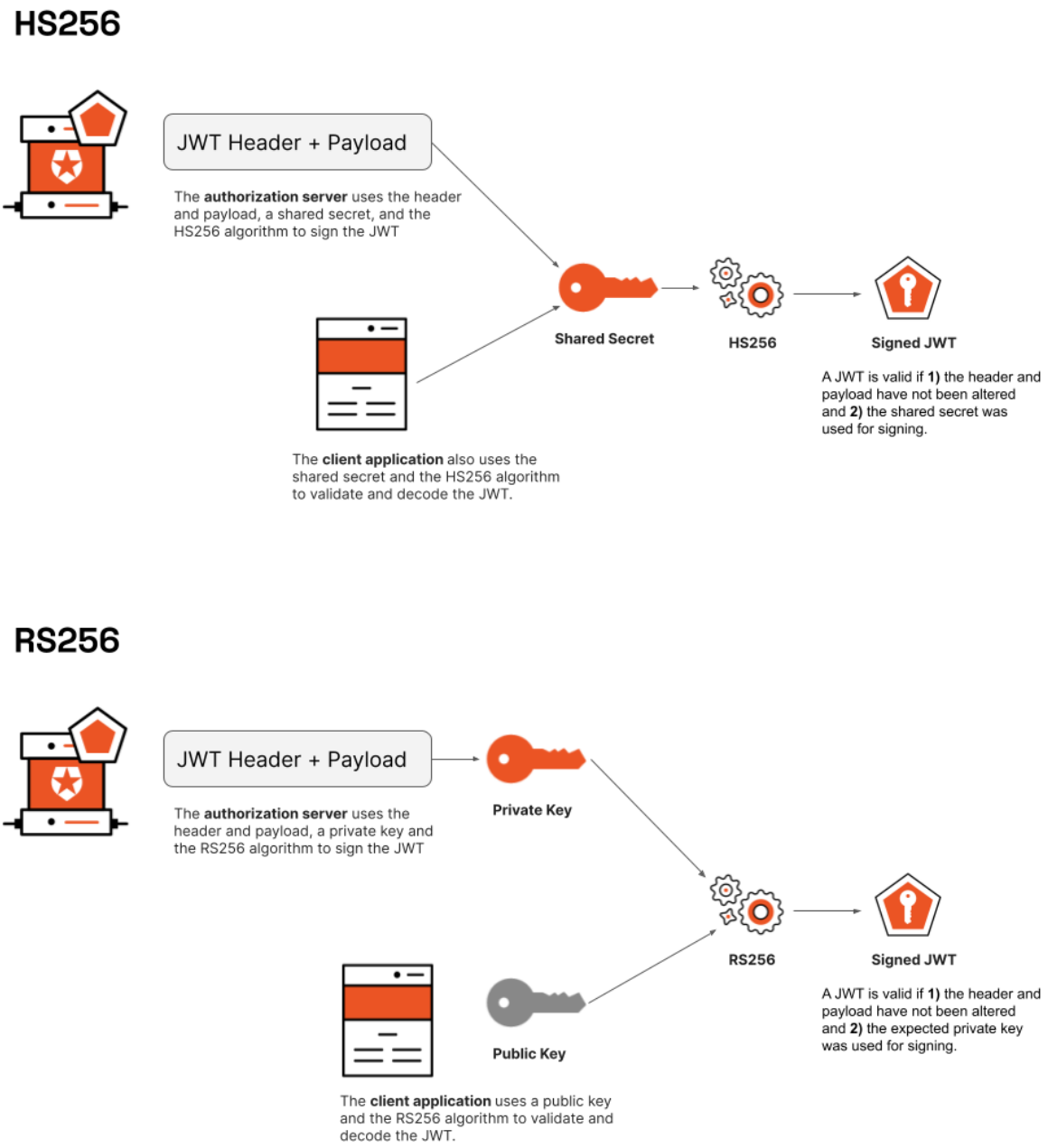

Figura 6.12: HS256 Y RS256 [5].

Como podemos ver, la integración de un proxy inverso en una aplicación web brinda mejoras en el rendimiento, seguridad, velocidad.

En este caso, los principales beneficios de la integración del proxy inverso van a ser en temas de seguridad, ya que al no contar con contenido estático, y al disponer de un único servidor, no se va a poder aprovechar las capacidades de balanceo de carga y aceleración de contenido que nos ofrece.

6.5.1. Servidor web seleccionado

Dentro de los servidores web, se pueden se encuentran dos opciones que predominan en el mercado, Apache y Ngix, Figura 6.13.

Figura 6.13: Apache y Nginx.

Apache es uno de los servidores web más utilizados y es conocido por su estabilidad, seguridad y escalabilidad. Es compatible con una amplia gama de tecnologías web. Además, Apache cuenta con una amplia comunidad de desarrolladores y una amplia documentación, lo que lo hace una opción popular para la implementación de sitios web. Por otro lado, Nginx es un servidor web de alto rendimiento cuya popularidad ha ido creciendo en los últimos años. Se utiliza comúnmente como un servidor proxy inverso, balanceador de carga y servidor de contenido estático. A diferencia de Apache, Nginx es más eficiente en el manejo de conexiones simultáneas. Es por esta razón que para este proyecto se ha seleccionado Nginx [43].

6.5.2. Seguridad

Como ya se había anticipado anteriormente, una de las principales razones por las que se ha integrado el proxy inverso es la seguridad.

```
add_header X-Frame-Options              SAMEORIGIN;
add_header X-Content-Type-Options       nosniff;
add_header X-XSS-Protection             "1; mode=block";
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

limit_req_zone $binary_remote_addr zone=mylimit:20m rate=10r/s;
limit_conn_zone $binary_remote_addr zone=limit_conn:10m;

client_body_buffer_size 10K;
client_header_buffer_size 1k;
client_max_body_size 8M;
```

```
12 large_client_header_buffers 2 1k;
13
14 client_body_timeout 10s;
15 client_header_timeout 10s;
16 send_timeout 15s;
17
18 log_format json_combined escape=json
19   '{'
20     '"time_local":"$time_local",'
21     '"remote_addr":"$remote_addr",'
22     '"remote_user":"$remote_user",'
23     '"request":"$request",'
24     '"status":␣"$status",'
25     '"body_bytes_sent":"$body_bytes_sent",'
26     '"request_time":"$request_time",'
27     '"http_referrer":"$http_referer",'
28     '"http_user_agent":"$http_user_agent"'
29   '}';
30
31
32 upstream API {
33 ____server api_rest:8000;
34 }
35
36 server {
37   listen [::]:443 ssl http2 ipv6only=on;
38   listen 443 ssl http2;
39
40   server_name  API-REST.com www.API-REST.com;
41   server_tokens off;
42
43   access_log /log/proxy.log json_combined;
44
45   if ($request_method !~ ^(GET|HEAD|POST|DELETE)$ ) {
46     return 500;
47   }
48
49   error_page 497 https://$host:443$request_uri;
50
51   ssl_certificate /certificado/certificado/certificateCRT.pem;
52   ssl_certificate_key /certificado/asia.ujaen.es_privatekey.pem;
53   ssl_stapling on;
54   ssl_stapling_verify on;
55   ssl_verify_depth 3;
56   ssl_dhparam /dhparams/dhparams.pem;
57   ssl_session_timeout 1d;
58   ssl_session_cache shared:ssl_session_cache:10m;
59   ssl_session_tickets off;
60   ssl_protocols TLSv1.3 TLSv1.2;
61   ssl_prefer_server_ciphers on;
62   ssl_ciphers
63   "TLS13-CHACHA20-POLY1305-SHA256:TLS13-AES-256-GCM-SHA384:TLS13-AES-128-GCM-SHA256:EECDH
       +CHACHA20:EECDH+AESGCM;";
64
65
66   location / {
67       proxy_pass http://API;
68       proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
69       proxy_set_header Host $host;
```

```
70      }
71
72    location /static/ {
73      autoindex on;
74      alias /static/;
75    }
76
77  }
```

Listado 6.5: Configuración del proxy.

Dentro de las medidas de seguridad adoptadas podemos encontrar las siguientes.

HTTPS

Se utiliza HTTPS entre el cliente y el servidor web para cifrar las comunicaciones, mientras que la conexión entre servidor web y la API se realiza mediante HTTP, ya que se comunican en local. Esto permite que se mantenga un balance adecuado entre seguridad y rendimiento.

Para la conexión HTTPS se utiliza TLS 1.2 o 1.3, excluyendo las versiones anteriores, que son consideradas inseguras en la actualidad. Los algoritmos soportados son AES 256 GCM SHA384, CHACHA20 POLY1305 SHA256, AES 128 GCM SHA256, ECDHE RSA AES256 GCMSHA384 y ECDHE RSA AES 128 GCM SHA256. Estos algoritmos se corresponden con los recomendados por diversos estándares de seguridad, ya que no se les conocen vulnerabilidades y ofrecen un grado de seguridad muy alto.

Protección ante ataques

Los principales esfuerzos en protección contra ataques vienen dados por las mitigaciones contra ataques DDoS. Para ello se han limitado el número de peticiones por minuto desde una misma dirección IP y las peticiones en ráfaga o burst. Adicionalmente, se han limitado las peticiones por zonas geográficas, de tal manera que si se detecta una alta actividad en una zona determinada, se podrá suspender el servicio en esa zona sin afectar al resto. También se ha limitado el tamaño máximo de los mensajes provenientes de los clientes, de tal manera que se limita la posibilidad de que se produzcan desbordamientos por la subida de documentos anormalmente grandes. Finalmente, se han establecido una serie de time outs, de tal manera que las conexiones no permanezcan abiertas demasiado tiempo.

Estándares

En lo referente a estándares de seguridad, el servidor web cumple con las recomendaciones de los principales estándares, como son PCI DSS (Payment Card Industry Data Security Standard), HIPAA (Health Insurance Portability and Accountability Act) y NIST (National Institute of Standards and Technology).

6.6. Contenedores

Para facilitar el despliegue de la API y su integración con otros servicios, se ha optado por utilizar contenedores de Docker. Estos contenedores permiten la virtualización a nivel de sistema operativo, lo que a su vez permite la ejecución de aplicaciones de forma aislada y portátil.

Con Docker, se pueden crear y gestionar los contenedores de la API y los servicios necesarios para su correcto funcionamiento. Para asegurar la interacción efectiva entre los contenedores, se ha utilizado Docker Compose, una herramienta que permite la definición y ejecución de aplicaciones Docker con múltiples contenedores.

En el archivo docker-compose.yml se describen los diferentes servicios que componen la API, tales como la base de datos y los servidores, así como las opciones de configuración de cada uno. Las variables de entorno, volúmenes y puertos expuestos son ejemplos de dichas opciones.

```
version: '3.3'

services:

  api_rest:
    build:
      context: ./API
    expose:
      - "8000"
    volumes:
      - ./API:/API
      - /media/almacen/TFM-David/Files:/API/FilesUsers
    restart: always

  nginx_reverse_proxy:
    build:
      context: ./Nginx
    volumes:
      - ./static:/static
      - /media/almacen/TFM-David/logs:/log
      - /media/almacen/certificado_uja/:/certificado
```

```
22      - /media/almacen/TFM-David/dhparams:/dhparams
23    ports:
24      - "8021:443"
25    depends_on:
26      - api_rest
27    restart: always
```

Listado 6.6: docker-compose.yml.

Una vez definido el archivo docker-compose.yml, se puede utilizar el comando docker-compose up para crear y arrancar todos los contenedores necesarios para la API. Docker Compose se encargará de la creación de los contenedores, la configuración de la red y la comunicación entre los servicios correspondientes.

6.7. Protormas lingüísticas

La lógica difusa se ha convertido en una herramienta muy utilizada en la ingeniería, la ciencia y otras áreas para la toma de decisiones, la control de procesos y otras aplicaciones.

Para implementar la lógica difusa, se puede utilizar una biblioteca de Python llamada scikit-fuzzy. Esta biblioteca proporciona una serie de funciones y herramientas para crear y manipular variables difusas, términos difusos y reglas difusas.

- Universo del discurso: El universo del discurso es el rango de valores posibles que una variable difusa puede tomar. En el ejemplo, la variable difusa *ritmo_cardiaco* tiene un universo del discurso que va desde 30 hasta 200, con incrementos de 1.

- Variables difusas: Las variables difusas son aquellas que tienen un universo del discurso que puede ser descrito por medio de términos lingüísticos o imprecisos. En el ejemplo, *ritmo_cardiaco* es una variable difusa que representa la frecuencia cardíaca. Para acceder a la variable difusa *ritmo_cardiaco*, se utiliza la siguiente línea de código:
  $ritmo\_cardiaco = ctrl.Antecedent(np.arange(30,\ 201,\ 1),\ 'ritmo\_cardiaco')$

- Términos difusos: Los términos difusos son etiquetas lingüísticas que se utilizan para describir la pertenencia de un valor a una variable difusa. En el ejemplo, se definen cuatro términos difusos para la variable *ritmo_cardiaco*: *bajo*, *moderado*, *alto* y *muy_alto*. Para crear el término difuso *bajo*, se utiliza la siguiente línea de

código:
$ritmo\_cardiaco['bajo'] = fuzz.trapmf(ritmo\_cardiaco.universe, [30, 30, 60, 70])$

- Funciones de pertenencia: Las funciones de pertenencia se utilizan para asignar un grado de pertenencia a un valor dado dentro del universo del discurso de una variable difusa. En el ejemplo, se utilizan funciones de pertenencia trapezoidales para definir cada término difuso. Por ejemplo, para crear la función de pertenencia trapezoidal del término difuso *moderado*, se utiliza el siguiente código:
$ritmo\_cardiaco['moderado'] = fuzz.trapmf(ritmo\_cardiaco.universe, [60, 70, 90, 110])$

```
1
2   ritmo_cardiaco = ctrl.Antecedent(np.arange(30, 201, 1), 'ritmo_cardiaco')
3   ritmo_cardiaco['bajo'] = hrl = fuzz.trapmf(ritmo_cardiaco.universe, [30, 30, 60, 70])
4   ritmo_cardiaco['moderado'] = hrm = fuzz.trapmf(ritmo_cardiaco.universe, [60, 70, 90,
        110])
5   ritmo_cardiaco['alto'] = hrh = fuzz.trapmf(ritmo_cardiaco.universe, [90, 110, 130,
        150])
6   ritmo_cardiaco['muy_alto'] = hrvh = fuzz.trapmf(ritmo_cardiaco.universe, [130, 150,
        200, 200])
```

Listado 6.7: Definición de la variable difusa ritmo cardíaco y de sus diferentes términos.

Capítulo 7

Caso de Estudio

En esta sección de caso de estudio, se abordará la recopilación de datos de movimiento y frecuencia cardíaca en diferentes escenarios, que van desde baja intensidad hasta alta intensidad. Los resultados obtenidos serán utilizados para calcular las protoformas lingüísticas previamente definidas.

7.1. Descripción

Este caso de estudio consiste en llevar la pulsera de actividad para recopilar datos sobre el movimiento y la frecuencia cardíaca en tres escenarios diferentes: baja intensidad, intensidad moderada y alta intensidad.

- En el primer escenario, de baja intensidad, se recopilarán datos mientras el usuario permanece sentado, mientras que realiza ejercicios de meditación.
- En el segundo escenario, de intensidad moderada, se realizarán ejercicios de musculación básicos, como levantamiento de pesas y sentadillas durante un período de tiempo determinado.
- En el tercer escenario, de alta intensidad, se realizarán ejercicios de alta intensidad, como burpees y otros ejercicios cardiovasculares, durante un período de tiempo determinado.

Una vez que se han recogido los datos de los tres escenarios, se procederá a analizar y comparar los datos recopilados para obtener una visión más completa del

comportamiento de la frecuencia cardíaca y la cantidad de movimiento del usuario en diferentes situaciones. Los datos obtenidos se utilizarán para calcular las protoformas lingüísticas previamente definidas.

7.2. Escenario de baja intensidad

Para el caso de estudio de la frecuencia cardíaca y la aceleración en condiciones de reposo o baja intensidad, se ha estado meditando y cambiando de postura de vez en cuando durante un total de 5 minutos. En este caso, la meditación se ha utilizado como una actividad que promueve la relajación y el control del estrés, y se ha realizado en condiciones de reposo o baja intensidad.

Durante la meditación, se ha estado observando la frecuencia cardíaca y la aceleración con el fin de evaluar los cambios en la actividad cardíaca y la actividad física en respuesta a la meditación. Cambiar de postura de vez en cuando se ha considerado una forma de evitar la incomodidad y la fatiga muscular que podrían surgir al permanecer en la misma posición.

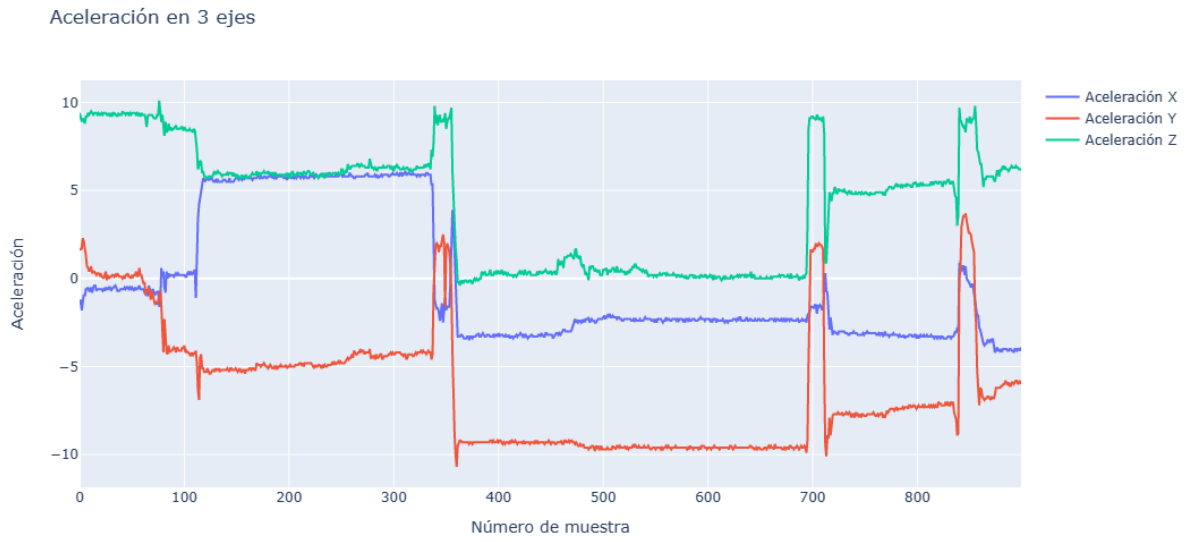

Figura 7.1: Aceleración durante el escenario de baja intensidad.

Como se puede apreciar en las Figuras 7.1 y 7.2, se perciben cambios en la aceleración en los cambios de postura, mientras que la frecuencia cardíaca se sitúa en unos niveles normales.

En lo referente a las protoformas lingüísticas, se ha obtenido para la frecuencia

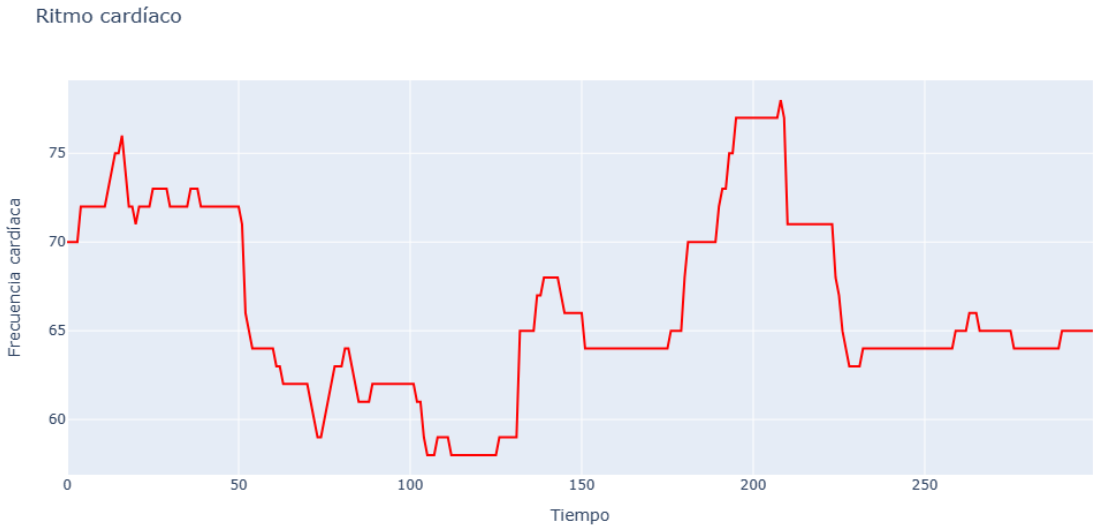

Figura 7.2: Frecuencia cardíaca durante el escenario de baja intensidad.

cardíaca: **La mayor parte del tiempo la frecuencia cardíaca es normal 0.99. En los últimos momentos la frecuencia cardíaca es normal 0.5.** Mientras que para la aceleración se ha obtenido: **La mayor parte del tiempo la aceleración es baja 0.99.**

7.3. Escenario de intensidad moderada

Para el caso de estudio de la frecuencia cardíaca y la aceleración en condiciones de intensidad moderada, se ha estado realizando ejercicios de musculación y sentadillas, alternándose durante 5 minutos.

Como se puede apreciar en las Figuras 7.3 y 7.4, se perciben patrones en la aceleración, que se corresponden con los ejercicios de musculación. En lo referente al ritmo cardíaco podemos ver la evolución conforme se desarrollan los diferentes ejercicios.

Respecto a las protoformas, obtenemos **La mitad del tiempo la frecuencia cardíaca es alta 0.73. En los últimos momentos la frecuencia cardíaca es muy alta 1.0** y **La mayor parte del tiempo la aceleración es baja 0.99**. Como podemos apreciar aunque la aceleración es mayor que en el anterior caso se sigue clasificando como baja, por lo que seria necesario realizar modificaciones en las funciones de pertenencia. Sin embargo, en la frecuencia cardíaca encontramos información de utilidad como que al final del ejercicio, la frecuencia cardíaca es muy alta

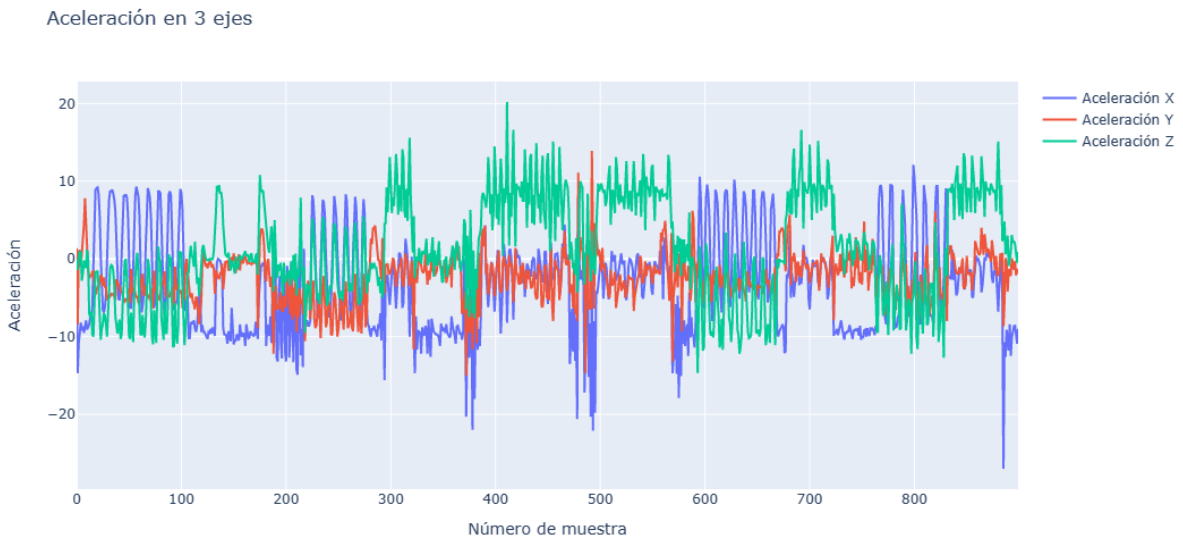

Figura 7.3: Aceleración durante el escenario de intensidad moderada.

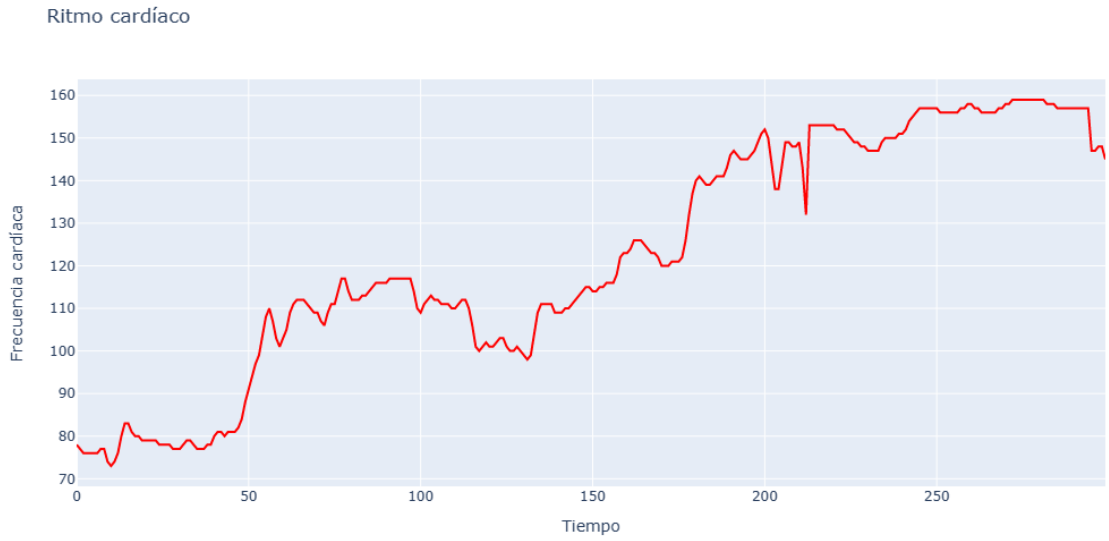

Figura 7.4: Frecuencia cardíaca durante el escenario de intensidad moderada.

7.4. Escenario de alta intensidad

Para el caso de estudio de la frecuencia cardíaca y la aceleración en condiciones de intensidad alta, se ha optado por realizar burpees. Un burpee es un ejercicio físico que involucra varios grupos musculares y es ampliamente utilizado en entrenamientos de alta intensidad y programas de acondicionamiento físico. El burpee se realiza desde una posición de pie, luego se baja al suelo en posición de flexión de brazos (push-up), se regresa a la posición de pie y se salta en el aire con las manos extendidas hacia arriba. Esto lo convierte en un ejercicio idóneo para un escenario de alta intensidad.

Figura 7.5: Aceleración durante el escenario de alta intensidad.

Tal como apreciar en las Figuras 7.5 y 7.6, se aprecian tanto aceleraciones altas como una frecuencia cardiaca elevada. Respecto a las protoformas, obtenemos las siguientes: **La mayor parte del tiempo la aceleración es alta 0.71** y **La mayor parte del tiempo la frecuencia cardíaca es muy alta 0.99. En los últimos momentos la frecuencia cardíaca es muy alta 1.0**

7.5. Conclusiones

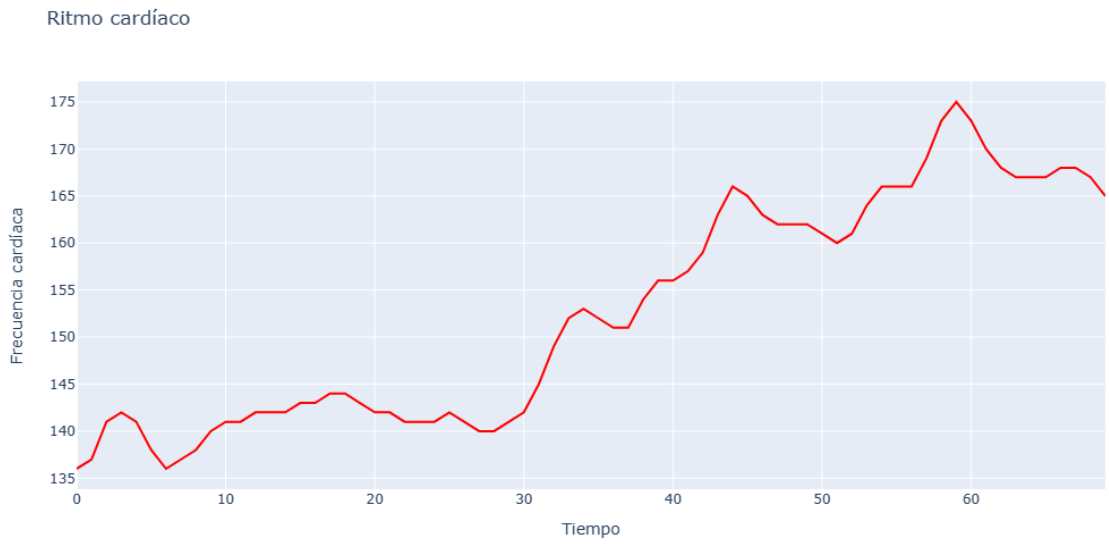

Figura 7.6: Frecuencia cardíaca durante el escenario de alta intensidad.

Capítulo 8

Conclusiones

En el marco de este trabajo de fin de máster, se ha diseñado una arquitectura que se basa en la utilización de dispositivos vestibles y teléfonos inteligentes, que se ha centrado en mejorar aspectos a las soluciones existentes, como la disponibilidad de los datos o la generación de resúmenes lingüísticos. Para ello, se emplean sensores como acelerómetros, giroscopios y sensores de frecuencia cardíaca, que recopilan información sobre los movimientos, actividad física y condición cardiovascular del usuario, y la transmiten de forma inalámbrica a un teléfono inteligente.

Asimismo, se ha revisado el proyecto para asegurar su alineación con las preocupaciones actuales y los Objetivos de Desarrollo Sostenible marcados por la Agenda 2030 de las Naciones Unidas. Se ha desarrollado una API REST que permite la comunicación entre los dispositivos y una plataforma central donde se almacenan y analizan los datos recolectados. Esta plataforma brinda a los usuarios acceso a sus datos y les permite realizar análisis y visualizaciones avanzadas. También se han proporcionado métodos para describir los flujos de datos utilizando el lenguaje natural y la lógica difusa, con el objetivo de mejorar la comprensión de los datos recolectados y facilitar su análisis.

En adición a lo anteriormente mencionado, se ha desarrollado una serie de aplicaciones específicas para dispositivos móviles y relojes inteligentes. Estas aplicaciones permiten a los usuarios recopilar y visualizar los datos de manera sencilla e intuitiva, mejorando así su experiencia de usuario. Estas aplicaciones están diseñadas para funcionar en consonancia con la plataforma central y la API REST, lo que permite una integración fluida y una gestión eficiente de los datos recolectados.

Finalmente, con el fin de validar y evaluar en condiciones reales la arquitectura propuesta, se ha desarrollado un caso de estudio en el que se ha utilizado la platafor-

ma para recolectar datos en diferentes escenarios durante un período determinado de tiempo.

8.0.1. Trabajo futuro

A pesar de que el proyecto cuenta con una arquitectura completa y funcional para la recolección y análisis de datos de actividad de los usuarios, existen diversas posibilidades de mejora y expansión que pueden ser implementadas en el futuro. A continuación, se describen algunas de las mejoras que se pueden considerar para futuras versiones del proyecto:

- Adición de soporte a nuevos sensores: Una posible mejora en el proyecto podría ser la incorporación de nuevos sensores a la arquitectura existente. En particular, sería interesante añadir soporte para sensores de SpO2, que miden la saturación de oxígeno en la sangre. La incorporación de este tipo de sensores permitiría recopilar información adicional sobre la condición cardiovascular del usuario, lo que mejoraría la precisión y la calidad de los datos recolectados.
- Incorporación de análisis de patrones de sueño: Sería interesante incorporar la capacidad de monitorear el sueño del usuario. Esto podría lograrse mediante la incorporación de sensores de sueño, que miden los patrones de sueño y la calidad del sueño. Esta información podría ser utilizada para brindar recomendaciones personalizadas a los usuarios para mejorar su calidad de sueño.
- Desarrollo de una función de alerta temprana: Una posible mejora sería desarrollar una función de alerta temprana que notifique al usuario si se detectan patrones de actividad física o de sueño que sugieren un riesgo para la salud. Esta función podría basarse en reglas difusas o en técnicas de aprendizaje automático para detectar patrones anormales y alertar al usuario.
- Mejora en la descripción de flujos de datos: Otra posible mejora sería la incorporación de protoformas más complejas para describir los flujos de datos, como puede ser la evolución temporal de dichos flujos. Esto permitiría una mejor comprensión de los datos recolectados y facilitaría su análisis. Además, podría mejorarse la descripción de flujos de datos utilizando técnicas de aprendizaje automático para detectar patrones y anomalías en los datos.
- Incorporación de reglas difusas: Las reglas difusas son una herramienta útil para combinar diferentes variables y determinar la intensidad del ejercicio realizado

por el usuario. Por lo tanto, se podría incorporar reglas difusas a la arquitectura existente para mejorar la precisión en la evaluación de la actividad física realizada por los usuarios.

- Integración de sensores de múltiples dispositivos, tanto wearable, como IoT de máquinas de ejercicio inteligente que permitan enriquecer la descripción lingüística con las reglas difusas desde múltiples fuentes de datos.

Capítulo 9

Anexo

9.1. Manual de instalación

En esta sección se detallarán los requisitos necesarios y los pasos requeridos para llevar a cabo el despliegue del sistema.

9.1.1. API, Proxy y DB

El despliegue del sistema se simplifica al haber sido construido mediante contenedores, en particular el proxy, la base de datos y la API. Al utilizar contenedores, la configuración necesaria y los puertos requeridos para la ejecución de cada servicio se especifican en el archivo docker-compose. Esto hace que el despliegue del sistema sea sencillo y coherente.

Es importante mencionar que la aplicación de la API tiene definidas sus librerías necesarias en un archivo de requisitos. Estas librerías se instalan automáticamente durante el proceso de creación del contenedor, lo que facilita aún más el despliegue de la aplicación.

Al tener las librerías especificadas en el archivo de requisitos, es más fácil garantizar la coherencia y la compatibilidad de las dependencias necesarias para el correcto funcionamiento de la aplicación. De esta manera, se asegura que la aplicación se ejecute correctamente en cualquier entorno de contenedor en el que se despliegue.

El primer paso para desplegar el sistema es ejecutar el comando **docker-compose up -d**, para iniciar los servicios de contenedores en segundo plano. Sin embargo, antes

de hacerlo, es necesario verificar que los certificados para HTTPS estén disponibles y cambiarlos si es necesario. Esto es importante para asegurar la seguridad de la comunicación entre los servicios y los usuarios.

```
asia@ASIA01:/media/almacen/TFM-David/API-REST-Android$ sudo docker compose up -d
[+] Building 38.1s (16/16) FINISHED
 => [api-rest-android-nginx_reverse_proxy internal] load .dockerignore                                    0.0s
 => => transferring context: 2B                                                                           0.0s
 => [api-rest-android-nginx_reverse_proxy internal] load build definition from Dockerfile                 0.0s
 => => transferring dockerfile: 113B                                                                      0.0s
 => [api-rest-android-api_rest internal] load .dockerignore                                               0.0s
 => => transferring context: 2B                                                                           0.0s
 => [api-rest-android-api_rest internal] load build definition from Dockerfile                            0.0s
 => => transferring dockerfile: 357B                                                                      0.0s
 => [api-rest-android-nginx_reverse_proxy internal] load metadata for docker.io/library/nginx:1.23.2-alpine  0.0s
 => [api-rest-android-nginx_reverse_proxy internal] load build context                                    0.1s
 => => transferring context: 1.86kB                                                                       0.0s
 => [api-rest-android-nginx_reverse_proxy 1/2] FROM docker.io/library/nginx:1.23.2-alpine                 0.1s
 => [api-rest-android-api_rest internal] load metadata for docker.io/library/python:3.9-slim-bullseye     0.0s
 => [api-rest-android-api_rest 1/5] FROM docker.io/library/python:3.9-slim-bullseye                       0.1s
 => [api-rest-android-api_rest internal] load build context                                               0.1s
 => => transferring context: 222B                                                                         0.0s
 => [api-rest-android-nginx_reverse_proxy 2/2] COPY ./default.conf /etc/nginx/conf.d/default.conf         0.1s
 => [api-rest-android-api_rest 2/5] WORKDIR /API                                                          0.0s
 => [api-rest-android-api_rest 3/5] RUN apt update  && apt install -y python3-dev libpq-dev gcc  && pip install   22.9s
 => [api-rest-android-api_rest] exporting to image                                                        1.4s
 => => exporting layers                                                                                   1.4s
 => => writing image sha256:d7f2bab5cc26e811fedcd978f683defd753a479b0c45e119692d62cdafbaad33              0.0s
 => => naming to docker.io/library/api-rest-android-nginx_reverse_proxy                                   0.0s
 => => writing image sha256:e551549c5ab95c367fb39d5cd63483b7d3da6c6470909d09cf1977294d5970b3              0.0s
 => => naming to docker.io/library/api-rest-android-api_rest                                              0.0s
 => [api-rest-android-api_rest 4/5] COPY ./requirements.txt ./                                            0.1s
 => [api-rest-android-api_rest 5/5] RUN pip install --no-cache-dir --upgrade -r ./requirements.txt       13.4s
[+] Running 3/3
 ⠿ Network api-rest-android_default                 Created                                                0.0s
 ⠿ Container api-rest-android-api_rest-1           Started                                                0.6s
 ⠿ Container api-rest-android-nginx_reverse_proxy-1  Started                                              0.9s
asia@ASIA01:/media/almacen/TFM-David/API-REST-Android$
```

Figura 9.1: Despliegue de la API y el Proxy.

Otro paso necesario es crear la colección correspondiente a los usuarios una vez que la base de datos esté en funcionamiento. La colección es esencial para el almacenamiento y la gestión de los datos de los usuarios.

Finalmente, es importante verificar que los servicios tienen conexión entre sí. Para ello, se pueden realizar pruebas de conectividad o verificar los registros de los servicios.

9.1.2. Aplicaciones

El proceso de instalación de aplicaciones puede ser más laborioso que el despliegue de contenedores. En el caso de la aplicación desarrollada para Android, se puede instalar directamente desde el archivo apk. Sin embargo, si se modifican los puertos y la ubicación de la API, habría que realizar una modificación previa en el archivo de configuración de la aplicación.

Por otro lado, para instalar la aplicación de Fitbit en un dispositivo wearable, se requeriría seguir una serie de pasos. En primer lugar, se tendría que descargar e instalar

la aplicación oficial de Fitbit, en el teléfono móvil, desde la Play Store. A continuación, se tendría que iniciar sesión o crear una cuenta. El paso más complicado sería desplegar la aplicación en el dispositivo wearable.

Para desplegar la aplicación en el dispositivo, se tendrían que habilitar las opciones de desarrollo desde la configuración del dispositivo y posteriormente, desde la línea de comandos con Fitbit CLI o con Fitbit Studio, desplegar la aplicación en el dispositivo.

Para ello harían falta una serie de comandos, siendo el primero de ellos **npx fitbit** dentro de la carpeta del proyecto. Esto abriría una pestaña en el navegador web para iniciar sesión en fitbit y poder cargar la aplicación posteriormente en los dispositivos. A continuación introduciríamos el comando **build** y el comando **install** para instalar la aplicación.

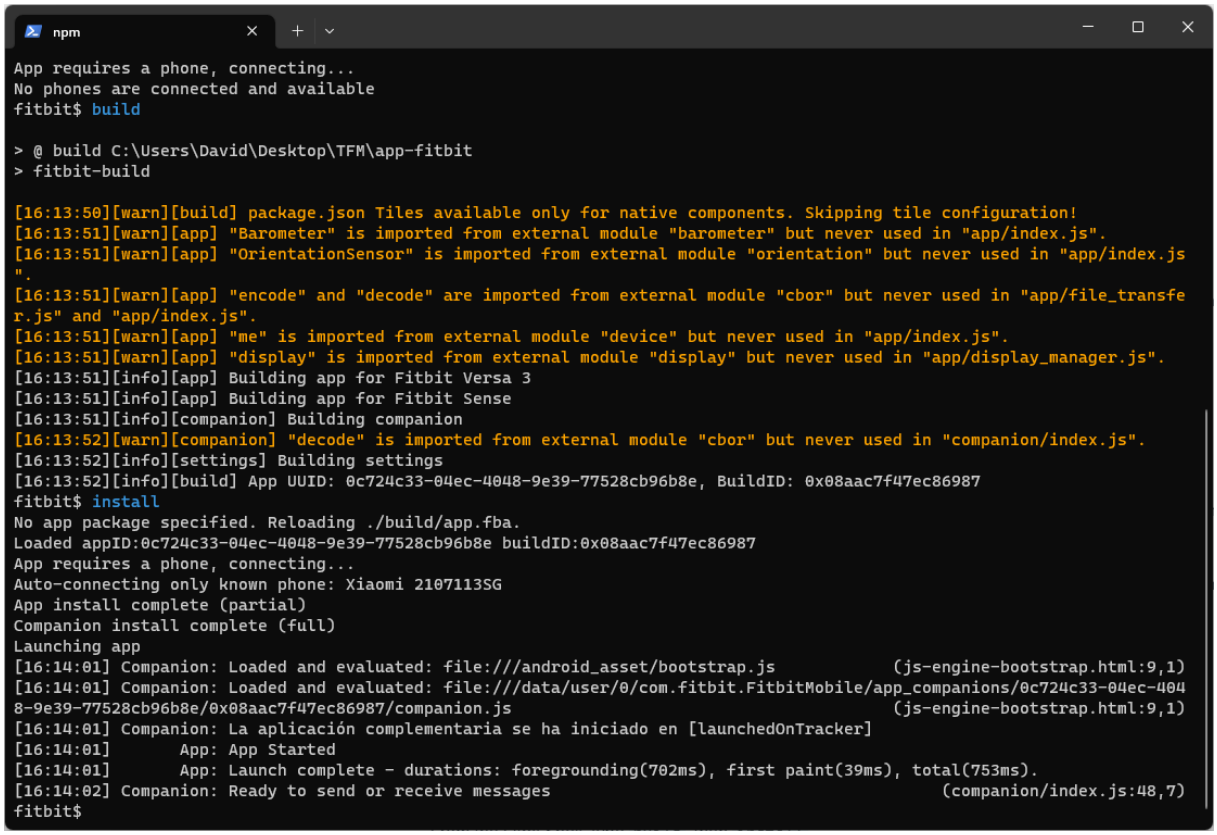

Figura 9.2: Instalación de la app en el dispositivo wearable.

Dada la complejidad que puede suponer estos pasos y dado que con el tiempo los procedimientos de despliegue podrían variar, se recomienda acceder al sitio oficial, donde se detalla el proceso: https://dev.fitbit.com/getting-started/

Bibliografía

[1] Aaron Parecki. Wearables graveyard, Jun 2017. URL https://www.flickr.com/photos/aaronpk/35227277035.

[2] La asamblea general adopta la agenda 2030 para el desarrollo sostenible - desarrollo sostenible. URL https://www.un.org/sustainabledevelopment/es/2015/09/la-asamblea-general-adopta-la-agenda-2030-para-el-desarrollo-sostenible/.

[3] Wikipedia. Waterfall system model. URL https://commons.wikimedia.org/wiki/File:Waterfall_system_model.jpg.

[4] Verlaciudad. Scrum, Jun 2013. URL https://www.flickr.com/photos/51211903@N06/9086950090/in/photostream/.

[5] Rs256 vs hs256 jwt signing algorithms, Feb 2021. URL https://community.auth0.com/t/rs256-vs-hs256-jwt-signing-algorithms/58609.

[6] Jessilyn Dunn, Ryan Runge, and Michael Snyder. Wearables and the medical revolution. *Personalized medicine*, 15(5):429–448, 2018.

[7] Sumit Majumder, Tapas Mondal, and M Jamal Deen. Wearable sensors for remote health monitoring. *Sensors*, 17(1):130, 2017.

[8] Alexandros Pantelopoulos and Nikolaos G Bourbakis. A survey on wearable sensor-based systems for health monitoring and prognosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(1): 1–12, 2009.

[9] F John Dian, Reza Vahidnia, and Alireza Rahmati. Wearables and the internet of things (iot), applications, opportunities, and challenges: A survey. *IEEE access*, 8:69200–69211, 2020.

[10] Yolanda-María de-la Fuente-Robles, Adrián-Jesús Ricoy-Cano, Antonio-Pedro Albín-Rodríguez, José Luis López-Ruiz, and Macarena Espinilla-Estévez. Past,

present and future of research on wearable technologies for healthcare: A bibliometric analysis using scopus. *Sensors*, 22(22):8599, 2022.

[11] Ramyar Saeedi, Keyvan Sasani, Skyler Norgaard, and Assefaw H Gebremedhin. Personalized human activity recognition using wearables: A manifold learning-based knowledge transfer. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 1193–1196. IEEE, 2018.

[12] Antonio-Pedro Albín-Rodríguez, Yolanda-María De-La-Fuente-Robles, José-Luis López-Ruiz, Ángeles Verdejo-Espinosa, and Macarena Espinilla Estévez. Ujami location: A fuzzy indoor location system for the elderly. *International Journal of Environmental Research and Public Health*, 18(16):8326, 2021.

[13] Antonio-Pedro Albín-Rodríguez, Adrián-Jesús Ricoy-Cano, Yolanda-María de-la Fuente-Robles, and Macarena Espinilla-Estévez. Fuzzy protoform for hyperactive behaviour detection based on commercial devices. *International Journal of Environmental Research and Public Health*, 17(18):6752, 2020.

[14] MA López-Medina, Macarena Espinilla, Ian Cleland, C Nugent, and Javier Medina. Fuzzy cloud-fog computing approach application for human activity recognition in smart homes. *Journal of Intelligent & Fuzzy Systems*, 38(1):709–721, 2020.

[15] María Dolores Peláez-Aguilera, Macarena Espinilla, Maria Rosa Fernandez Olmo, Javier Medina, et al. Fuzzy linguistic protoforms to summarize heart rate streams of patients with ischemic heart disease. *Complexity*, 2019, 2019.

[16] Carmen Martinez-Cruz, Antonio J Rueda, Mihail Popescu, and James M Keller. New linguistic description approach for time series and its application to bed restlessness monitoring for eldercare. *IEEE Transactions on Fuzzy Systems*, 30 (4):1048–1059, 2021.

[17] Javier Medina, Luis Martinez, and Macarena Espinilla. Subscribing to fuzzy temporal aggregation of heterogeneous sensor streams in real-time distributed environments. *International Journal of Communication Systems*, 30(5):e3238, 2017.

[18] Department of Economic United Nations and Social Affairs Sustainable Development. Transforming our world: the 2030 agenda for sustainable development, 2015. URL https://sdgs.un.org/2030agenda.

[19] Winston W Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338, 1987.

[20] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001, 2001.

[21] Patricio Letelier and Mª Carmen Penadés. Métodologías ágiles para el desarrollo de software: extreme programming (xp). *Universidad Politécnica de Valencia*, 17, 2012.

[22] Craig Larman and Victor R Basili. Iterative and incremental developments. a brief history. *Computer*, 36(6):47–56, 2003.

[23] James M Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, 149(2):430–437, 2003.

[24] Lotfi A Zadeh. Fuzzy logic. In *Granular, Fuzzy, and Soft Computing*, pages 19–49. Springer, 2023.

[25] L.A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning-III. *Information Sciences*, 9(1):43–80, 1975. doi: 10.1016/0020-0255(75)90017-1.

[26] Maysam F Abbod, Diedrich G von Keyserlingk, Derek A Linkens, and Mahdi Mahfouf. Survey of utilisation of fuzzy technology in medicine and healthcare. *Fuzzy Sets and Systems*, 120(2):331–349, 2001.

[27] Gang Feng. A survey on analysis and design of model-based fuzzy control systems. *IEEE Transactions on Fuzzy systems*, 14(5):676–697, 2006.

[28] Leonel Suganthi, S Iniyan, and Anand A Samuel. Applications of fuzzy logic in renewable energy systems–a review. *Renewable and sustainable energy reviews*, 48:585–607, 2015.

[29] Ronald R Yager and Lotfi A Zadeh. *An introduction to fuzzy logic applications in intelligent systems*, volume 165. Springer Science & Business Media, 2012.

[30] Lotfi A. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Computers & Mathematics with Applications*, 9(1):149–184, 1983. doi: 10.1016/0898-1221(83)90013-5.

[31] Javier Medina, Macarena Espinilla, Daniel Zafra, Luis Martínez, and Christopher Nugent. Fuzzy fog computing: A linguistic approach for knowledge inference in wearable devices. In *Ubiquitous Computing and Ambient Intelligence: 11th International Conference, UCAmI 2017, Philadelphia, PA, USA, November 7–10, 2017, Proceedings*, pages 473–485. Springer, 2017.

[32] D. Díaz, J. Medina, A. Montoro, José L. López, and M. Espinilla. Linguistic summaries for dwellings energy poverty monitoring. In José Bravo, Sergio Ochoa, and Jesús Favela, editors, *Proceedings of the International Conference on Ubiquitous Computing & Ambient Intelligence (UCAmI 2022)*, pages 693–704, Cham, 2023. Springer International Publishing. ISBN 978-3-031-21333-5.

[33] Suzanne Robertson and James Robertson. *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.

[34] Welcome to flask. URL https://flask.palletsprojects.com/en/2.2.x/.

[35] Tom Christie. Django rest framework. URL https://www.django-rest-framework.org/.

[36] The falcon web framework. URL https://falcon.readthedocs.io/en/stable/.

[37] Pylons Project. Pyramid the start small, finish big, stay finished framework. URL https://trypyramid.com/.

[38] Fastapi framework, high performance, easy to learn, fast to code, ready for production. URL https://fastapi.tiangolo.com/.

[39] Inc. Square. Okhttp. URL https://square.github.io/okhttp/.

[40] auth0.com. Json web tokens introduction. URL https://jwt.io/introduction.

[41] Api development for everyone. URL https://swagger.io/.

[42] Will Johnson. Rs256 vs hs256 what's the difference?, May 2022. URL https://auth0.com/blog/rs256-vs-hs256-whats-the-difference/#HS256-Signing-Algorithm.

[43] Advanced load balancer, web server, & reverse proxy, Mar 2023. URL https://www.nginx.com/.