UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

PoCAR: puntos de atención sanitaria en entornos rurales y remotos

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: Melero Garrigos, Laura

Tutor/a: Pérez Pascual, Mª Asunción

Cotutor/a: Alberola Oltra, Juan Miguel

CURSO ACADÉMICO: 2021/2022

Contenido

Figuras

Tablas

Acrónimos

- **AMQP:** Advanced Message Queuing Protocol.
- **API:** Application Programming Interface.
- **AWS:** Amazon Web Service.
- **BLE:** Bluetooth Low Energy.
- **CoAP:** Constrained Application Protocol.
- **DDS:** Data Distribution Service.
- **DGT:** Dirección General de Tráfico.
- **EHR:** Electronic Health Record.
- **HTTP:** HyperText Transfer Protocol.
- **HTTPS:** HyperText Transfer Protocol Secure.
- **IA:** Inteligencia Artificial.
- **IaaS:** Infraestructure as a Service.
- **IoT:** Internet of Things.
- **IP:** Internet Protocol.
- **LoRa:** Long Range.
- **MAC:** Media Access Control.
- **MVC:** Model View Controller.
- **MQTT:** Message Queuing Telemetry Transport.
- **OPC:** OLE for Process Control.
- **OSI:** Open Systems Interconnection.
- **OTA:** Over The Air.
- **PHP:** Hypertext Preprocessor.
- **PaaS:** Platform as a Service.
- **PoCAR:** Point-of-care Attention in Rural environments.
- **PWA**: Progressive Web Apps.
- **QoS:** Quality of Service.
- **REST:** Representation State Transfer.
- **RHM:** Remote Health Monitoring.
- **SaaS:** Software as a Service.
- **SSL:** Secure Sockets Layer.
- **TCP:** Transmission Control Protocol.
- **TLS:** Transport Layer Security.
- **UUID:** Universally Unique Identifier.
- **WBAN:** Wireless Body Area Network.
- **WPAN:** Wirless Personal Area Network.
- **ZBS:** Zona básica de salud.

Resumen

Uno de los grandes problemas a los que se enfrenta la sociedad española es el deterioro de los servicios médicos de las zonas rurales. Este problema es más complejo en el interior de España, en donde la población se encuentra en pequeños núcleos rurales, separados entre sí por grandes distancias y alejados de los centros de salud. Este problema afecta a unos 7,5 millones de habitantes.

Con la finalidad de proporcionar un sistema de monitorización remota de pacientes, surge el proyecto de PoCAR (Puntos de atención sanitaria en entornos rurales y remotos), financiado por la fundación FISABIO, y en el que se engloba este trabajo de final de grado.

Abstract

One of the great problems facing Spanish society is the deterioration of medical services in rural areas. This problem is especially serious in the interior of Spain, where the population lives in small rural towns, separated from each other by great distances, and far from health centers. This problem affects some 7.5 million inhabitants.

In order to provide a remote patient monitoring system, the PoCAR project (Distance Point-of-care Attention in Rural environments) was created, financed by the FISABIO foundation, and in which this final degree project is included.

1 Introducción

1.1 Contexto y problemática

En los últimos años, España ha notado un aumento del deterioro de los servicios médicos en las zonas rurales, provocado no solo por los recortes y por la pandemia del Covid-19, sino por el aislamiento geográfico de estos municipios (lo cual supone una mayor dificultad y coste en la provisión de determinados servicios, no solo el médico).

Según el estudio realizado por el Banco de España [1], las zonas rurales españolas tienen un peor acceso a servicios que las de otros países de Europa; teniendo en cuenta las largas distancias que debe recorrer un ciudadano para acceder a servicios básicos como hospitales, ambulatorios, escuelas o supermercados.

Mientras que un usuario que vive en una zona urbana recorre una media de 3,5 km, uno que vive en una zona rural, debe recorrer aproximadamente 12,4 km. Esta distancia, aumenta hasta unos 30 km en algunas provincias con menor densidad de población, tal y como muestra la Figura 1 y la Figura 2.

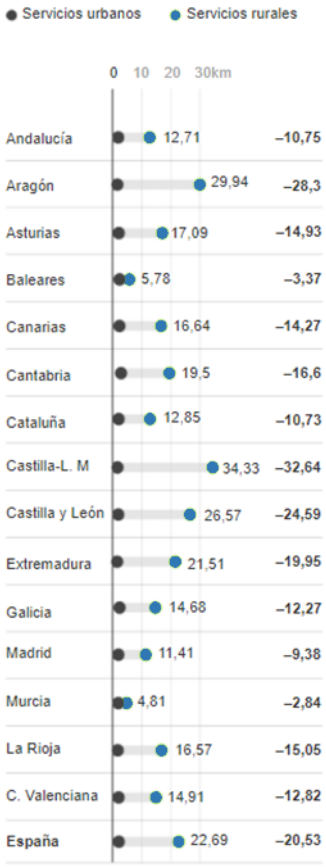


Figura 1. Accesibilidad a servicios locales y regionales (distancia media por persona al servicio más cercano)
Fuente. El País [2]

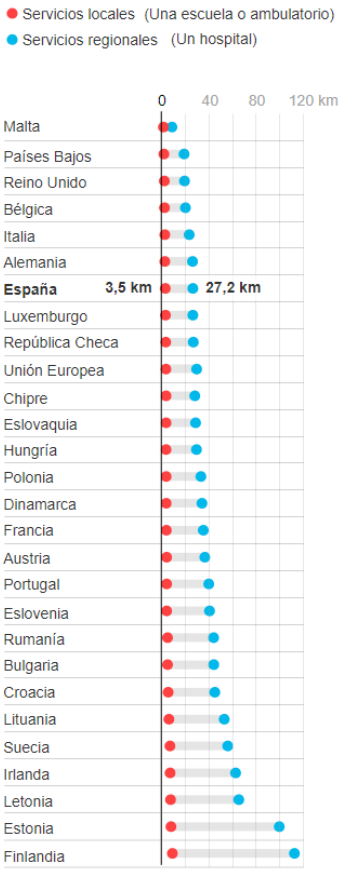


Figura 2. Diferencias en el acceso a servicios locales (distancia media por persona al servicio más cercano)
Fuente. El País [2]

Tal y como indica el estudio del Ministerio de Agricultura, Pesca y Alimentación [3] la población española en áreas rurales se cifra en 7,5 millones de personas teniendo en cuenta que entre el 30 % y el 50 % de la población de Extremadura, Castilla-La Mancha, Castilla y León y Aragón está censada en dichas áreas.

Las zonas más despobladas se caracterizan por la elevada edad de sus ciudadanas. En [4] se detalla que la comunidad de Castilla y León cuenta con una media de edad de 55 años, una cifra impulsada al alza por la cantidad de personas de la tercera edad que reside en esos territorios. Esto significa que, en los municipios con un censo menor a 2.000 personas, 1 de cada 6 personas, es mayor de 80 años.

Todos estos datos indican que hay una amplia población de edad avanzada (octogenarios) que vive en zonas rurales que presentan un difícil acceso y que requieren de largos desplazamientos para, por ejemplo, visitar a sus médicos.

Cierto es que en España no hay límite de edad para poder conducir, pero eso no excluye la dificultad de desplazamiento para esos ciudadanos. Según la DGT (ver artículo [5], en España solo el 15,5 % del total de las personas con carné de conducir tiene más de 65 años (esto equivale a 4,1 millones de personas). En 2019 fallecieron en siniestro viales 419 personas mayores, el 28% de las víctimas totales.

Se denota, por tanto, que la solución al problema de acceso a la sanidad no puede residir a corto plazo en que más personas mayores tengan permisos de conducir (ya que supone un riesgo para ellos y para el resto de la población) pero tampoco en poner más hospitales puesto que los costes son insostenibles por la infraestructura sanitaria actual. En cambio, se puede pensar en soluciones que no requieran el desplazamiento de los pacientes gracias a la conexión a internet.
En 2020 España se situaba en el quinto lugar en conectividad digital (alcance y calidad), teniendo en las zonas rurales una cobertura del 71,3% de banda ancha de 30 MB, un 18,6% de banda ancha de 100 MB, un 97% de red móvil 3G y un 92,8% de red móvil 4G. Todos estos datos, pueden verse reflejados en la Figura 3.

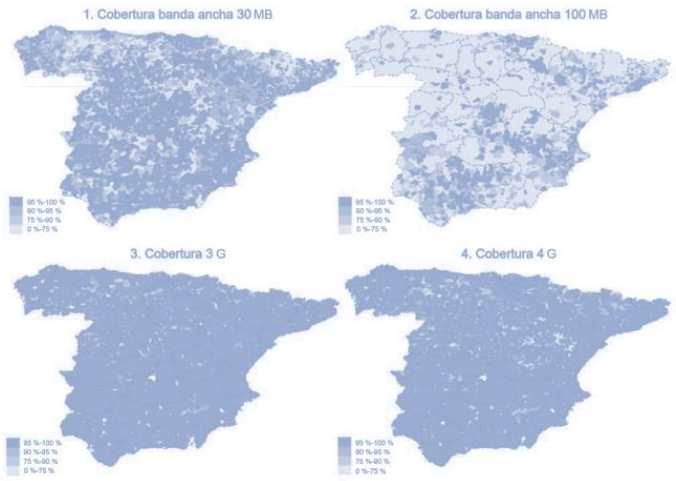


Figura 3. Cobertura de red en España (Junio 2019)
Fuente. Banco de España – Eurosistema [1]

Estos valores aseguran que en la gran mayoría de zonas rurales se podría establecer una conexión de internet sobre la cual enviar y recibir pequeñas tramas de datos. En el caso de ese pequeño porcentaje de zonas rurales que no tienen acceso a conexión, se podría utilizar la tecnología inalámbrica LoRa [6], que asegura la conexión a grandes distancias, con muy poco consumo de red y en lugares con poca cobertura.

1.2 Estado del arte de los sistemas RHM basados en BLE

Los beneficios de la telemedicina permiten establecer nuevos estándares para mantener a los pacientes conectados de forma continua con los médicos y especialistas del sector. Tal como se evidencia en [7], este tipo de sistema reportan los siguientes beneficios:

- **Acceso mejorada a la asistencia:** ofrece un mejor acceso a la sanidad por parte de las poblaciones rurales y desatendidas, facilitando así una correcta atención a personas con movilidad limitada, sin carné de conducir o con cualquier otra dificultad. Este sistema, además permite que los centros o puestas de salud rurales ofrezcan una conexión con los médicos especialistas que están en hospitales localizados a grandes distancias.

- **Asistencia de alta calidad:** puede suponer la mejora del diagnóstico y tratamiento del paciente gracias a su continua monitorización respaldada muchas veces por una IA (Inteligencia Artificial) que permite tratar los datos del paciente de forma automática presentando alertas o posibles soluciones tanto al médico como al

paciente.

- **Eficiencia clínica:** permite la reducción de costes en cuanto a asistencia presencial ya que se ahorra en limpieza, desplazamientos, horas de trabajo (citas más rápidas) e infraestructuras. Además, la integración con historiales clínicos electrónicos (EHR) permiten comparar los datos de las pruebas de forma rápida y clara.

- **Entorno seguro para pacientes y sanitarios:** tal y como hemos podido ver durante la pandemia del Covid-19, las enfermedades infecciosas (como el propio Covid-19 o la gripe entre otros) son un problema para los sanitarios que, de esta forma, quedan aislados de los posibles virus. Este sistema también presenta una gran ventaja para pacientes con riesgos como los inmunodeprimidos, a los cuales se les garantiza una óptima atención sin poner en riesgo su salud.

El artículo [7] explora más en profundidad acerca de la monitorización de pacientes y las consultas a distancia. En él, se expone la importancia de poder transmitir datos biométricos a través de distintos dispositivos (pulsómetros, termómetros…) con la finalidad de que los sanitarios puedan ver en tiempo real (o casi) el estado del paciente y tomar una decisión al respecto. Adicionalmente, se habla de la importancia de las futuras generaciones de IA que, a través de estos sistemas, permitirá no solo el tratamiento en tiempo real, sino la predicción de distintas enfermedades.

Todo ello, tal como se ha mencionado anteriormente, es muy beneficioso para personas que viven en zonas rurales o donde se requiera un desplazamiento difícil de realizar por distancia, estado de salud etc. Con la finalidad de establecer el estado del arte de los sistemas RHM basados en BLE, se realiza un estudio acerca de sus implementaciones actuales y se realiza una serie de aportaciones para mejorar los sistemas ya implementados. Los sistemas de monitorización de atención médica a distancia pueden transmitir las mediciones desde distintos protocolos como podrían ser Wi-Fi, Bluetooth o incluso Zigbee. La mayoría de los dispositivos médicos se decantan por el uso de Bluetooth, más concretamente de BLE (Bluetooth Low Energy) debido entre otros muchos motivos al poco coste energético que supone su uso.

Estos sistemas ya se encuentran presentes en la sociedad desde hace un par de años, aunque su uso se intensifica a raíz de la pandemia. A continuación, se detallan algunos de estos sistemas que ya has sido implementados con éxito.

1.2.1 Bwell

Tal como expone el artículo [8] Bwell es un sistema de monitorización de pacientes a distancia implementado en diversos hospitales de Barcelona que permite la visita y control no presencial de pacientes con insuficiencias cardíacas.

El registro de mediciones obtenidas se realiza a través de dos dispositivos: una báscula para controlar el peso y un tensiómetro para el pulso y la tensión conectadas mediante BLE (Bluetooth Low Energy). Adicionalmente, se cuenta con una serie de formularios, que al igual que las mediciones disponen de una serie de alarmas sobre irregularidades y riesgos. Con esto, BWell consigue limitar las visitas y hospitalizaciones innecesarias, así como garantizar la correcta atención a personas con movilidad reducida que no pueden desplazarse hasta su centro hospitalario. El sistema contiene las características que se pueden ver en la Figura 4.

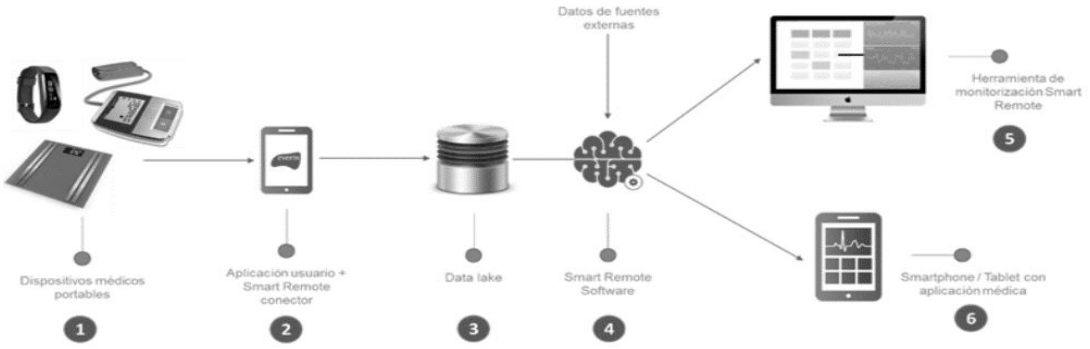


Figura 4. Características técnicas de BWell
Fuente. eSmartCity.es [8]

Por una parte, los dispositivos médicos portables que envían los datos a través de BLE a la aplicación móvil del usuario. Esta aplicación es escalable, rápida e intuitiva y su uso no depende del sistema operativo. Esos datos se almacenan en Data Lake [9] que permite la persistencia grandes cantidades de datos (estructurados, no estructurados o semi estructurados) para su posterior procesamiento y análisis mediante el BWell Software, el cual emplea análisis de datos para la toma de decisiones. El resultado de todos estos procesos y datos puede ser visualizado por los médicos desde la herramienta de monitorización web o desde la aplicación móvil.

1.2.2 VitalOn

El sistema VitalOn [10] ofrece un nuevo concepto al unificar en una única plataforma la teleasistencia, la telemedicina y el bienestar gracias a la monitorización de pacientes con una amplia gama de dispositivos médicos que se comunican mediante Bluetooth Low Energy.

Los dispositivos desarrollados por [11] se utilizan para ofrecer una monitorización para personas mayores y enfermos crónicos: pulsioxímetro, báscula inteligente, monitor de presión arterial, termómetro, medidor de glucosa en sangre, pulsera de actividad/sueño y sensor de caídas. Todos los datos se recopilan a través de aplicaciones móviles, las cuales además ofrecen soluciones concretas para abordar el envejecimiento y las infecciones crónicas como la diabetes, la hipertensión y las insuficiencias cardíacas. Esto, según el Dr. Haim Amir, fundador y presidente de Essence SmartCare comenta en el artículo [10], proporciona calidad de vida e independencia a los pacientes sobretodo en el caso de las personas mayores que conviven solas.

1.2.3 Neebo

Neebo [12] es una pulsera wearable para niños de entre 0 y 5 años diseñada para monitorizar a los niños pequeños proporcionándoles una mayor seguridad a sus padres. Las funciones que implementa este dispositivo son:

- Monitorización de la frecuencia cardíaca.
- Seguimiento de los niveles de oxígeno.
- Sistema de escucha a escondidas, para poder escuchar lo que sucede alrededor del niño.
- Monitor de actividad y de duración, notificando a los padres cuando el niño está despierto. Esta última se activa durante el modo sueño.
- Estado térmico, medición de temperaturas que permite detectar la más mínima febrícula para poder detectar en la mayor brevedad posible cualquier enfermedad.
- Estadísticas sobre las actividades, los hábitos y el bienestar diarios.
- Sistemas de alerta sonoras y visuales hasta que se confirme la recepción.

Todas estas funciones son vistas desde una aplicación móvil (únicamente iOS) que recibe las mediciones a través de BLE (Bluetooth Low Energy) y que los sube a la nube mediante Wi-Fi. Hay que destacar que no es necesario que el dispositivo móvil esté cerca de la pulsera, por ejemplo al dormir en habitaciones separadas, ya que también envía los datos por BLE a la base de carga de la pulsera. La aplicación permite múltiples cuidadores y la duración de la batería es de 3 días gracias al uso de la tecnología de bajo consumo BLE.

A modo de resumen, en la Tabla 1 se muestra una comparación de los 3 sistemas analizados.

Tabla 1. Comparación sistemas RHM reales

Nombre sistema	BWell	VitalOn	Neebo
Grupo	Pacientes con problemas cardíacos	Personas mayores y enfermos crónicos	Niños entre 0 y 5 años
Monitorización	Peso, pulso y tensión	Pulso, oxígeno, peso, presión arterial, glucosa, sueño, actividad y caídas	Frecuencia cardíaca, oxígeno, sonido, actividad y temperatura
Dispositivos	Báscula y tensiómetro	Pulsioxímetro, báscula inteligente, monitor de presión arterial, termómetro, medidor de glucosa en sangre, pulsera de actividad/sueño y sensor de caídas.	Todo en una pulsera wearable
Lugar	Domicilio paciente	Domicilio paciente	Wearable
Plataforma/s	Dos (médico y paciente)	Si	Si
Aplicación	Móvil y web	Móvil	Móvil
Procesamiento de datos	Si	Si	Si
Nodo central	Si (aplicación móvil)	Si	Si (base de carga de la pulsera y aplicación móvil)

2 Objetivos

Según lo detallado en las secciones anteriores, a continuación se define el objetivo y principal de este TFG así como los distintos objetivos secundarios

2.1 Objetivo primario

- Desarrollar un framework de asistencia sanitaria remota para monitorización, atención y valoración médica.

2.2 Objetivos secundarios

- Analizar la problemática expuesta por los médicos y realizar una búsqueda soluciones, considerando el estado del arte para determinar el estado actual de dichas tecnologías.

- Diseñar e implementar un sistema de captación de parámetros fisiológicos, como pueden ser el nivel de oxígeno en la sangre, presión arterial, temperatura, ritmo cardíaco y peso del paciente.

- Diseñar e implementar diversas aplicaciones web que servirán de interface médico-paciente. El diseño se establecerá en base a las propuestas del personal sanitario, siguiendo los estándares de usabilidad y experiencia de usuario, teniendo en cuenta que los pacientes serán mayoritariamente personas de edad avanzada.

- Diseñar e implementar el backend del sistema que constará de una API REST que permitirá gestionar los recursos almacenados en una base de datos MySQL cumpliendo los requisitos de confidencialidad e integridad del Reglamente General de Protección de Datos).

- Diseñar e implementar el sistema de comunicación entre los distintos dispositivos para poder enviar y recibir mensajes en tiempo real de la manera más segura posible.

- Evaluar los desarrollos realizados mediante distintas pruebas de validación.

3 Metodología empleada

Para alcanzar los objetivos descritos en la sección anterior, a continuación se detalla la metodología de trabajo que se ha seguido durante la implementación del proyecto:

- **Entrevista con los médicos:** se comenta la problemática y se establecen los requisitos del proyecto.

- **Planteamiento de la solución:** en función de los requisitos establecidos en la reunión con los médicos, se plantea la solución para:
    - **Captura de mediciones:** teniendo en cuenta los dispositivos de los cuales se dispone y la tecnología por la cual se comunican (BLE) se realiza un análisis de las tecnologías candidatas (Arduino, Python y MicroPython) y se concluye desarrollar una aplicación en Python que se ejecutará en un dispositivo Raspberry (en este caso, la versión utilizada es una Raspberry Pi 3B pero bastaría con una mucho más económica como la Raspberry Pi Zero 2 W) y que permita conectarse mediante BLE a los distintos dispositivos (tensiómetro, báscula…) así como comunicarse con ambas aplicaciones.

    - **Backend:** se diseña la estructura de la base de datos. La base de datos se desarrolla en MySQL y se puede acceder a ella mediante una API REST realizada en PHP mediante el MVC.

    - **Aplicación web médicos:** se realizan mockups de la página web para los médicos. Se les envía un video a los mismos para que les den validez.
      Los mockups se realizan mediante el software de Axure [13] y se decide implementar la aplicación mediante Angular.

    - **PWA (Progressive web apps) pacientes:** se realizan mockups de la web para los pacientes. Se les envía un video a los mismos para que les den validez.
      Los mockups se realizan mediante el software de Axure y se decide implementar la aplicación mediante Angular.

    - **Comunicación entre los sistemas:** se tienen en cuenta distintas opciones, optando finalmente por el uso de AWS (Amazon Web Services) IoT (comunicación MQTT).

- **Desarrollo:** en esta fase, de desarrollan todos los puntos anteriores.

- **Validación:** se establecen distintos tipos de pruebas para evaluar los desarrollos realizados.
    - **Testing manual:** durante el desarrollo del proyecto se está constantemente probando la aplicación e intentando que sea lo más robusta posible.

    - **Selenium** [14]**:** se realizan testeos sobre las principales secciones de la página.

    - **Médicos:** se espera la implementación del sistema en un hospital de Castellón, en el cual se pueda probar el sistema con la mayor profundidad posible. Esta fase, está en desarrollo.

La organización de esta memoria se puede ver en mayor profundidad en el índice, pero consiste en:

- **Análisis problemática, y búsqueda de solución:** búsqueda de información sobre la situación actual en España y se analiza lo que es un sistema RHM y como podría ser de ayuda.

- **Estado del arte:** búsqueda sistemas actuales semejantes al cual se quiere realizar.

- **Desarrollo:** análisis de las múltiples opciones planteadas y la elección de la utilizada. Además se explica todo el proceso de desarrollo tanto hardware, como software (backend y frontend).

- **Validación:** realización de múltiples tests para encontrar problemas en la aplicación y se intentan solucionar.
- **Conclusiones y trabajo futuro:** se exponen los resultados y se plantean mejoras y futuras tareas.

4 Desarrollo

La fase de desarrollo es la tercera fase expuesta en la sección anterior. En ella, se explica que es un RHM y las tecnologías necesarias para llevarlo a cabo (MQTT, AWS y BLE) incluyendo las distintas capas que forman el proyecto (capa física, frontend, backend y comunicación entre dispositivos).

Tal como se ha podido observar anteriormente, los RHM cuenta más o menos con una estructura semejante a la de la **¡Error! No se encuentra el origen de la referencia.**.

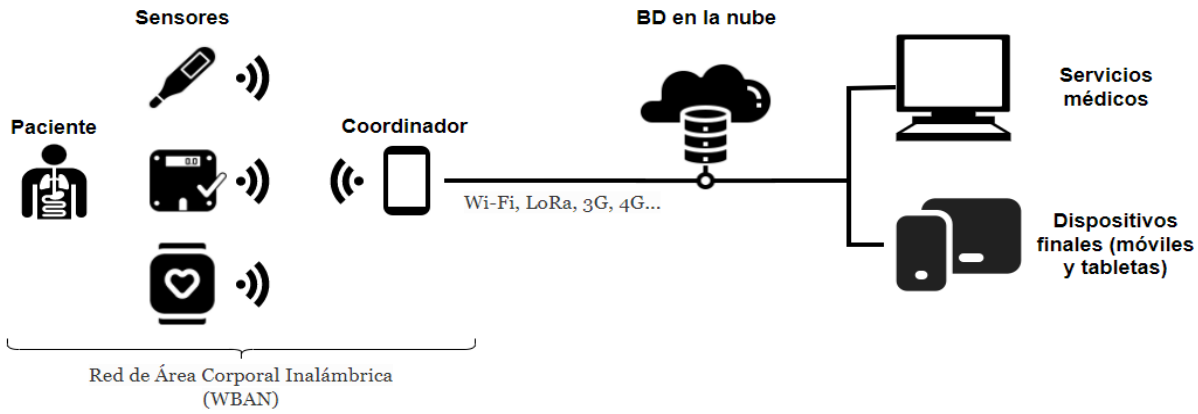


Figura 5. Características sistema RHM

En primer lugar, observamos una Red de Área Corporal Inalámbrica que se establece al implementar distintos sensores en el cuerpo humano o cerca de él. Estos dispositivos se pueden comunicar con el coordinador (ordenador, dispositivo móvil o tableta, placa…) mediante distintos protocolos tal como se ha visto anteriormente, siendo uno de los más usados en la actualidad el BLE (Bluetooth Low Energy) por su bajo consumo energético.

Posteriormente, los datos recogidos por el coordinador se suben a la base de datos. La subida de datos puede ser por diferentes protocolos en función de la conectividad de la zona, por ejemplo, en zonas rurales hay mejor conexión LoRa aunque la posibilidad de transferir datos mediante 3G o 4G no está descartada en ningún caso. Finalmente, los datos son procesados, analizados y visualizados desde los distintos dispositivos finales como los servidores médicos, páginas web o aplicaciones para móviles o tabletas.

Teniendo en cuenta que los requisitos del sistema son:

- Obtener los datos de los distintos dispositivos médicos y poder visualizarlos en tiempo real.
- Poder establecer una comunicación médico-paciente en tiempo real.
- Almacenar y gestionar los datos necesarios con la finalidad de poder visualizar los datos de los pacientes en todo momento y asegurar correcto funcionamiento del sistema.

Los requisitos anteriormente mencionados implican que la estructura de nuestro sistema sea muy similar a la anterior, quedando tal y como se puede visualizar en la Figura 6.

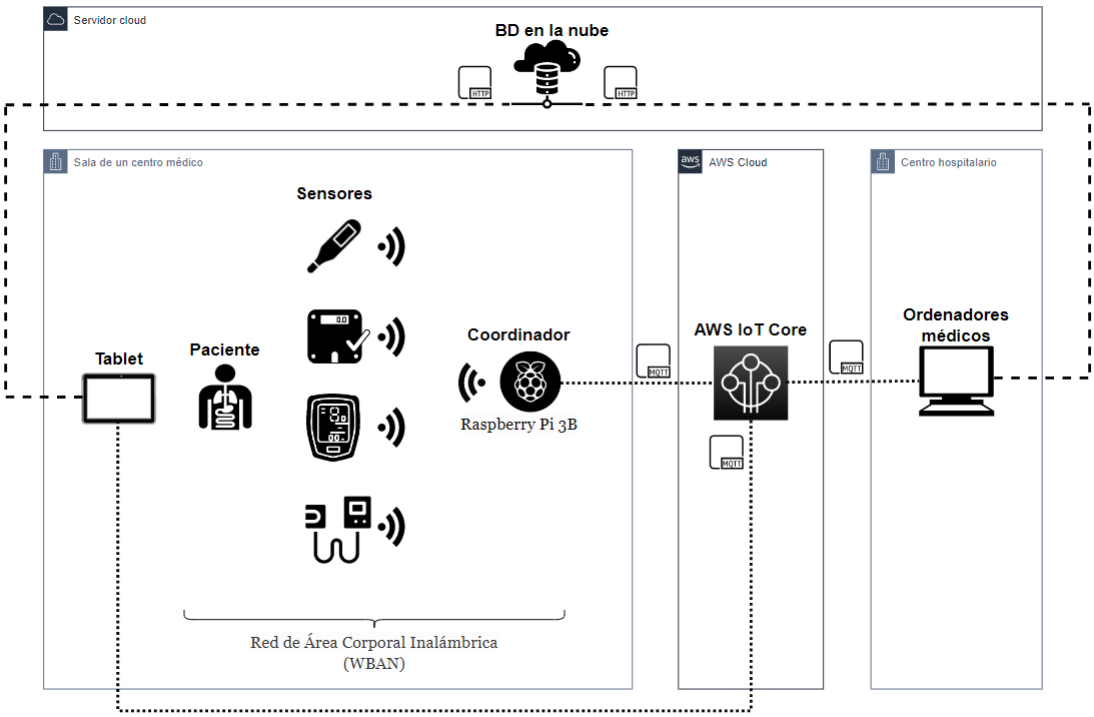


Figura 6. Características sistema RHM propio

Los cuatro dispositivos que monitorizarán al paciente serán el termómetro, el tensiómetro, la báscula y el pulsioxímetro. Todos ellos se comunicarán mediante BLE con el coordinador, que será una Raspberry Pi 3B. La forma de enviar peticiones para tomar mediciones de los sensores será desde la consulta de la puesta sanitaria o el centro sanitarios. El enfermero, técnico o médico solicitará mediante la aplicación web la obtención de ciertos datos para determinados pacientes. Es decir, cuando el médico quiera enviar una orden o notificación al paciente o al coordinador de una determinada sala, lo hará mediante MQTT (en el caso de las páginas web, usando WebSockets [12]).

Por otra parte, el coordinador no dispondrá de acceso a la base de datos, si no que lo harán únicamente las aplicaciones, tanto la de la sala a la que asistirá el paciente a realizar la consulta remota (sala de un centro médico) como la de la consulta del médico (centro hospitalario). La conexión con la base de datos será mediante Wi-Fi, haciendo uso del protocolo HTTPS (HTTP con TLS o SSL). Por tanto, cuando el médico solicite una medida, el coordinador se la proporcionará y será él mismo quien la suba a la base de datos.

A continuación, se van a presentar las tecnologías empleadas en el proyecto en el consiguiente orden:

- Protocolo de comunicación MQTT.
- AWS (Amazon Web Service).
- Protocolo de comunicaciones Bluetooth Low Energy.
- Estructura de la capa física.
- Backend.
- Frontend.
- Comunicación entre dispositivos.

4.1 Protocolo de comunicación MQTT

El sistema a desarrollar, tal como se ha indicado en los requisitos, ha de ser en tiempo real, ya que se necesita poder obtener las mediciones de forma instantánea. El IoT (Internet of Things) es una agrupación de objetos físicos que se conectan e intercambian datos a través de internet u otras redes de comunicación. Estos objetos pueden contener sensores, software y otras tecnologías. Por tanto, nuestro sistema es un sistema IoT ya que vamos a disponer de distintos dispositivos (la aplicación del médico, la del paciente, el coordinador y los dispositivos para tomar las medidas) que se comunican entre sí para establecer una consulta remota y conocer los resultados de las mediciones. Para ello se plantearon dos grupos de protocolos: los de tipo cliente/servidor y los de tipo publicar/subscribir.

La opción de cliente/servidor se caracteriza por la necesidad de conocer ciertos datos como la IP para establecer comunicaciones previas, con lo cual, no es tan escalable como la de publicador/subscriptor, que permiten la suscripción a un topic y esperar a recibir una respuesta. Existen múltiples protocolos dentro de estos dos grupos, como por ejemplo: OPC, HTTP, MQTT, CoAP, DDS o AMQP. En este caso, se pondrá el foco de atención en HTTP y MQTT.

HTTP (Hypertext Transfer Protocol) es un protocolo de tipo cliente/servidor. Su estilo de arquitectura de software REST (Representation State Transfer) consiste en un modelo de datos previos donde los clientes pueden acceder a los recursos previa petición. Cada vez que se realiza una petición, se ha de realizar una nueva conexión. Al analizarlo desde el punto de vista de nuestro sistema, el hecho de realizarlo mediante peticiones HTTP no sería eficiente, ya que, por ejemplo, habría que estar constantemente recargando o volviendo a solicitar datos, por lo que no se podría pretender esperar que llegase un dato en cualquier momento, cosa que dificultaría el desarrollo y ralentizaría la página. Por ende, este protocolo quedó descartado.

En cambio, MQTT (Message Queuing Telemetry Transport) es un protocolo de tipo publicar/subscribir cuya principal diferencia con HTTP es que la conexión se mantiene abierta y se reutiliza en cada comunicación. Esto proporciona múltiples ventajas en comparación a HTTP, que son escalabilidad y asincronismo. Adicionalmente, MQTT tiene un consumo energético muy bajo, interesante para dispositivos que requieren una elevada autonomía. Su necesidad de potencia es baja, es decir, puede ponerse en cualquier pequeño dispositivo (como un ESP32 o una Raspberry) sin que se vea afectado el rendimiento. El ancho de banda que necesita es mínimo, otro factor muy interesante para el proyecto, ya que está destinado a zonas más aisladas y con menos conexión. Por último, los QoS (Quality of Service) es un sistema de acuerdo entre remitente y receptor para asegurar que el mensaje es enviado y llega determinadas veces.

4.1.1 Arquitectura MQTT

La arquitectura de MQTT [15] se basa en: publicadores, subscriptores y el broker.

Los clientes (publicadores y subscriptores) establecen una conexión TCP/IP con el broker. Esta conexión se mantiene abierta hasta que cada cliente la cierra. El cliente envía un mensaje de CONNECT con parámetros básicos como su client-id. El bróker contesta con un mensaje CONNACK que indica si la conexión ha sido realizada con éxito o no. Esta comunicación se puede observar en la Figura 7.

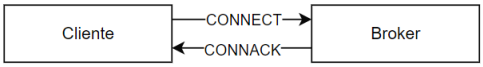


Figura 7. Conexión MQTT

Posteriormente, se puede ver que si el cliente quiere enviar un mensaje utiliza un PUBLISH que contiene el topic y le payload (mensaje). A la hora de publicar un mensaje existen diferentes niveles de QoS:

- **QoS 0:** cómo se puede ver en la Figura 8 envía el mensaje como máximo una vez, es decir, en ningún momento, se asegura de que cada uno de los clientes subscritos al topic lo reciba.

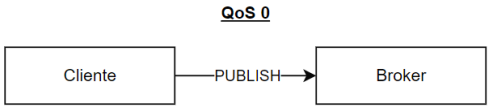


Figura 8. Publicación mensaje MQTT QoS 0

- **QoS 1:** se asegura de que el mensaje lo reciba al menos una vez el receptor gracias a que cuando el cliente recibe el mensaje envía un PUBACK. Mientras no se reciba ese mensaje, de forma periódica se intentará reenviar el mensaje hasta que lo reciba una vez. Esto se puede visualizar en la Figura 9.

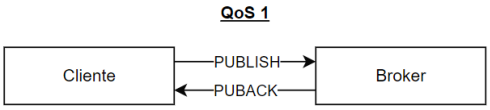


Figura 9. Publicación mensaje MQTT QoS 1

- **QoS 2:** este nivel se asegura que el mensaje solo sea recibido una vez por cada dispositivo, tal como se puede ver en la Figura 10. Para ello, se utiliza el mensaje PUBREC.

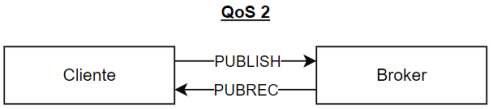


Figura 10. Publicación mensaje MQTT QoS 2

Para subscribirse y de-suscribirse se emplean mensaje de tipo SUBSCRIBE y UNSUBSCRIBE, que el servidor responde con SUBACK y UNSUBACK, tal como se puede observar en la Figura 11.

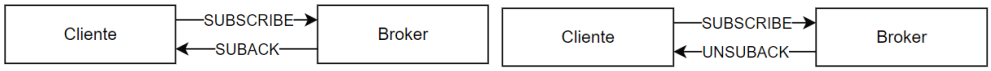


Figura 11. Proceso subscrición y de-subscripción MQTT

De manera periódica, el cliente se envía mensajes de tipo PINGREQ que el broker contesta con PINGRESP para mantener la sesión viva, tal como se puede ver en la Figura 12. Finalmente, los clientes envían mensajes de tipo DISCONNECT para cerrar la conexión.

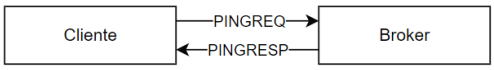


Figura 12. Proceso mantenimiento sesión MQTT

4.2 AWS (Amazon Web Services)

Tal como se ha visto en la anterior sección, se va a realizar la comunicación mediante MQTT pero podría suponer un problema de seguridad el que la información no vaya cifrada, puesto que cualquiera que supiera el *topic* al cual subscribirse podría escuchar toda la información. Se plantearon diversas tecnologías cloud que ofrecen un mayor nivel de seguridad en la comunicación como AWS, Google Cloud o Microsoft Azure, de las cuales se eligió AWS por los motivos descritos posteriormente.

AWS [13] es una plataforma en la nube que cuenta con más de 200 servicios, ofreciendo todo tipo de tecnologías (cómputo, almacenamiento, bases de datos, aprendizaje automático, inteligencia artificial, internet de las cosas…) con una implementación más sencilla, más segura y adaptada a distintos tipos de presupuestos. AWS cuenta con la mayor selección de servicios web, incluyendo una amplia gama para IoT como: IoT Greengrass, IoT Core, IoT Device Management, IoT Device Defender y IoT Analitics. Cada uno de estos servicios queda perfectamente identificado en la Figura 13 (aunque también existen otros como IoT SiteWise, IoT Things Graph o FreeRTOS) obtenida de [16].

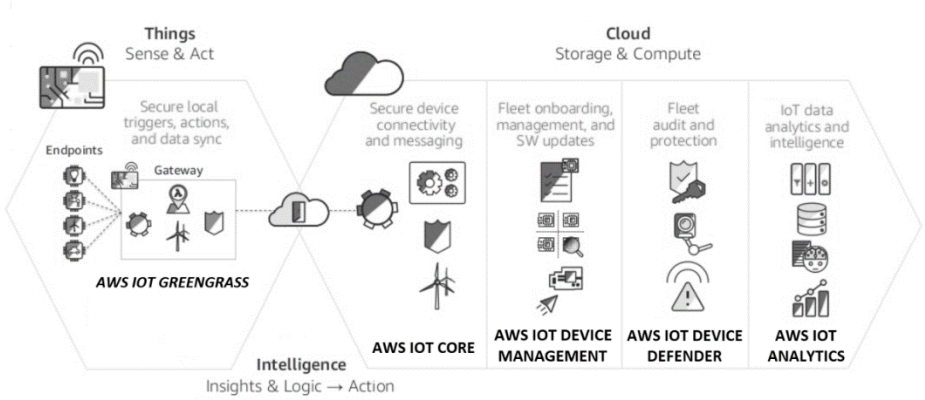


Figura 13. Arquitectura de servicios principales de la plataforma AWS IoT
Fuente. Telefónica Tech [16]

La infraestructura que sigue AWS IoT Core es la misma que se puede observar en la Figura 14.

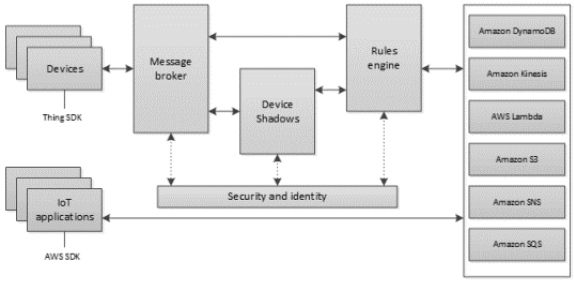


Figura 14. Elementos core requeridos para el funcionamiento de AWS IoT
Fuente. Telefónica Tech [16]

Por tanto, la elección de usar AWS reside en las siguientes ventajas [17]:

- Gran cantidad de servicios que permiten el mejor desarrollo de un proyecto cloud de cualquier tipo (SaaS, PaaS o IaaS).

- Flexibilidad y escalabilidad.

- Mayor red de servidores (en comparación a otras plataformas como Google Cloud o Microsoft Azure).

- Gran nivel de seguridad gracias a:
    - Certificaciones y acreditaciones. Se requiere el uso de certificados X.509 como credencial de identificación y acceso, es decir, solo nuestros dispositivos van a ser capaces de comunicarse entre ellos.

    - Soporte 24/7 y en distintas zonas geográficas.

    - Control y seguimiento del acceso de los usuario a través de IAM.

    - Cifrado de datos para su almacenamiento.

    - Posibilidad de configurar reglas de firewall públicas, privadas o mixtas.

- Bajos costes, únicamente se paga por lo que se usa. AWS cuenta con el pago por horas o por incidencias. En este caso, como se estará conectado a la región de us-west-1, suponiendo que el dispositivo está encendido

24/7, el coste por año por cada dispositivo será de 0,042$, lo cual hace que el desarrollo sea muy económico. Se pueden consultar más variaciones para los costes a través de la web oficial de AWS [18].

- Alto nivel de compatibilidad y de estandarización.
- Gran capacidad analítica y escalabilidad.
- Implementación de dispositivos LoRa, que podrían ser muy importantes ante la baja conectividad en la que puede llegar a desarrollarse el proyecto.

4.3 Protocolo de comunicaciones Bluetooth Low Energy

Bluetooth es un protocolo de redes inalámbricas de área personal (WPAN) estandarizado (ver [19] utilizado para la transmisión de datos. Emplea una modulación de tipo espectro por salto de frecuencia de 2.4GHz con un total de 79 frecuencias; lo que hace que sea menos propenso a interferencias y que sus transmisiones sean difíciles de interceptar. Este protocolo, tiene un alcance máximo de 1Km (suponiendo que su trayectoria es en línea recta y sin interferencias) y un tiempo de retardo reducido que ronda entre los 5 y los 10ms.

Desde la versión de Bluetooth 4.0, existe otro subconjunto adicionalmente del clásico llamado BLE (Bluetooth Low Energy). El Bluetooth de baja energía se caracteriza, entre otras características (como su robustez o su bajo tiempo para despertar o reconectar), por consumir mucho menos que el clásico, lo cual favorece su uso en dispositivos de menor tamaño y potencia. En este caso, todos los dispositivos médicos empleados para la realización del proyecto utilizan únicamente BLE, lo que asegura un menor consumo energético. Los dispositivos de la marca Lifevit [20]. utilizados son:

- Tensiómetro de brazo inteligente BMP-200 Wireless.
- Termómetro inteligente Kelvin.
- Pulsioxímetro inteligente OL-750.
- Báscula inteligente Kryos.

Cada uno de los dispositivos, enviará sus tramas de datos a una Raspberry Pi 3B [21], la cual será la encargada de recoger, tratar y subir los correspondientes resultados. Las maneras en que los distintos dispositivos puede enviar las tramas son:

- **Broadcast:** El dispositivo BLE envía tramas de datos con información relevante al dispositivo que puede observar cualquiera. Los datos que suele proporcionar son:
    - Nombre del dispositivo.
    - Dirección MAC del dispositivo.
    - Información acerca del fabricante (manufacturer).
    - Flags.
    - Identificador UUID del servicio que proporciona entre otros.

- **Notificaciones:** Para poder recibir notificaciones por parte de un dispositivo BLE es necesario esperar que se produzca la recepción (de forma asíncrona) a través de una característica contenida por un servicio. Es por ello, que el proceso para leer notificaciones reside en:
    - **Paso 1:** Conectarse al dispositivo BLE mediante el conocimiento de su MAC.
    - **Paso 2:** Obtener el servicio correspondiente dada su UUID (para obtener este valor, se puede consultar en algunos casos el manual de desarrollo del producto o utilizar aplicaciones como nRF Connect [22].
    - **Paso 3:** Obtener la característica que nos interese propia del servicio sabiendo su UUID (se puede obtener de igual forma que la UUID del paso 2).
    - **Paso 4:** Escribir una característica para poder registrar las notificaciones. El valor de esta característica será de b"\x01\x00" tal como indica el siguiente artículo [23].

4.4 Estructura de la capa física

La capa física es la primera capa del modelo OSI. Esta tiene la función de codificar en señales los dígitos binarios que representan las tramas de la capa de enlace de datos así como transmitir o recibir dichas señales a través de un medio físico (wireless o con cable) que conectan los distintos dispositivos de red.

En este caso, los medidores como el termómetro, el tensiómetro, la báscula o el pulsioxímetro se conectan mediante la tecnología wireless BLE a la Raspberry. Por tanto, en esta sección se va a explicar el proceso a seguir para interconectar los dispositivos anteriormente mencionados. Más concretamente, por una parte se van a exponer las distintas clases diseñadas para poder comunicar el controlador con los medidores a través de la librería Bluepy [24] y por otra parte las especificaciones de cada uno de los dispositivos así como su conexión y tratamiento de datos.

4.4.1 Clases generales

Las clases generales (diseñadas para evitar la repetición innecesaria de código) se pueden dividir distintos grupos:

- Clases destinadas a la obtención de tramas BLE bien a través de información proveniente de tramas broadcast o bien de notificaciones. Estas clases utilizan como base la librería Bluepy. Hay dos clases en este grupo:
    - ScanBTLE – Tramas broadcast.
    - CharacteristicsBTLE – Tramas notificaciones.
- Clase destinada a funciones de utilidad. Este archivo contiene distintas funciones que se pueden utilizar en cualquier momento, como por ejemplo para *parseo* de datos o para convertir un número entero sin punto ni coma en un decimal.
- Clase para conectar y subscribirse y publicar en *topics* de AWS IoT a través de MQTT.
- Clase para analizar y solicitar los valores de uno u otro dispositivo.

4.4.1.1 ScanBTLE – Tramas broadcast

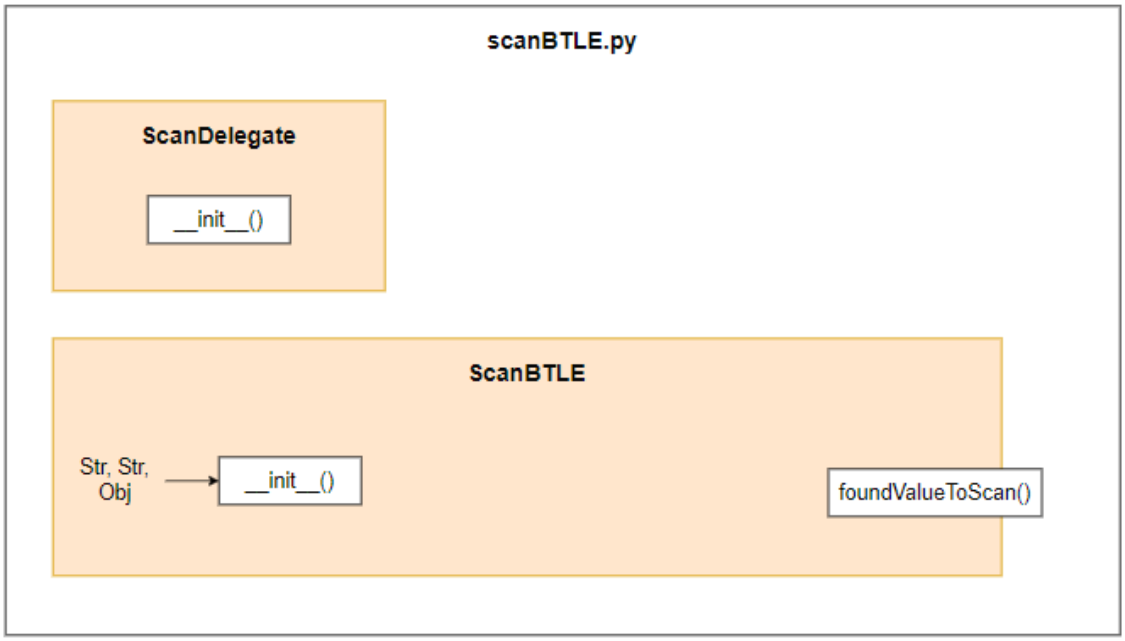


Figura 15. Diseño clases scanBTLE.py

La clase *ScanBTLE* es la encargada de buscar una propiedad concreta dentro de las tramas broadcast enviadas por el dispositivo. Para ello, requiere el uso de la librería Bluepy, más concretamente de las clases *Scanner* y *DefaultDelegate*.

Al inicializarse la clase, es necesario pasarle la MAC del dispositivo que va a buscar, el nombre de la descripción y el objeto que ha generado la llamada (en este caso, como solo se ha implementado un dispositivo que transfiera las tramas por *broadcast*, será de tipo *Weight*). Posteriormente, la llamada a la función *foundValueToScan*() es la encargada de crear un objeto *Scanner*, asignarle un delegado y empezar a buscar dispositivos. Cuando una de las tramas *broadcast* de los dispositivos coincide con los datos pasados al constructor, se procede al tratamiento de los datos. En caso de que no se encuentre la trama deseada se llamará a la función *notificateError*() propia del objeto pasado a la función.

4.4.1.2 CharacteristicsBTLE – Tramas notificaciones

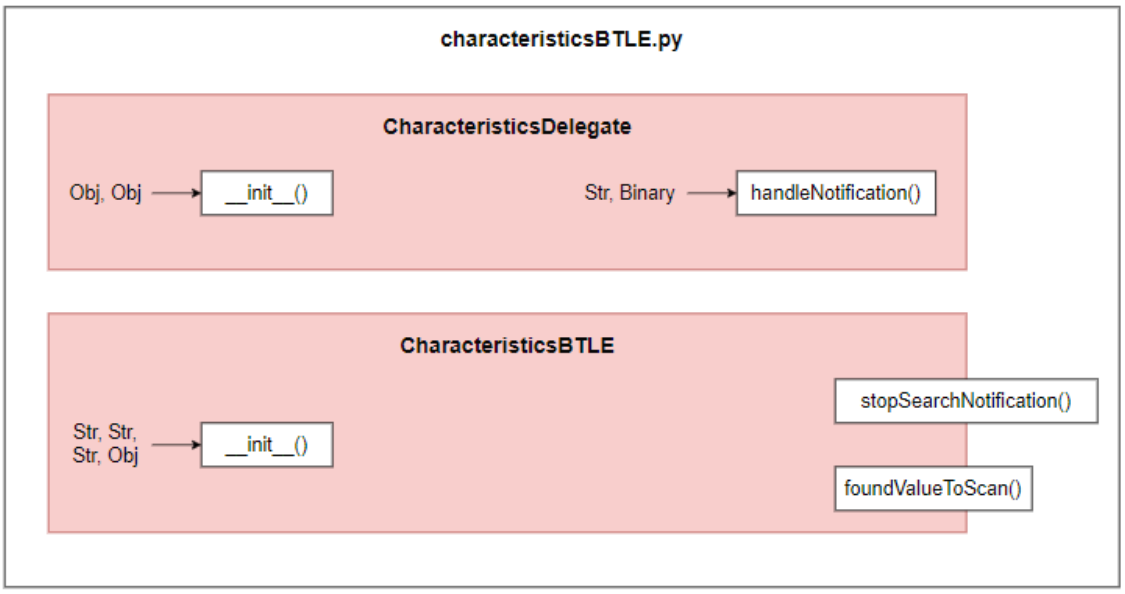


Figura 16. Diseño clases characteristicsBTLE.py

La clase *CharacteristicsBTLE* es la encargada de obtener tramas de datos enviadas por los dispositivos mediante notificaciones. Para ello, precisa del uso de la librería Bluepy (al igual que *ScanBTLE*), más concretamente las clases *Peripheral* y *DefaultDelegate*.

En este caso se observa que al instanciar la clase es necesario pasarle 4 parámetros. Estos parámetros son la MAC del dispositivo al que conectarse, la UUID del servicio que obtener, la UUID de la característica que se quiere buscar dentro del servicio y por último el objeto que ha generado la llamada (en este caso, podrá ser de tipo *BloodPreassure*, *Temperature* o *PulseOximeter*). Es importante destacar que la recepción de notificaciones es asíncrona y que estaremos esperando su llegada hasta encontrar la trama correcta mediante la función *foundValueToScan*().

En primer lugar, lo que hará la función es intentar conectarse al dispositivo correspondiente dada su dirección MAC. Si este proceso falla se llama la función que enviará un mensaje de error mediante MQTT a través del objeto paso a la misma. Si la conexión se realiza con éxito, procederá a obtener el servicio y la característica de este. Además, se asigna el delegado que será el que recibirá de forma asíncrona las notificaciones. Por último, es importante escribir sobre la característica puesto que es necesario (tal como se había comentado anteriormente) para pedirle al dispositivo que nos envíe tramas. Todo este proceso se puede ver más detalladamente en la Figura 17.

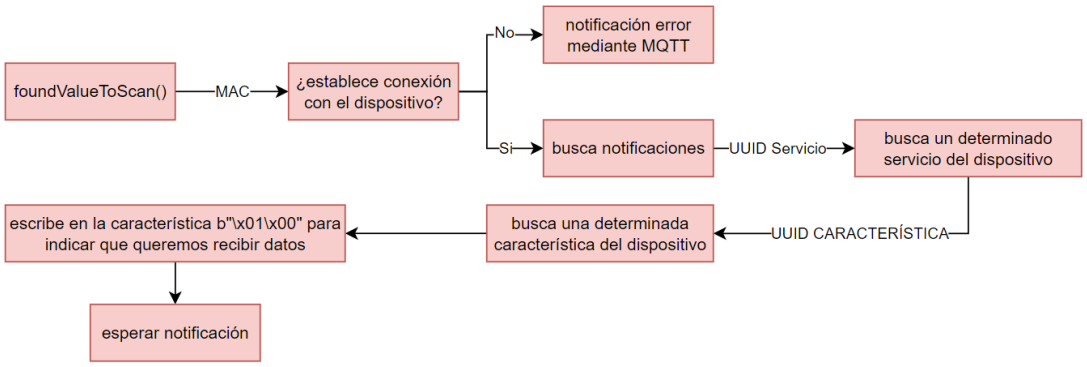


Figura 17. Flujo obtención de datos a través de una característica de un dispositivo BLE

Cada vez que se produzca una nueva notificación se llamará a *handleNotification*(), la cual se encargará de preguntar si la trama está dentro de unos requisitos preestablecidos. Si lo está, llamará a la función de *stopSearchNotification*() de la clase *CharacteristicsBTLE* para que detenga la búsqueda de nuevas tramas.

4.4.1.3 Utilities

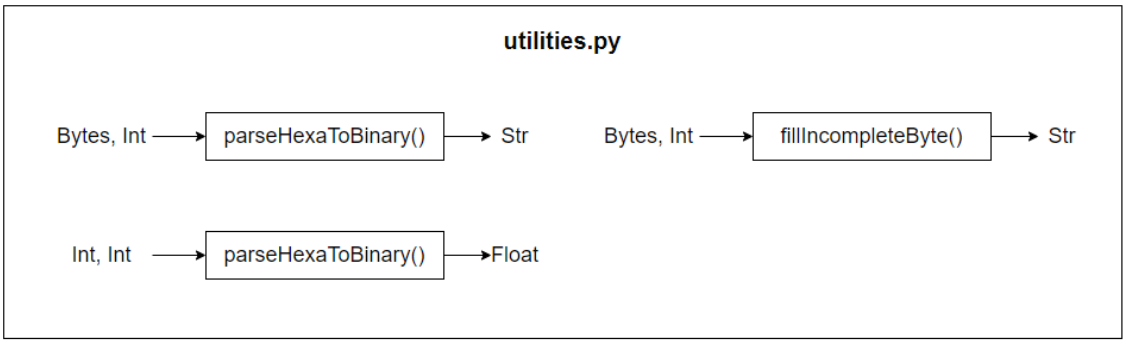


Figura 18. Diseño funciones auxiliares

Estas funciones son utilizadas para prestar ayuda en casos como el *parseo* de datos que son utilizadas en más de una clase a la vez; motivo por el cual se decide separarlas para evitar la repetición de código.

4.4.1.4 MqttAwsIotCore

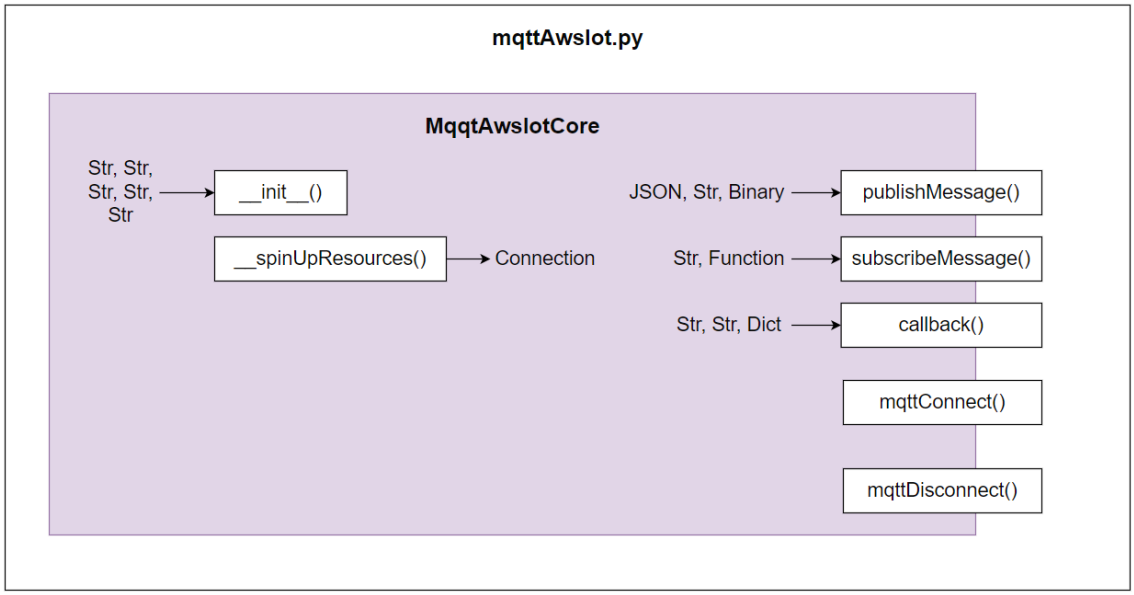


Figura 19. Diseño clase mqttAwsIot.py

Esta clase, es llamada sobre todo desde el *main* (desde el resto de los archivos que son llamados desde el *main*, la librería no funciona correctamente a causa de problemas con los hilos) con la finalidad no solo de establecer la comunicación con AWS IoT Core, sino también con la finalidad de poder subscribirse y publicar en determinados *topics*.

Al iniciar la clase, los parámetros de AWS IoT que se requieren son:

- El punto de acceso.
- El ID que le vamos a otorgar a este dispositivo, este ID ha de ser único, por ende, se utilizará la dirección física (MAC) del mismo.
- Dirección a:
  - El archivo del certificado (*objectName.cert.pem*).
  - El archivo de la clave privada (*objectName.private.key*).
  - El archivo de la autoridad certificadora (*root-CA.crt*).

Para obtener los certificados anteriores, se ha de crear un objeto en AWS. Este objeto deberá estar destinado a la plataforma Linux/OSX y utilizará la SDK de Python. Para ver más sobre el proceso, consultar la documentación oficial [25].

4.4.1.5 AnaliceOrder

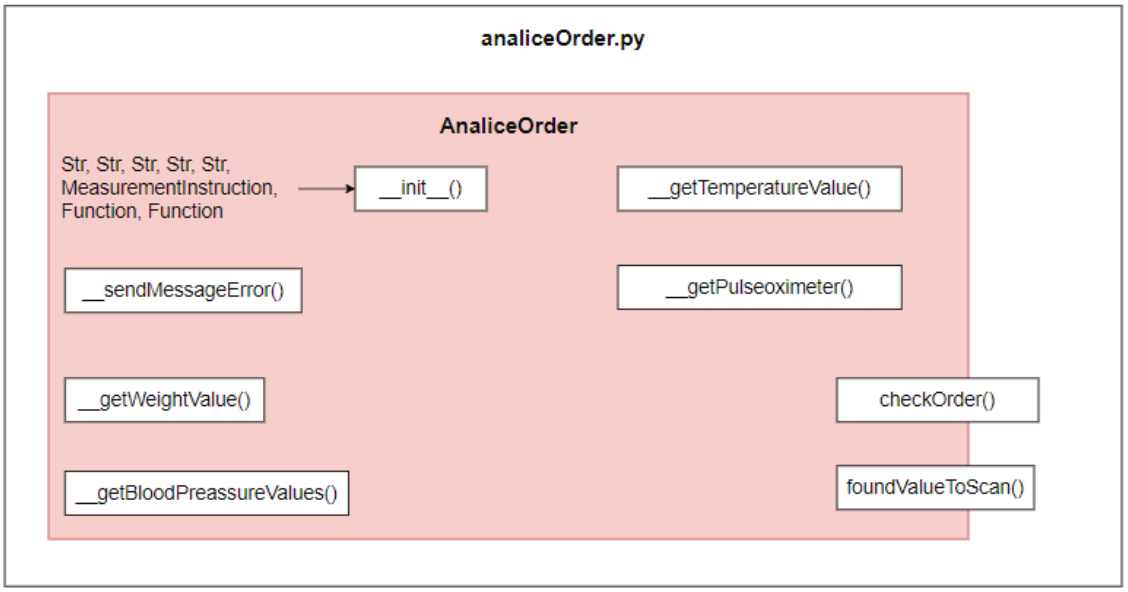


Figura 20. Diseño clase analiceOrder.py

Esta clase es la que ayuda a identificar el tipo de medida que se ha de tomar e instancia una u otra clase en función de ello. El objeto de tipo *MeasurementInstruction*, utiliza el mismo formato que para las aplicaciones web. La estructura que sigue es de tipo JSON y es como la siguiente:

MeasurementInstruction
```
{
  status: number,
  message: {
    data: {
      type: string,
      typeBBDD: string,
      time: Date,
      description: string,
      idRoom: number,
      macPatientDevice: string,
      coordinator: Coordinator
      sip: string,
      device: Device,
      iconClass?: string,
      measurementResult?: MeasurementResult[]
    },
    error: string
  }
}
```

Coordinator
```
{
  id?: number,
  mac: string,
  idRoom: number
}
```

MeasurementResult
```
{
  name: string,
  value?: number,
  units: string
}
```

Device
```
{
  id?: number,
  idCoordinator: number,
  mac: string,
  service: string,
  characteristic: string,
  description: string,
  type: string
}
```

4.4.2 Clases específicas

Las siguientes clases han sido desarrolladas para cada uno de los dispositivos que se va a conectar a la Raspberry. Estas clases son independientes para facilitar la implementación, pero todas siguen una estructura muy similar para facilitar su integración con las clases generales anteriormente mencionadas.

Los distintos dispositivos se pueden agrupar en dos grupos:

- Dispositivos que se comunican mediante tramas *broadcast*. En este grupo sólo se incluye la báscula inteligente Kryos.

- Dispositivos que se comunican mediante notificaciones. En este caso, residen todo el resto de los dispositivos: el tensiómetro de brazo inteligente BMP-200 Wireless, el termómetro inteligente Kelvin y el pulsioxímetro inteligente OL-750.

A continuación, se explica las características de cada uno de los dispositivos, así como las clases diseñadas para su implementación.

4.4.2.1 Báscula inteligente Kryos

La báscula inteligente Kryos permite obtener el peso del paciente haciendo un análisis completo de la composición corporal: IMC (Índice Masa Corporal), porcentaje de grasa corporal, nivel de hidratación, masa ósea y muscular, tasa de metabolismo basal, porcentaje de obesidad y edad metabólica del cuerpo. En este caso, la báscula ejerce una comunicación de tipo *broadcast* en la que envía tramas de datos a través del *Manufacturer* (aunque esto no sea lo habitual).

4.4.2.1.1 Características principales

Tabla 2 Características Báscula inteligente Kryos.

Características técnicas			Características dispositivo	
Dimensiones	300 x 300 x 30 mm		**Nombre del dispositivo**	Chipsea-BLE
Peso	1.7 Kg		**Dirección MAC**	5c:ca:d3:6b:4f:12
Rango arranque precisión	5 – 15 Kg		**Tipo de comunicación**	BLE mediante tramas broadcast
Rango pesaje	0.5 – 180 Kg		**Nombre descripción**	Manufacturer
Unidades división	100 gr			
Rango precisión	0.5 Kg			
Entorno operativo	**Temperatura**	-10 – 40 °C		
	Humedad	< 90%		

4.4.2.1.2 Trama de datos

Las tramas de datos enviadas por el dispositivo se corresponderían con el siguiente prototipo:

b'02010615FF**FFF0**01**043300**41CDB403FAFBFCFD5CCAD36B4F120C09436869707365612D424C45'

Tabla 3. Trama de datos Báscula inteligente Kryos

Byte	Descripción
0-1	FFF0 (Logo)
2	Número Versión Broadcast (De normal 01)
3	Propiedades del Mensaje (Ver Tabla 4)
4~5	Peso
6~9	No utilizado
10~11	
12~13	
14~19	Dirección MAC

Tal como se observa en la documentación oficial, el inicio de las especificaciones del paquete de transmisión empieza en los **bytes 0 y 1** con el Logo FFF0. El peso viene dado en los **bytes 4 o 4 y 5**. Es importante que, a la hora de convertir los bytes en formato hexadecimal a números enteros, se coja primero el byte 5 y luego el 4. En este caso sería: 0033 (hexadecimal) = 51 (entero). Un ejemplo con números decimales sería: 12F7 (hexadecimal) = 4855 (entero). Para obtener más información sobre las unidades en las que se ha tomado la medición o donde poner el punto decimal, es necesario hacer uso del **byte 3**.

Tabla 4. Propiedades mensaje en la trama de datos Báscula inteligente Kryos

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reservados		Unidades Seleccionadas		Punto Decimal		Reservado	

	00 = Kg 01 = Jin 10 = LB 11 = ST:LB	00 = 2 decimales 01 = 0 decimales 00 = 1 decimal

Al descomponer el **byte 3** en bites el resultado será: 01000000. Al analizarla en profundidad, se puede obtener que dispone de 2 decimales y que las unidades obtenidas son los Kg. En ambos casos, son los valores por defecto.

4.4.2.1.3 Clase Weight

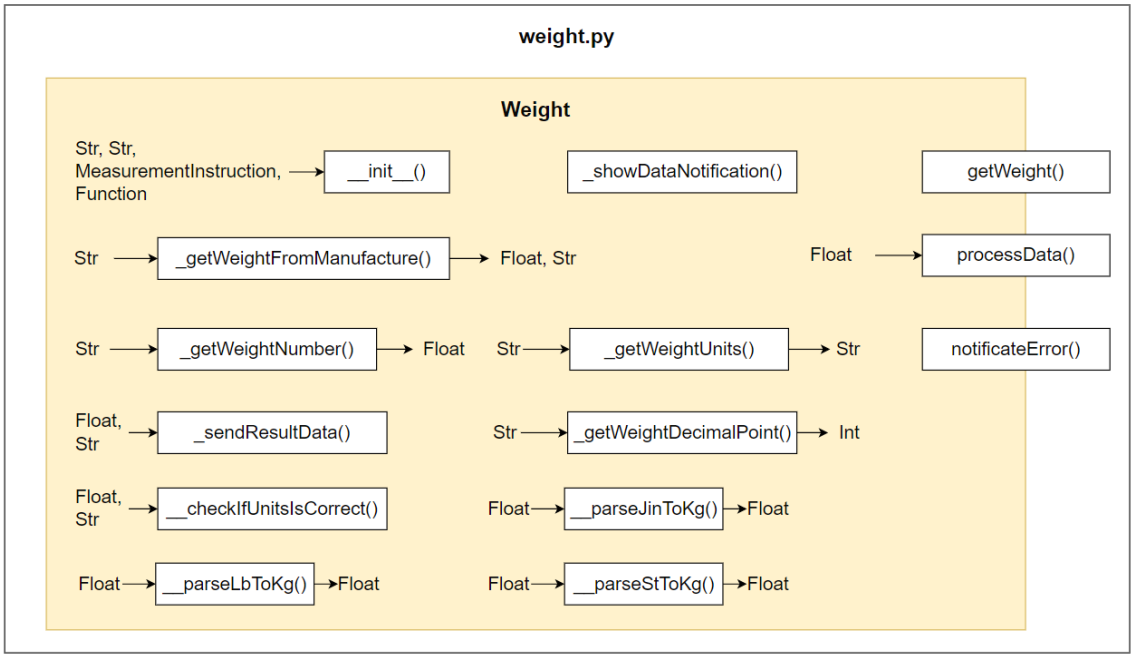


Figura 21. Diseño clase weight.py

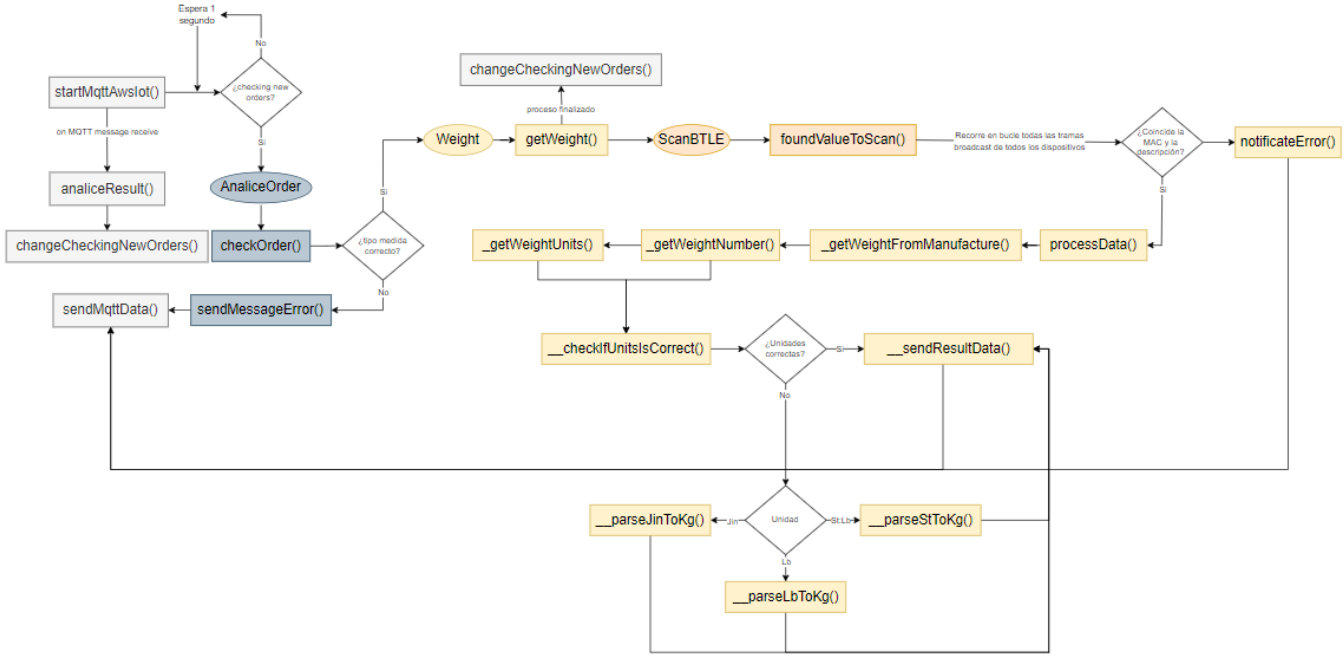


Figura 22. Diagrama de flujo para la toma del peso

Al iniciarse el programa, la función *startMqttAwsIot()* inicializa la conexión con AWS IoT y se subscribe a un *topic* para recibir las ordenes (es la unión de la constante *ORDERS_TOPIC* y la dirección *MAC* del dispositivo).

Cuando de manera asíncrona llega un mensaje, se instancia la clase *AnaliceOrder*, a la cual se le pasan todos los parámetros necesarios (ID, servicio, característica, descripción y tipo de dispositivo). También se pasa el objeto completo recibido por MQTT para poder añadirle luego los resultados de la medición, la función de sendMqttData() y la de *changeCheckingNewOrders()*). Posteriormente se llama a la función *checkOrder()* que pide que se tome un tipo u otro de medida en función del tipo solicitado. Si no encuentra el tipo de dispositivo, lanza un mensaje de error mediante MQTT.

La clase *Weight* es la encargada de realizar todo el proceso de obtención y tratamientos de datos con la finalidad de obtener el peso del paciente. Al crear su instancia se le ha de enviar la ID (*MAC*) del dispositivo, el tipo de descripción buscada, el mensaje completo y la función de *sendMqttData*(). Posteriormente, se procede a la búsqueda de tramas mediante la función *getWeight()* que instanciará la clase *ScanBTLE* y llamará a *foundValueToScan()*. La función *foundValueToScan()* empezará a buscar entre todos los dispositivos BTLE que tiene a su alrededor, uno cuya dirección física y descripción coincidan con los proporcionados. En caso de no encontrar ninguna trama, se llamará a la función de *notificateError().* Si encuentra una trama, la enviará a la función *processData()* con la finalidad de que empiece a analizarla. Esta función será la encargada de llamar a *__getWeightFromManufacture()* que solicitará 2 cosas:

- El valor del peso, en formato decimal (float) haciendo una llamada a *__getWeightNumber().* Esta función, tal como se ha visto anteriormente, cogerá los bytes 4 y 5 en formato hexadecimal y los convertirá a números enteros. Posteriormente llamará a la función *putDecimalPoint()* con la finalidad de que añada el punto de la parte decimal en la sección necesaria.

- Las unidades de medida con que se ha tomado la medición. En este caso, tal como se ha visto anteriormente, es necesario analizar los bits 1 y 2 del tercer byte de la trama. Para ello, se requerirá, al igual que en la función *__getWeightDecimalPoint(),* de la función de utilidades *parseHexaToBinary()*.

Una vez encontrado el valor y la unidad, mediante la función *__checkIfUnitsIsCorrect()* comprueba si el resultado está en Kg, y en caso de no estarlo, hace un cambio de unidades. Finalmente, se llama a la función de *__sendResultData()* que será la encargada de modificar el objeto que se había recibido por MQTT para incluir la respuesta. El *topic* al cual se enviará la respuesta será la constante de *ORDERS_TOPIC_RESULT* más la *MAC* de la aplicación del paciente y su SIP.

4.4.2.2 Termómetro inteligente Kelvin

El termómetro inteligente Kelvin permite obtener la temperatura corporal en distintas partes del cuerpo (en la oreja o la cabeza) y en distintas unidades (en grados centígrados y grados kelvin). El nivel de precisión con el que entrega los resultados en la pantalla del dispositivo es de un único decimal, pero en la trama, la precisión es de dos decimales.

El dispositivo, cuanta además con alertas por fiebre, avisando con el número de vibraciones y duración el estado del paciente. En este caso, el termómetro ejerce una comunicación mediante notificaciones presentes en una determinada característica de un servicio del dispositivo.

4.4.2.2.1 Características principales

Tabla 5. Características Termómetro inteligente Kelvin

Características técnicas			**Características dispositivo**	
Rango de medición	32.0 – 43.0 °C (89.6 – 109.4 °F)		**Nombre del dispositivo**	Belter_TP
Precisión	± 0.3 °C		**Dirección MAC**	01:b6:ec:b9:01:d6
Resolución	0.1 °C (0.1 °F)		**Tipo de comunicación**	BLE mediante notificaciones
Entorno operativo	**Temperatura**	-10 – 40 °C	**UUID del servicio**	0000fff0-0000-1000-8000-00805f9b34fb
	Humedad	< 90%		
Alerta fiebre	**Temperatura normal**	Vibración corta	**UUID de la característica**	0000fff4-0000-1000-8000-00805f9b34fb

	Fiebre baja (37.2 – 37.4 °C)	Doble vibración y parpadeo en pantalla cada 1 segundo		
	Fiebre alta (> 37.5 °C)	Triple vibración y parpadeo en pantalla cada 0.5 segundos		

4.4.2.2.2 Trama de datos

Las tramas de datos enviadas por el dispositivo se corresponderían con el siguiente prototipo:

b'0220dd0aff020e5f0000000000005b'

Tabla 6. Trama de datos Termómetro inteligente Kelvin

Byte	Descripción				
0	SOP	02 = Encabezado			
1	Dispositivo	20 = Termómetro			
2	Com	DD	Número transmisión grupos	DD	
3	Tamaño	0A	Número transmisión sesión grupo	0A	
4	Tipo	FF = Resultado actual	AA = Último valor histórico	EE = Error	DD = Código información
5	COM	00 = Temperatura oreja °C 01 = Temperatura oreja °F 02 = Temperatura cabeza °C 03 = Temperatura cabeza °F		01 = Medida baja (L) 02 = Medida alta (H) 03 = Medida muy baja (EL) 04 = Medida muy alta (EL) 05 = Batería baja 06 = Error	FF = Unidad correcta FE = Historial completo FD = Número versión FC = Apagado dispositivo
6	Temp_H	Byte grande temperatura		FF	
7	Temp_L	Byte bajo temperatura		FF	
8	Año	Por defecto 00			
9	Mes	Por defecto 00			
10	Día	Por defecto 00			
11	Hora	Por defecto 00			
12	Minutos	Por defecto 00			
13	Segundos	Por defecto 00			
14	XOR	XOR (Byte 1 ~ Byte 13)			

Tal como se puede observar, desde el **byte 4** se puede obtener si es una medida recién tomada, si es una medida tomada anteriormente, si hay un error o si es un código de información. En este caso, como es FF, se puede concluir que la trama es correcta y actual. Si su valor fuese EE, lo correcto sería descartarla para evitar posibles confusiones. El valor del siguiente byte, el **byte 5**, nos permite saber en qué lugar se ha tomado la temperatura y con que unidad de medida. En este caso, como el valor es 02, se concluye que la medición se ha tomado en la frente y que ha sido medida en °C. Por último, el valor de la temperatura se encuentra en la siguiente trama. Este valor se encuentra en los **byte 6 y 7**. A la hora de convertirlo de hexadecimal a número entero, es importante que sea primero el **byte 6 y luego el 7**. En este caso, como el valor es 0e5f (hexadecimal) = 3679 (entero) = 36.7 (decimal).

4.4.2.2.3 Clase Temperature

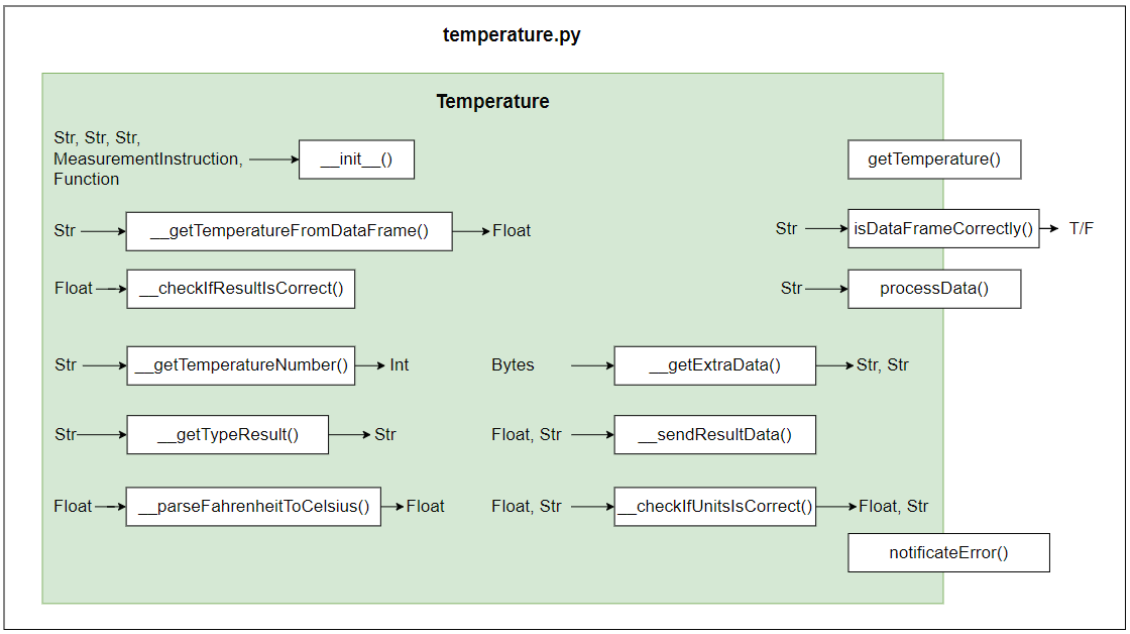


Figura 23. Diseño clase temperature.py

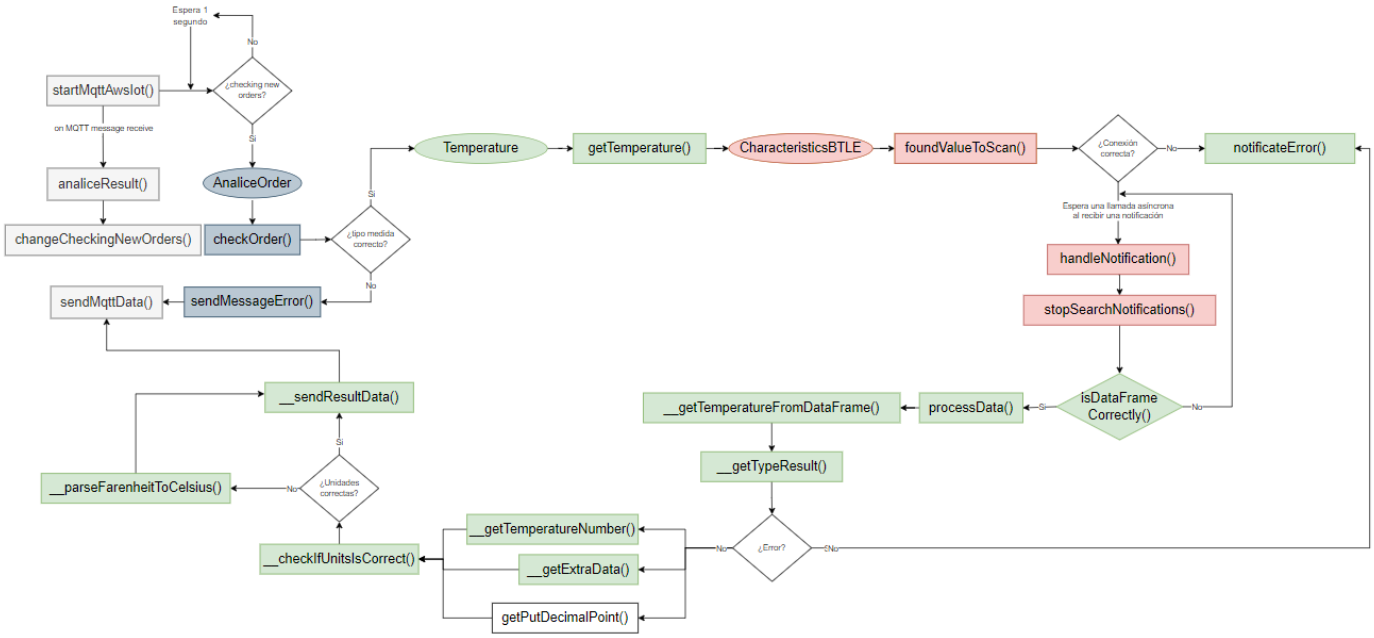


Figura 24. Diagrama de flujo para la toma de la temperatura

La clase *Temperature* es la encargada de realizar todo el proceso de obtención y tratamientos de datos con la finalidad de obtener la temperatura corporal del paciente. Tal como se observa en el diagrama de flujo, todas las clases contienen una estructura parecida, que incluye la parte del *main* (gris claro) y la de *AnaliceOrder* (gris oscuro).

Al crear su instancia (cuando se detecta que el mensaje recibido por MQTT es de tipo *Temperature*) se le ha de enviar la MAC del dispositivo, el identificador del servicio, la correspondiente característica, el propio mensaje al que añadirle el resultado de la medición y la función que permite enviar tramas MQTT. Posteriormente, se procede a la búsqueda de tramas mediante la función *getTemperature()* que instanciará la clase *CharacteristicsBTLE* y llamará a *foundValueToScan()*.

La función *foundValueToScan()* empezará tal como habíamos comentado previamente, a intentar establecer conexión con el dispositivo. En caso de no conseguirlo llamará a la función de *notificateError()* que será la encargada de enviar

un mensaje MQTT de vuelta con un error. En caso de que la conexión sea correcta, se creará su delegado y se esperará la recepción de notificaciones enviadas por la característica correspondiente al servicio indicado.

Cada vez que se reciba una nueva trama, la función *handleNotification*() de la clase *CharacteristicsDelegate* será la encargada de llamar a la función *isDataFrameCorrectly*(), que en este caso, comprobará si la trama tiene al menos 14 bytes. Si la condición se cumple, enviará la trama a la función *processData*() con la finalidad de que empiece a analizarla. Esta función será la encargada de llamar a *__getTemperatureFromDataFrame*() que solicitará en primer lugar, la comprobación del cuarto byte con la finalidad de saber si la trama ha sido recogida correctamente. Este proceso lo realizará la función *__getExtraData*(). Si el resultado es error, no se seguirá el proceso de tratamiento de datos y retornará *None*. En caso de que la respuesta no genere error, se pedirán 2 parámetros:

- El valor de la temperatura llamando a la función *__getTemperatureNumber()* que cogerá el sexto y séptimo byte y lo convertirá de hexadecimal a entero. Posteriormente, se llamará a la función *putDecimalPoint()* para convertirlo de tipo int a tipo float.

- La unidad de medida con que se ha tomado la medición y el lugar donde se ha tomado mediante la llamada a la función *_getExtraData().* Dicha función extraerá las unidades del byte número 4 y la ubicación del 5.

Finalmente, si los valores de temperatura y unidad son distintos de *None*, llamará a *__sendResultData()* y de lo contrario a *notificateError()*.

4.4.2.3 Tensiómetro de brazo inteligente BMP-200 Wireless

El tensiómetro de brazo inteligente BMP-200 Wireless permite obtener la presión sistólica, la presión diastólica y el pulso del paciente mediante el método oscilométrico (el dispositivo detecta el movimiento de la sangre a través de la arteria branquial, convirtiendo la presión sanguínea en una lectura digital). En este caso, el tensiómetro ejerce una comunicación mediante notificaciones presentes en una determinada característica de un servicio del dispositivo.

4.4.2.3.1 Características principales

Tabla 7. Características Tensiómetro de brazo inteligente BMP-200 Wireless

Características técnicas			Características dispositivo	
Dimensiones	152 x 55 x 36 mm		**Nombre del dispositivo**	eBlood-Pressure
Rango medición	**Presión**	0 – 280 mmHg	**Dirección MAC**	f4:5e:ab:ac:62:13
	Pulso	40 – 200 BMP		
Entorno operativo	**Temperatura**	10 – 40 °C	**Tipo de comunicación**	BLE mediante notificaciones
	Humedad	15 – 80 %		
Rango medición	**Presión**	0 – 280 mmHg	**UUID del servicio**	0000fff0-0000-1000-8000-00805f9b34fb
	Pulso	40 – 200 BMP		
			UUID de la característica	0000fff4-0000-1000-8000-00805f9b34fb

4.4.2.3.2 Trama de datos

Las tramas de datos enviadas por el dispositivo se corresponderían con el siguiente prototipo:

b'0c**008b**004a0000**0064**000000’

Tabla 8. Trama de datos Tensiómetro de brazo inteligente BMP-200 Wireless

Byte	Descripción
0	Que datos
1~2	Sistólica

3~4	Diastólica
5~6	Esquema
7~8	Pulso
9	ID usuario
10~11	Estatus de la medición
12	Los 2 dígitos del año (ej. 2013 = 13 = 0xOD)
13	Mes
14	Día
15	Hora
16	Minutos
17	Segundos

Tal como se puede observar, la presión sistólica está formada por los **bytes 1 y 2** (aunque cierto es, que como la presión sistólica máxima es de aproximadamente de 140 mmHg, nunca va a superar el valor 255 y por tanto, con solo coger el **byte 2** sería suficiente). En este caso sería: 008b (hexadecimal) = 139 (entero). La presión diastólica se puede obtener observando los dos siguientes bytes, los **bytes 3 y 4** (de igual forma que la presión diastólica, no puede ser mucho mayor de 90 mmHg, por tanto se puede utilizar únicamente el **byte 4**). En este caso sería: 004a (hexadecimal) = 74 (entero). Por último, se puede obtener las pulsaciones del corazón. Para ello, se utilizan los **bytes 7 y 8** (aunque de igual forma, como el máximo está en (226 – edad) en caso de hombres y (220 – edad) en caso de mujeres, se podría llegar a coger únicamente el **byte 8**).

4.4.2.3.3 Clase BloodPreassure

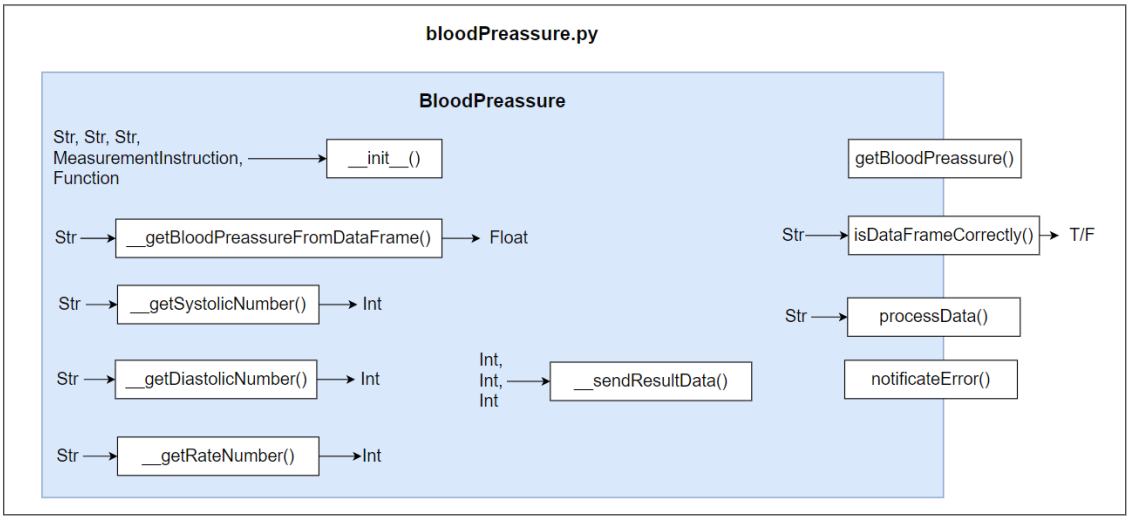


Figura 25. Diseño clase bloodPreassure.py

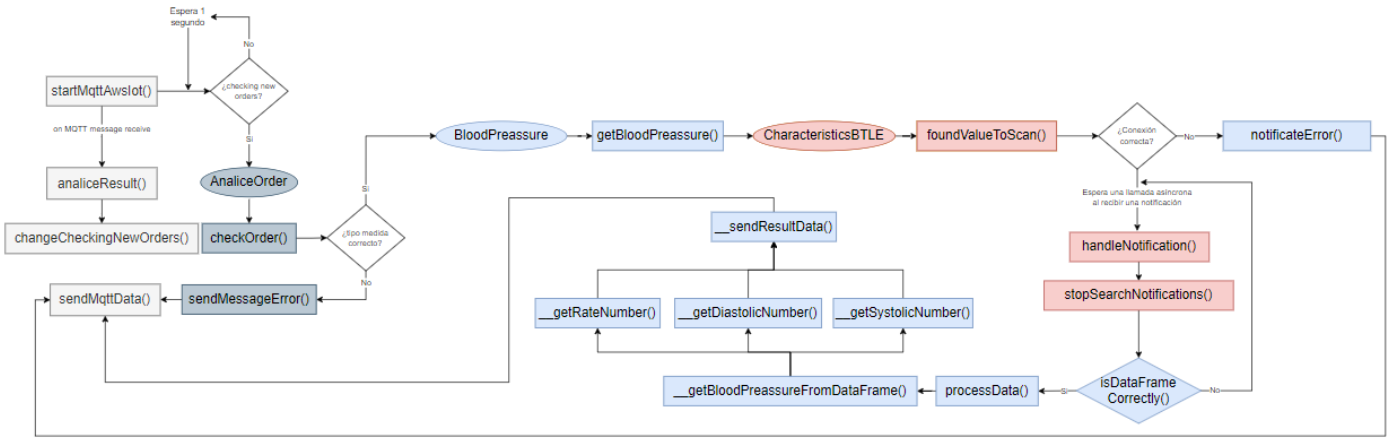


Figura 26. Diagrama de flujo para la toma de la tensión

La clase *BloodPreassure* es la encargada de realizar todo el proceso de obtención y procesamiento de datos con la finalidad de obtener la presión sistólica, la diastólica y las pulsaciones del paciente. Tanto la parte de inicializar AWS, como ver qué tipo de medida se quiere tomar y el proceso que se sigue en *CharacteristicsBTLE* (rojo) es el mismo que para todos los dispositivos que se comuniquen por notificaciones BLE, es decir, como por ejemplo en el caso de la temperatura.

Al crear su instancia (cuando se detecta que el mensaje recibido por MQTT es de tipo *BloodPreassure*) se le ha de enviar la MAC del dispositivo, el identificador del servicio, la correspondiente característica, el propio mensaje al que añadirle el resultado de la medición y la función que permite enviar tramas MQTT. Posteriormente, se procede a la búsqueda de tramas mediante la función *getBloodPreassure()* que instanciará la clase *CharacteristicsBTLE* y llamará a *foundValueToScan*(). La función *foundValueToScan*() empezará tal como habíamos comentado previamente, a intentar establecer conexión con el dispositivo. En caso de no conseguirlo llamará a la función *notificateError()*. En caso de que la conexión sea correcta, se creará su delegado y se esperará la recepción de notificaciones enviadas por la característica correspondiente al servicio indicador.

Cada vez que se reciba una nueva trama, la función *handleNotification*() de la clase *CharacteristicsDelegate* será la encargada de llamar a la función *isDataFrameCorrectly*(), que en este caso, comprobará si la trama tiene al menos 12 bytes. Si la condición se cumple, enviará la trama a la función *processData*() con la finalidad de que empiece a analizarla. Esta función será la encargada de llamar a *__getBloodPreassureFromDataFrame()* que solicitará 3 cosas:

- El valor de la presión sistólica llamando a la función *__getSystolicNumber()* que cogerá el segundo byte y lo convertirá de hexadecimal a entero.

- El valor de la presión diastólica llamando a la función __getDiastolicNumber() que cogerá el cuarto byte y lo convertirá de hexadecimal a entero.

- El valor de las pulsaciones por minuto llamando a la función __getRateNumber() que cogerá el octavo byte y lo convertirá de hexadecimal a entero.

Finalmente, se enviará el resultado de los datos mediante un mensaje MQTT al topic anteriormente mencionado.

4.4.2.4 Pulsioxímetro inteligente OL-750

El pulsioxímetro inteligente OL-750 permite obtener el porcentaje de saturación de oxígeno en sangre (SOP2) y las pulsaciones por minuto del paciente. En este caso, el pulsioxímetro ejerce una comunicación mediante notificaciones presentes en una determinada característica de un servicio del dispositivo.

4.4.2.4.1 Características principales

Tabla 9. Caracterísiticas Pulsioxímetro inteligente OL-750

Características técnicas			Características dispositivo	
Dimensiones	58 x 32 x 34 mm		**Nombre del dispositivo**	BLT_M70C
Rango medición	**SOP2**	70 a 99 %	**Dirección MAC**	64:69:4e:a3:b6:77
	Pulso	30 – 235 BPM		
Precisión	**SOP2**	70 – 80 % : ± 3 % | 80 – 99 % : ± 2 %	**Tipo de comunicación**	BLE mediante notificaciones
	Pulso	± 2 % o ± 2 LPM		
Resolución	**SOP2**	1 %	**UUID del servicio**	0000ffe0-0000-1000-8000-00805f9b34fb
	Pulso	1 BPM		
Periodo actualización	**SOP2**	< 13 s	**UUID de la característica**	0000ffe1-0000-1000-8000-00805f9b34fb
	Pulso	13 s		
Entorno operativo	**Temperatura**	5 – 40 °C		

	Presión atmosférica	700 – 1060 hPa		
	Humedad	15 % - 80 %		

4.4.2.4.2 Trama de datos

Las tramas de datos enviadas por el dispositivo se corresponderían con el siguiente prototipo:

b’0305080a0b0b0c0b0b0b0a0a090908a36258ff00’

En este caso, la documentación no era del todo clara y su información acerca de las tramas que contenían el resultado no eran semejantes a las obtenidas ni por un dispositivo móvil ni mediante la Raspberry Pi. Para ello, se decidió realizar un proceso de ingeniería inversa, donde sabiendo el resultado, hubiese bytes que contuviesen dichos valores. Una muestra del análisis realizado es:

Tabla 10. Trama de datos Pulsioxímetro inteligente OL-750

Valor oxígeno		Valor pulsaciones		Trama de datos
Hexa	**Int**	**Hexa**	**Int**	
69	99	5a	90	b'03030303030302020202020202032563 5aff00'
				b'aa4c41420007070707070707060606050504040 0'
				b'000000000000003c3a3a3b3c3c3a393633312da2'
				b'26231f1c1915120f0d0d121d2e3d4c55da91'
62	98	58	88	b’0305080a0b0b0c0b0b0b0a0a090908a36258ff00’
				b'aa0d430310160000000000000079’
61	97	55	85	b'00000000000000000000000000000096155 1100'
				b'00000000000002825211e1b181614110f0d0b0a'
				b'054c41510007070606070707070606050505 04'
61	97	55	85	b'01010101010101010101010000000096155 1100'

Se observa en primer lugar, que los bytes que contienen los parámetros buscados son la de los bytes 16 y 17. De forma aparente, tal como habíamos indicado anteriormente, no parece que haya una estructura clara que indique cuales son las tramas válidas y cuales no, pero si se pueden observar una serie de condiciones que permiten filtrar el listado de tramas y obtener solo aquellas que queremos. Las condiciones son:

- El número de bytes. Tal como se puede observar, las tramas que contienen los datos son siempre aquellas que tienen 19 bytes, ya que si tiene menos, es muy probable que siquiera llegue a los 17 bytes que se precisan para poder obtener el valor del oxígeno y el de las pulsaciones.

- La trama siempre acaba en 00, es decir, el **byte 19** siempre es 00. Esto lo podemos ver en la penúltima trama del último caso, donde parece cumplir la condición anteriormente mencionada pero no finaliza en 00.

- La trama nunca comienza en aa, es decir el **byte 0** nunca es igual a aa. Tal como se puede ver en la segunda trama del primer caso, se cumplen las dos condiciones anteriores pero empieza por las dos letras anteriormente mencionadas y no se detectan los valores buscados.

Por tanto, las condiciones que se aplicarán para filtrar la cantidad masiva de notificaciones sin aparente relación serán las anteriormente mencionadas; que durante largos periodos de testeo no parecen presentar ninguna excepción.

4.4.2.4.3 Clase PulseOximeter

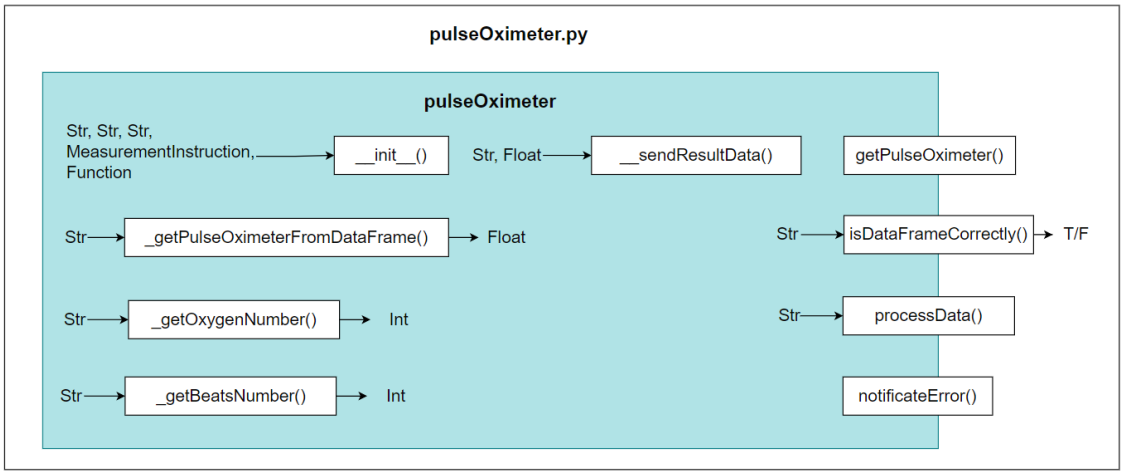


Figura 27. Diseño clase pulseOximeter.py.

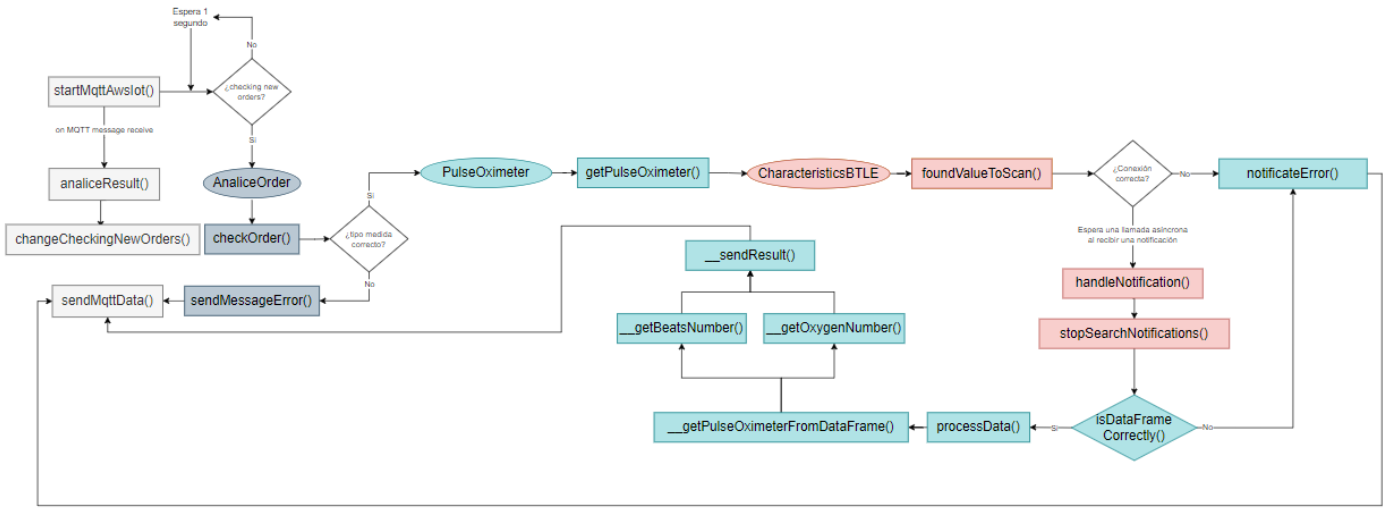


Figura 28. Diagrama de flujo para la toma del pulso y el oxígeno

La clase *PulseOximeter* es la encargada de realizar todo el proceso de obtención y procesamiento de datos con la finalidad de obtener la del porcentaje de saturación de oxígeno en sangre y las pulsaciones por minuto de los pacientes. Al igual que en la clase anterior, al hacer uso de la clase *CharacteristicsBTLE*, mantiene una estructura casi idéntica a la que utiliza la clase *BloodPreassure* o la clase *Temperature*.

Al crear su instancia se le ha de enviar la MAC del dispositivo, el identificador del servicio, la correspondiente característica, el propio mensaje al que añadirle el resultado de la medición y la función que permite enviar tramas MQTT. Posteriormente, se procede a la búsqueda de tramas mediante la función *getPulseOximeter()* que instanciará la clase *CharacteristicsBTLE* y llamará a *foundValueToScan()*. La función foundValueToScan() empezará tal como habíamos comentado previamente, a intentar establecer conexión con el dispositivo. En caso de no conseguirlo llamará a la función de *notificateError().* En caso de que la conexión sea correcta, se creará su delegado y se esperará la recepción de notificaciones enviadas por la característica correspondiente al servicio indicador. Cada vez que se reciba una nueva trama, la función *handleNotification()* de la clase *CharacteristicsDelegate* será la encargada de llamar a la función *isDataFrameCorrectly(),* que en este caso, comprobará todo lo mencionado en la sección anterior:

- La trama ha de ser mayor de 19 bytes.
- La trama no debe comenzar por el byte en hexadecimal aa.
- La trama debe acabar siempre con un byte a 0.

Si las condiciones se cumplen, enviará la trama a la función *processData()* con la finalidad de que empiece a analizarla. Esta función será la encargada de llamar a *__getPulseOximeterFromDataFrame()* que solicitará 2 cosas:

- El valor del oxígeno llamando a la función *__getOxygenNumber()* que cogerá el byte número 16 y lo convertirá de hexadecimal a entero.

- El valor de las pulsaciones por minuto llamando a la función *__getBeatsNumber()* que cogerá el byte diecisieteavo y lo convertirá de hexadecimal a entero.

Finalmente, mediante la función de *__sendResult()* se enviará el resultado de los valores recogidos por MQTT como en los casos anteriores.

4.4.3 Autoejecución software Raspberry Pi

Cuando la Raspberry Pi (parte del sistema que se conecta y obtiene los datos de los sensores) sufre un reinicio, el programa desarrollado en Python no se ejecuta por defecto. Para solucionar este problema, se hace uso del software PM2 [26]. Tal como mencionan en su página web, PM2 es un administrador de procesos de tipo daemons. Esto significa que permite de forma más simple generar y controlar programas persistentes en segundo plano. De esta manera, se consigue que al reiniciarse la Raspberry el programa se lance de forma automática. Para consultar el proceso de instalación ver la sección 3.2.2 del documento de Anexos.

4.5 Backend

La parte de backend consistirá en una API (Application Programming Interface) escrita en PHP aplicando el modelo MVC (Model View Controller). La elección de este modelo es que es un patrón de diseño que optimiza y estandariza el código. Tal como indica el acrónimo, consiste en una estructura de 3 capas: model, controller y view. Esta estructura se puede ver más fácilmente en la Figura 29. Estructura MVC para la API en PHP.

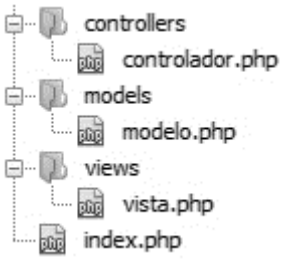


Figura 29. Estructura MVC para la API en PHP

4.5.1 Estructura base de datos

En la reunión con los médicos se expuso como está el sistema sanitario montado y los requisitos propios del sistema a desarrollar. Los requisitos fueron:

- Debían existir diferentes zonas básicas de salud, cada ZBS, podía tener distintos centros (hospitales, centros de salud, puestas sanitarias…).

- Cada paciente tiene un centro y un médico asignado por defecto. Esto no quita que otro médico (en caso de urgencia) no pueda visualizar los datos de pacientes de otras ZBS (se decide que un médico pueda visualizar N zonas básicas de salud).

- Un centro puede tener diversas salas donde se realizarán las consultas remotas, en cada sala, habrá un coordinador y un dispositivo (por ejemplo una Tablet) y N dispositivos para tomar las medidas.

- Otro requisito fue que no se podía acceder al sistema sin una cita previa, por ende, había que crear un sistema de citas (calendario).

En la Figura 30. Diagrama explicativo organización sistema principal se puede visualizar de forma clara un ejemplo de cómo está organizado el sistema.

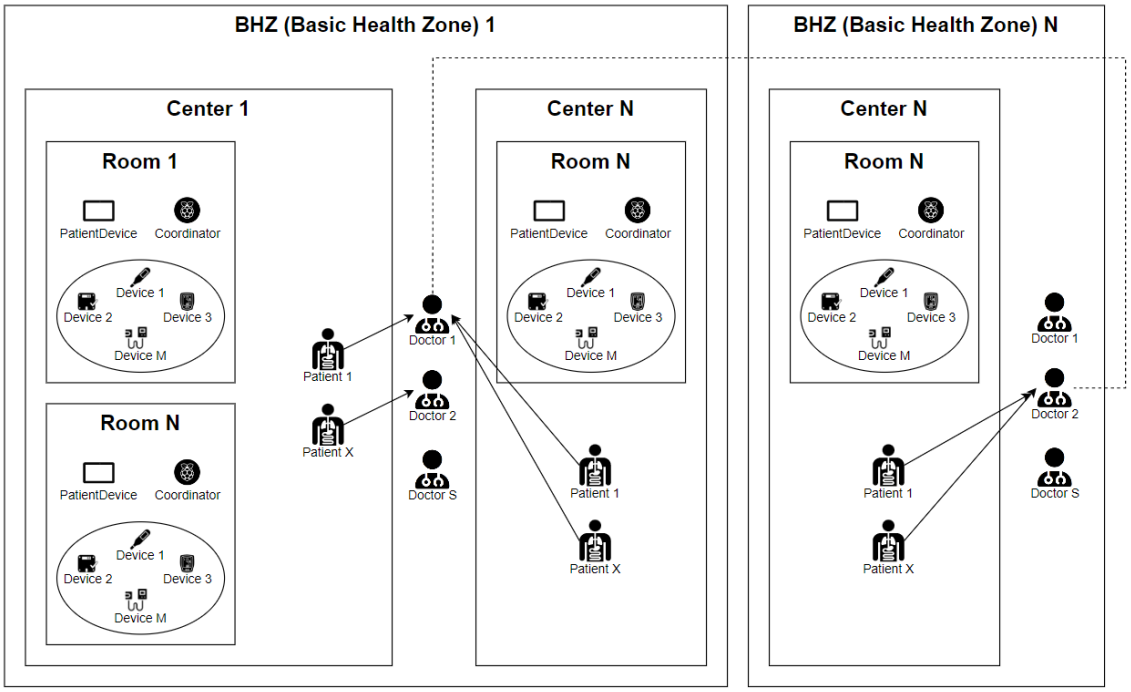


Figura 30. Diagrama explicativo organización sistema principal

Tal como se observa habrá N *BHZ* y cada *Basic Health Zone* tiene determinados *Centers*. Estos *Centers* pueden ser puestas sanitarias, hospitales, centros de salud… Cada *Center* tiene distintas *Rooms* donde los pacientes acuden a realizar la consultar con el médico. Cada *Room* dispondrá de una Tablet en la cual el paciente realizará la consulta con el médico; también dispondrá de un coordinador que será el que se encargará de obtener los datos de los M *Devices* disponibles en la sala (en principio son 4, pero este número puede variar). Adicionalmente, cada *Center* tiene X *Patients* y cada *Patient* tiene un *Doctor* predeterminado. Hay que destacar que cada Doctor puede estar asignado a varias *BHZ* para poder visualizar todos los pacientes de cada *BHZ*. La estructura final de la base de datos relacional es la de la Figura 31.

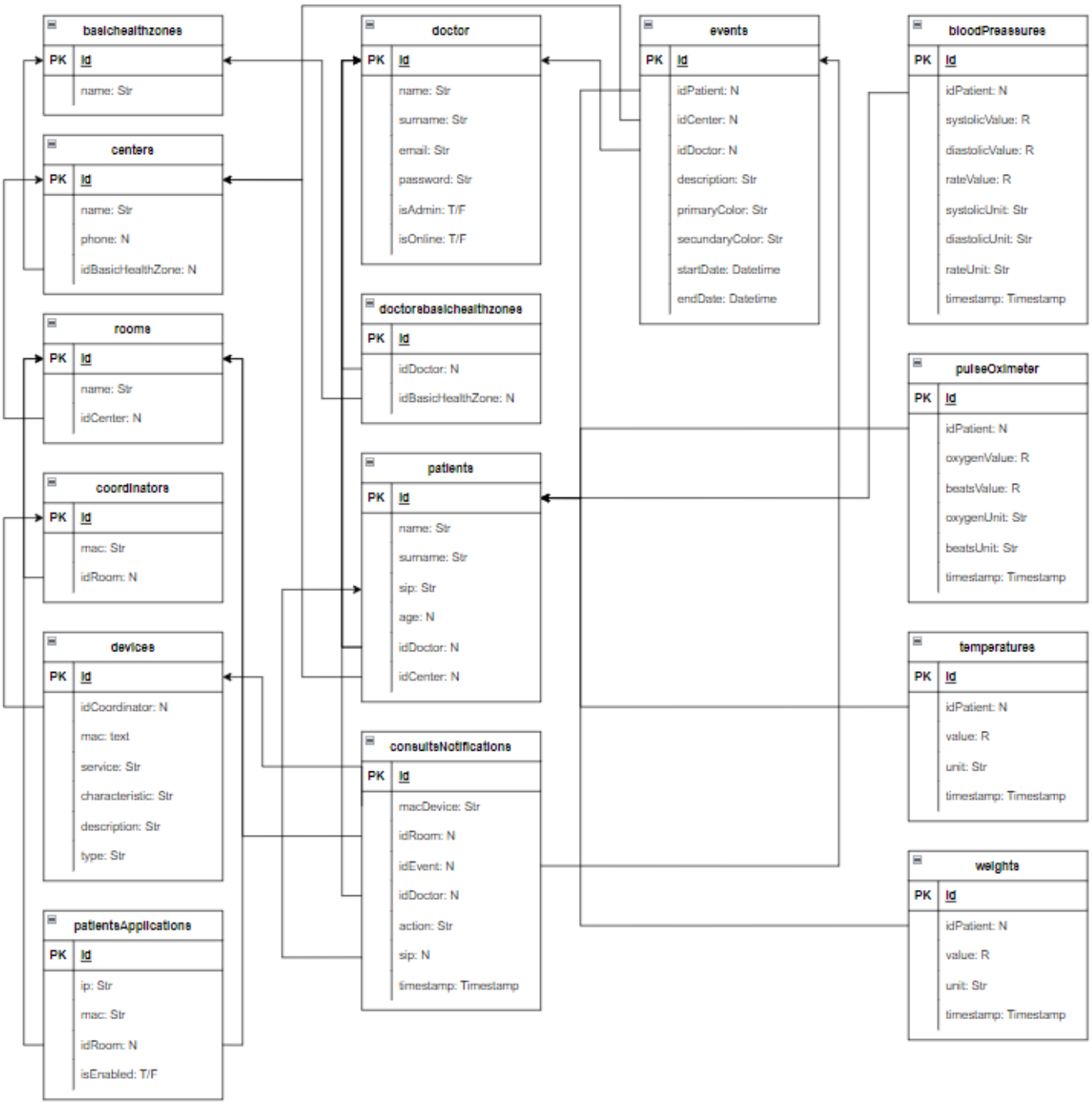


Figura 31. Diseño base de datos

4.6 Frontend

La parte de frontend del sistema constará de 2 aplicaciones separadas: la aplicación web médicos y la PWA (Progressive Web Apps) de las salas de los centros para pacientes. Los motivos por los cuales se ha decidido realizar dos aplicaciones separadas son:

- La aplicación web para los médicos es mucho más pesada ya que contiene mucho más código al cual el paciente no va a necesitar tener acceso.

- La aplicación para los pacientes es una PWA, es decir, que se podrá instalar de forma nativa en una Tablet ya que estos dispositivos son más fáciles de utilizar por gente mayor que un ordenador con ratón y teclado.

Ambas aplicaciones están desarrolladas mediante el framework de Angular [27]. El motivo de la elección de Angular no era solo que permitía crear aplicaciones para todo tipo de dispositivos, sino además, generar un código más modularizado y simple de entender entre otras cosas. Al utilizar Angular, los lenguajes utilizados son principalmente: Typescript (JavaScript fuertemente tipado), HTML y SCSS (el cual nos permite realizar estilos en cascada, por lo cual, el código queda más ordenado que con CSS).

Para el diseño de la aplicación, se ha utilizado PrimeNg [28], una colección de componentes de interfaz de usuario, que nos proporciona unas bases sobre las cuales se realizarán posteriores modificaciones. Uno de los puntos más fuertes que tiene es su sistema PrimeFlex [29], que permite hacer que las aplicaciones sean responsives gracias, entre otros, a su sistema de columnas. También se han hecho uso de las siguientes librerías externas:

- **Angular 12.0+ calendar** [30]**:** esta librería ofrece un gran abanico de posibilidades para crear tu propio diccionario. De las librerías gratuitas encontradas, era la mejor con amplia diferencia.

- **Font Awesome** [31]**:** aunque es cierto, que PrimeNg tiene una librería de iconos gratuita, en determinados casos como el de los dispositivos médicos, queda corta. Con esta API, se complementan dichas carencias.

- **VideoSdk** [32]: uno de los requisitos indispensables para los médicos, era que pudiesen conectarse mediante videollamada con los pacientes. En términos generales, las API de videollamadas suponen un alto coste.
  Esta API, en comparación con las otras, ofrece un servicio muy económico y múltiples opciones que se explicarán en la siguiente sección.

4.6.1 VideoSdk

La API de VideoSdk, permite una total personalización de sus funcionalidades e interfaz. Algunas de las múltiples características que contiene son:

- Desarrollo para plataformas web (Prebuilt, Javascript, React (aunque sirven también para Angular, sobretodo la primera y la tercera)) y móvil (React Native, FLutter, Android e iOS).

- Salas ilimitadas y con *layouts* dinámicos (se ajustan en función del número de participantes).

- Ecosistema de APIS como AWS S3 para almacenar las videollamadas grabadas, Google Calendar o Wordpress.

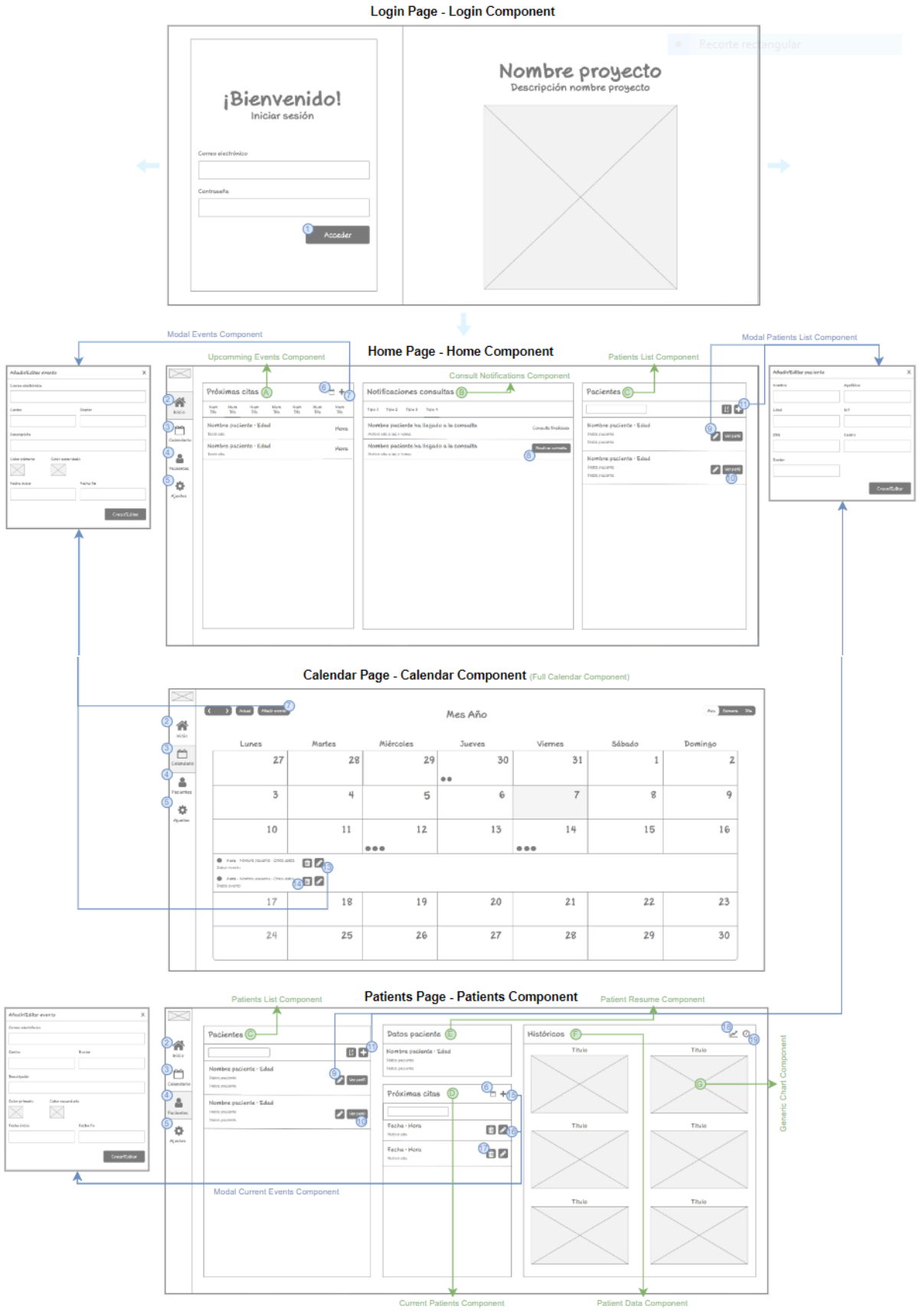
- Chat, transferencia de archivos y pizarra.

- Posibilidad de grabar las llamadas.

Los precios varían en función de las características a utilizar. En todas las variaciones, se incluyen 10.000 minutos al mes gratuitos. En este caso, se va a realizar una implementación en SD, ya que no se necesita una resolución excesiva y en entornos rurales, tal como se había visto, la conexión siempre es peor, por tanto, cuanto menos consumo de internet mejor. Tampoco se realiza la implementación de *streaming* ni la de grabación de pantalla ya que no es necesario. El precio, por ende, se basaría en 0,002$ más 0,0006$ por minuto extra (es decir, a parte de esos 10.000 gratuitos) de participante de videollamada en calidad SD con audio.

4.6.2 Aplicación médicos

En la Figura 32 se pueden visualizar los principales componentes utilizados en las distintas secciones de la página y el flujo de estas mediante los *wireframes* finales.


Login Page - Login Component
¡Bienvenido!
Iniciar sesión
Correo electrónico
Contraseña
Acceder
Nombre proyecto
Descripción nombre proyecto
Modal Events Component
Upcomming Events Component
Home Page - Home Component
Consult Notifications Component
Patients List Component
Modal Patients List Component
Próximas citas
Notificaciones consultas
Pacientes
Calendar Page - Calendar Component (Full Calendar Component)
Mes Año
Lunes
Martes
Miércoles
Jueves
Viernes
Sábado
Domingo
Patients List Component
Patients Page - Patients Component
Patient Resume Component
Datos paciente
Históricos
Modal Current Events Component
Generic Chart Component
Current Patients Component
Patient Data Component

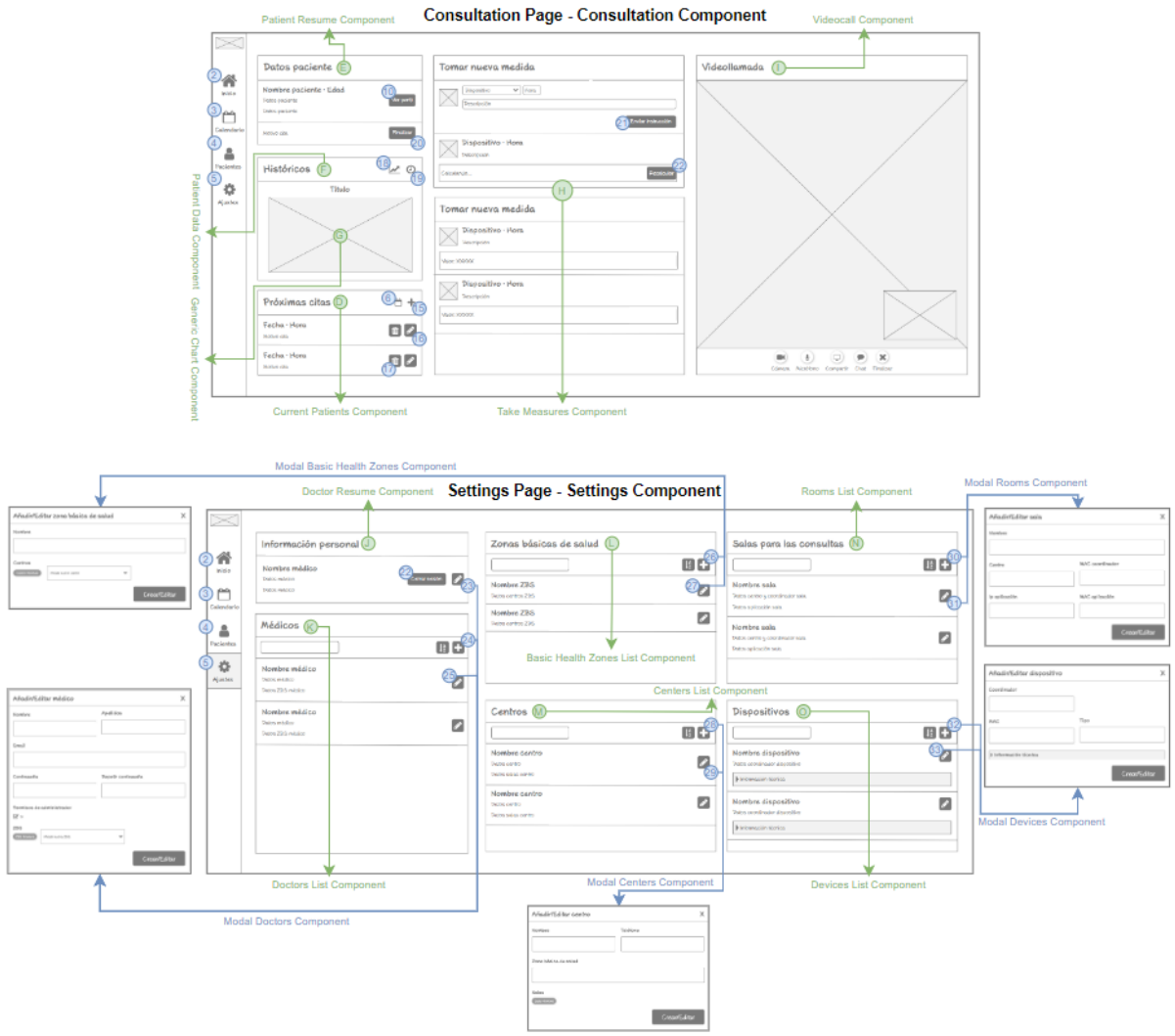


Figura 32. Diagrama conexiones páginas y componentes (app médico)

La primera página que aparece en la web es el *login*, al introducir, el correo electrónico y la contraseña de manera correcta y pulsar en el botón de *Acceder* ① se redirigirá al usuario a la página de inicio. Por el contrario, se mostrará un mensaje de error.

El menú lateral tiene distintas secciones, redirigiendo cada una de ellas a una página distinta. La ② al inicio (Home), la ③ al calendario (Calendar), la ④ a la sección de pacientes (Patients) y la ⑤ a los ajustes (Doctors). Este menú se repite en cada una de las páginas, quedando remarcada siempre la opción en la que se encuentra el usuario. El botón ⑥, al igual que el ③, llevará al médico al calendario. El botón ①⓪, al igual que el ④, llevará al médico a la sección de pacientes, pero esta vez, con el paciente seleccionado por defecto.

Cada uno de los paneles que se puede ver en la página de inicio, tienen sus propios componentes Ⓐ, Ⓑ y Ⓒ. El componente Ⓒ será reutilizado para la sección de pacientes, tal como se puede visualizar en la Figura 32.

Al hacer click en el botón de crear un evento ⑦, se abrirá un modal para poder crear uno (con la fecha seleccionada por defecto como el día actual en el caso del calendario y en el panel de próximas citas como el día seleccionado en el panel superior). Al crearse, se actualiza el panel o calendario correspondiente de manera dinámica. Los botones ⑨ y ①① realizan la misma acción, abrir un modal que permita visualizar editar o crear un paciente. De igual forma que en el caso anterior, al editar o añadir un nuevo paciente, la lista de pacientes se actualiza de manera dinámica.

El panel de notificaciones consulta Ⓑ, es uno de los encargados de quedar a la escucha de que mediante AWS IoT (componente Aws Mqtt) le lleguen las notificaciones desde los dispositivos de las salas médicas donde acudirán los pacientes (este solo lee y añade, no almacena en BBDD, el encargado de leer y almacenar dichas notificaciones es el App Component, ya que de no hacerlo así, aunque el médico tuviese web abierta, si estuviese en otra sección no las

recibiría). Al hacer click en el botón ⑧, se envía una notificación al dispositivo de la consulta que se ha de iniciar la llamada para determinado paciente. En el caso de la página del calendario, seleccionarse un día concreto, aparecen sus eventos (en el caso semanal y diario aparecen directamente, sin necesidad de pulsar). Sobre cada evento se puede visualizar un botón para eliminarlo ⑭, que hará que aparezca un modal para confirmar la eliminación y un botón para editarlo ⑬, que abrirá el modal para poder editarlo.

En la página de pacientes, se muestra el listado de pacientes Ⓒ. Al tener seleccionado uno, se muestra su información personal Ⓔ, sus próximos eventos Ⓓ y sus datos sobre las mediciones tomadas Ⓕ. En este caso, el panel Ⓓ, abre un modal muy semejante al abierto en la página de inicio y de calendario, pero se decidió hacer uno separado ya que no se requerían muchos de los parámetros que en el resto de los casos sí. Los botones 15, 16 y 17 realizan funciones muy semejantes a los ⑦, ⑬ y ⑭, pero esta vez para los eventos concretos de un paciente, no para todos los eventos de un médico (al hacer cualquier modificación, el panel se actualiza de manera dinámica). Hay que destacar que el panel de mediciones puede presentar n gráficos que se pueden filtrar gracias a los botones ⑱ y ⑲, cada uno de ellos se genera transfiriendo determinados datos al componente Ⓖ, que en función de los datos genera un tipo de grafico u otro.

La página de consultas, solo se abre de manera automática cuando se inicia una llamada (si no es el caso, no deja acceder). En ella podemos observar los paneles Ⓔ, Ⓕ y Ⓓ reutilizados de la sección de pacientes. El componente H está formado por 2 paneles, que permiten enviar ordenes al coordinador y esperar una determinada respuesta. El componente de videocall Ⓘ, es uno de los más importantes ya que era una funcionalidad necesaria que solicitaban los médicos, esta crea la videollamada mediante la API.

Por último, la página de ajustes permite al médico poder hacer una completa gestión del sistema, siempre y cuando tenga permisos de administrador. Si no los tiene, solo podrá visualizar el panel Ⓙ (si no es admin, no podrá auto otorgarse permisos). Desde esta página se podrán visualizar, crear y editar: médicos (panel Ⓚ, crear ㉔ y editar ㉕), ZBS (Ⓛ, crear ㉖ y editar ㉗), centros (Ⓜ, crear ㉘ y editar ㉙), salas (Ⓝ, crear ㉚ y editar ㉛) y dispositivos (Ⓞ, crear ㉜ y editar ㉝). Al igual que en el resto de las páginas, al modificar alguno de los datos, se actualizan todos los componentes que usan dichos datos de manera dinámica.

4.6.3 Aplicación pacientes

Al igual que en el caso anterior, mediante la Figura 33 se puede visualizar el flujo entre las páginas de la aplicación así como los componentes que la forman.

En este caso, el sistema de login verifica distintas condiciones para confirmar una cita (se pueden ver en la Figura 34) tras insertar el SIP y pulsar el botón ①. Cuando un médico inicia la videollamada, la aplicación del paciente se redirige de manera automática a la página de consultas, la cual tiene tres paneles, divididos en dos componentes, uno para la recogida de medidas Ⓐ y uno para la videollamada Ⓟ. Hay que destacar que ambas páginas contienen el componente de AWS MQTTque es indispensable para realizar toda la comunicación.

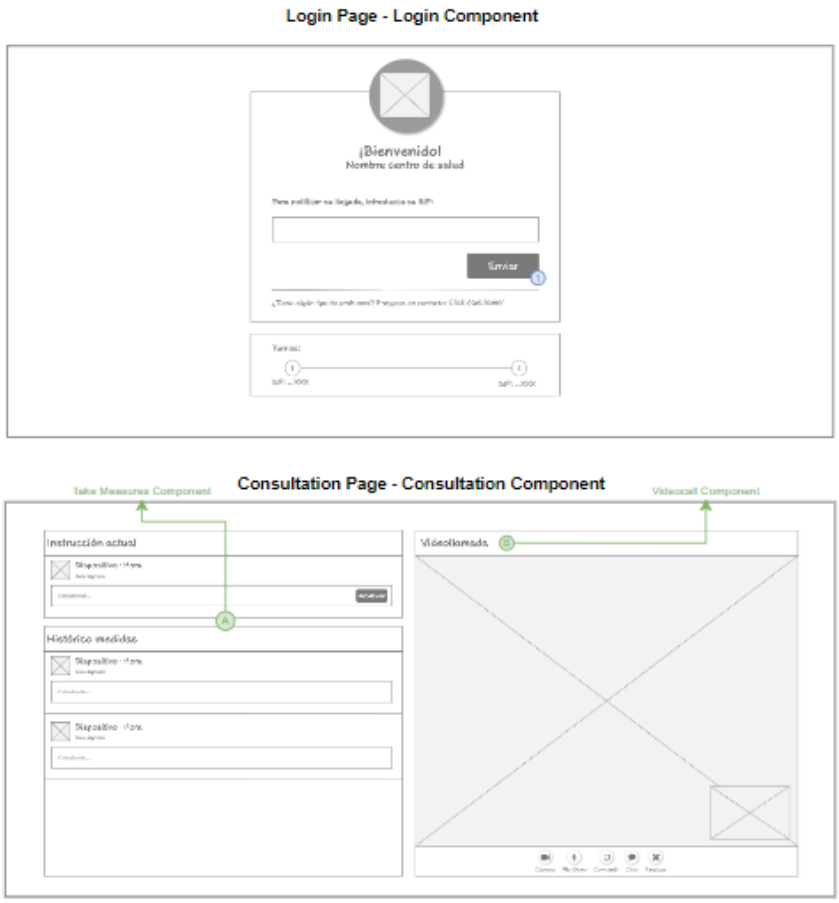


Figura 33. Diagrama conexiones páginas y componentes (app paciente)

4.7 Comunicación entre dispositivos

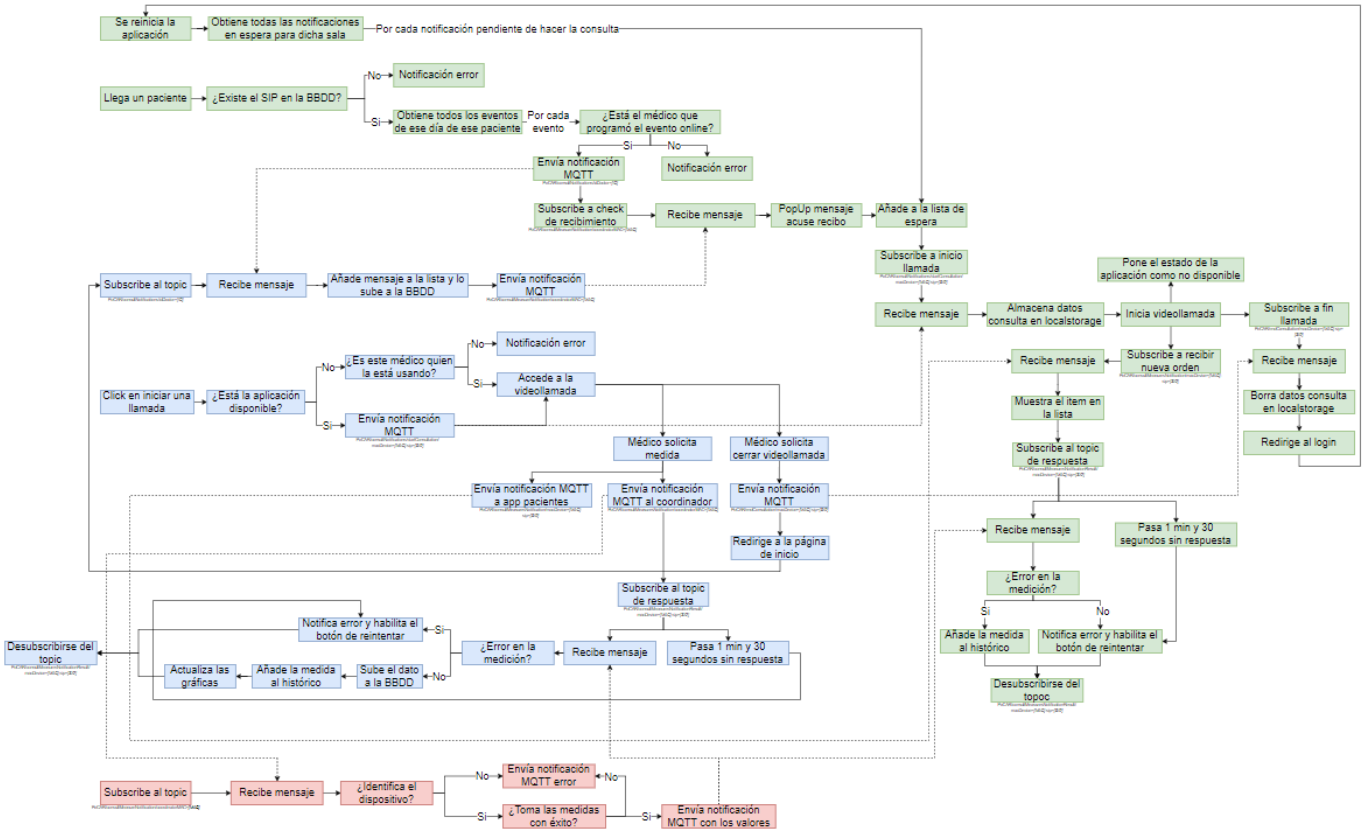


Figura 34. Diagrama de flujo comunicación entre el coordinador y las apps del médico y del paciente

En la Figura 34 que se puede observar la comunicación para tomar una medida entre las 2 aplicaciones (la que usa el médico y la que usa el paciente) y el coordinador (Raspberry Pi). En este caso, la parte en rojo se corresponde al coordinador, mientras que la azul pertenece a la aplicación del médico y por ende la verde a la del paciente.

En él, se observan comprobaciones de todo tipo para asegurar una correcta comunicación y corrección ante posibles errores. Toda la comunicación se realiza mediante MQTT, más concretamente, tal como se había comentado anteriormente, mediante AWS IoT que proporciona un mayor nivel de seguridad en el envío y recepción de mensajes.

5 Validación

El sistema cuenta con distintas formas de testeo con la finalidad de corregir la mayor cantidad de errores posibles antes del uso del producto.

El primer tipo de testeo es el manual, realizado durante todo el proceso de desarrollo, en el cual se han intentado forzar los errores con la finalidad de crear un sistema lo más robusto posible. Por ejemplo, los formularios, no dejan ser enviados hasta que no están completos todos los campos que se han puesto como obligatorios. Otro ejemplo podría ser, todo el proceso de la toma de mediciones, donde en la Figura 34 se observa que se comprueba en todo momento múltiples condiciones que hacen que si no se recibe respuesta no se quede pensando en bucle o que si algún paciente ya está en videollamada en determinada sala, no se pueda inicializar otra.

El segundo tipo de testeo realizado es con el software de Selenium [14], donde se prueban las principales acciones de la aplicación (sobre todo, aquellas que tienen contacto con la base de datos). Los resultados de este testing son positivos puesto que se han realizado un total de 26 pruebas, en las cuales, en 3 se han detectado errores que han sido solucionados. En la Tabla 11 se puede ver un resumen de las pruebas realizadas mediante Selenium. Para más información consultar la sección 2 del documento de Anexos.

Tabla 11. Resumen tests Selenium

	Nº de tests	Nº de errores detectados	Nº de errores solucionados
Login	2	0	0
Eventos	4	0	0
Pacientes	2	2	2
Médicos	5	1	1
ZBS	2	0	0
Centros	2	0	0
Salas	5	0	0
Dispositivos	4	0	0

Queda pendiente realizar la validación en un hospital de Castellón, donde se harán las pertinentes pruebas con pacientes reales, con la finalidad probar el sistema con la mayor profundidad posible y poder corregir los errores pertinentes para poder hacer un uso real de la misma, así como validar la usabilidad de las aplicaciones cliente.

6 Conclusiones y trabajo futuro

El proyecto fue diseñado con la finalidad de cubrir una necesidad: evitar los desplazamientos a las personas de las distintas zonas rurales a sus respectivos hospitales para realizar controles rutinarios. Las funcionalidades básicas que se debían proporcionar al proyecto han sido alcanzadas con éxito. Estas funcionalidades son:

- Poder tomar mediciones de distintos dispositivos (termómetro, báscula, pulsioxímetro y tensiómetro) de manera remota. Estas mediciones quedan almacenadas para que puedan ser visualizadas en cualquier momento.

- Tener un sistema de citas para poder administrar de manera correcta las distintas consultas.

- Disponer de un sistema de video consulta remoto médico-paciente de forma eficiente (recordar que en las zonas más rurales la conectividad puede ser baja).

- Proporcionar un alto nivel de robustez y seguridad.

- Ofrecer una solución fácil de administrar (el panel de administrador de la sección web permite modificar cualquiera de las necesidades).

De cara al futuro, tal como se ha indicado anteriormente, el trabajo indispensable sería:

- Probar en un entorno real diferentes casuísticas para asegurar el correcto funcionamiento del sistema. Esto implicaría la implementación de uno o diversos sistemas que deberían estar un determinado periodo en funcionamiento con la finalidad de recoger datos que puedan ayudar a mejor el sistema.

- Modificar la API de VideoSDK para que esté íntegramente en español y con la gama cromática y fuente usadas por defecto en la aplicación.

Adicionalmente a ese trabajo, podría ser necesario o interesante:

- Implementar nuevos dispositivos de medición. Esta implementación no debería suponer una gran cantidad de tiempo (siempre y en cuanto se comuniquen por BLE) ya que como se ha visto anteriormente el sistema está muy modularizado y por ende se facilita la incorporación de nuevos dispositivos.

- Desarrollar un sistema OTA para el coordinador que permitiese enviar actualizaciones de forma segura y rápida.

Por último, algunas de las competencias transversales mejoradas durante el desarrollo del proyecto son:

- Comprensión e integración.

- Conocimiento de problemas contemporáneos.

- Aplicación y pensamiento práctico.

- Análisis y resolución de problemas.

- Planificación y gestión del tiempo.

7 Referencias bibliográficas

[1] M. Alloza, V. González Díez, E. Moral Benito, and P. Tello Casas, "El acceso a servicios en la España rural," Sep. 2021. https://www.bde.es/f/webbde/SES/Secciones/Publicaciones/PublicacionesSeriadas/DocumentosOcasionales/21/Fich/do2122.pdf (accessed Jun. 26, 2022).

[2] A. Maqueda, "Las zonas rurales de España tienen peor acceso a los servicios básicos que las de otros países europeos," Sep. 08, 2021. https://elpais.com/economia/2021-09-08/las-zonas-rurales-en-espana-tienen-peor-acceso-a-servicios-que-las-del-resto-de-la-ue.html (accessed Jul. 02, 2022).

[3] Ministerio de agricultura pesca y alimentación, "La población de las áreas rurales en España supera los 7,5 millones de personas," Dec. 27, 2021. https://www.mapa.gob.es/es/prensa/ultimas-noticias/la-poblaci%C3%B3n-de-las-%C3%A1reas-rurales-en-espa%C3%B1a-supera-los-75-millones-de-personas-/tcm:36-583990 (accessed Jun. 26, 2022).

[4] Segovia Audaz, "La edad media en el medio rural es de 55 años, nueve más que en las capitales," Sep. 07, 2022. https://segoviaudaz.es/la-edad-media-en-el-medio-rural-es-de-55-anos-nueve-mas-que-en-las-capitales/#:~:text=Informaci%C3%B3n-,La%20edad%20media%20en%20el%20medio%20rural%20es%20de%2055,m%C3%A1s%20que%20en%20las%20capitales (accessed Jun. 26, 2022).

[5] Antena 3 Noticias, "¿Hay un límite de edad para dejar de conducir?," Aug. 21, 2021. https://www.antena3.com/noticias/sociedad/hay-limite-edad-dejar-conducir_20210820611f78cf5690c10001737a7c.html (accessed Jun. 26, 2022).

[6] "LoRa Alliance." https://lora-alliance.org/ (accessed Jun. 26, 2022).

[7] Intel, "Tecnología de telemedicina con IA e IoT." https://www.intel.es/content/www/es/es/healthcare-it/telemedicine.html (accessed Jun. 26, 2022).

[8] eSMARTCITY, "BWell: sistema integral de monitorización de pacientes a distancia," Dec. 02, 2019. https://www.esmartcity.es/comunicaciones/comunicacion-bwell-sistema-integral-monitorizacion-pacientes-distancia (accessed Jun. 26, 2022).

[9] Google Cloud, "Que es un data lake?" https://cloud.google.com/learn/what-is-a-data-lake?hl=es-419#:~:text=Un%20data%20lake%20es%20un,ignorando%20los%20l%C3%ADmites%20de%20tama%C3%B1o. (accessed Jun. 26, 2022).

[10] Geriatricarea, "VitalOn, combina en una única plataforma la monitorización remota de pacientes," Jun. 2021. https://www.geriatricarea.com/2021/06/07/vitalon-combina-en-una-unica-plataforma-la-monitorizacion-remota-de-pacientes/ (accessed Jun. 26, 2022).

[11] EssenceSmartcare, "VitalOn." https://www.essencesmartcare.com/solutions/vitalon/ (accessed Jun. 26, 2022).

[12] Neebo, "Neebo Monitor." https://neebomonitor.com/ (accessed Jun. 26, 2022).

[13] "Axure." https://www.axure.com/ (accessed Jul. 03, 2022).

[14] "Selenium." https://www.selenium.dev/ (accessed Jun. 26, 2022).

[15] L. Llamas, "¿Qué es MQTT? Su importancia como protocolo IoT," Apr. 17, 2019. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/ (accessed Jun. 26, 2022).

[16] I. Rihawi, "Explorando la nube de AWS para el universo IoT," Jul. 14, 2020. https://empresas.blogthinkbig.com/explorando-la-nube-de-aws-para-el-universo-iot/ (accessed Jun. 26, 2022).

[17] ILIMIT, “Ventajas de migrar a AWS,” Apr. 19, 2021. https://www.ilimit.com/blog/ventajas-migrar-aws/ (accessed Jun. 26, 2022).

[18] “Precios de AWS IoT Core”, Accessed: Jun. 26, 2022. [Online]. Available: https://aws.amazon.com/es/iot-core/pricing/?nc=sn&loc=4

[19] IEEE SA, “IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN),” Jun. 14, 2005. https://standards.ieee.org/ieee/802.15.1/3513/ (accessed Jun. 26, 2022).

[20] Barcelona Health Hub, “Lifevit.” https://lifevit.es/ (accessed Jun. 26, 2022).

[21] Raspberry Pi, “Raspberry Pi 3 Model B.” https://www.raspberrypi.com/products/raspberry-pi-3-model-b/ (accessed Jun. 26, 2022).

[22] Nordic Semiconductor ASA, “nRF Connect for Mobile.” https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=es_419&gl=US (accessed Jun. 26, 2022).

[23] J. Nieminen, T. Savolainen, Patil. Basavaraj, M. Isomäki, Z. Shelby, and C. Gomez, “RFC 7668: IPv6 over BLUETOOTH(R) Low Energy,” Oct. 2015. https://www.researchgate.net/publication/256295508_RFC_7668_IPv6_over_BLUETOOTHR_Low_Energy (accessed Jun. 26, 2022).

[24] I. Harvey, “bluepy,” Oct. 19, 2013. https://github.com/IanHarvey/bluepy (accessed Jun. 26, 2022).

[25] “Try the AWS IoT quick connect.” https://docs.aws.amazon.com/iot/latest/developerguide/iot-quick-start.html (accessed Jun. 26, 2022).

[26] “PM2.” https://pm2.keymetrics.io/ (accessed Jun. 28, 2022).

[27] “Angular.” https://angular.io/ (accessed Jun. 29, 2022).

[28] “PrimeNg.” https://www.primefaces.org/primeng/ (accessed Jun. 29, 2022).

[29] “PrimeFlex.” https://www.primefaces.org/primeflex/ (accessed Jun. 29, 2022).

[30] M. Lewis, “Angular 12.0+ calendar,” Apr. 26, 2016. https://openbase.com/js/angular-calendar/documentation (accessed Jun. 29, 2022).

[31] R. Madole and G. Tagliatela, “Font Awesome.” https://fontawesome.com/start (accessed Jun. 29, 2022).

[32] “VideoSdk.” https://videosdk.live/ (accessed Jun. 29, 2022).