ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE LA SALUD: INGENIERÍA BIOMÉDICA.

REGISTRO REMOTO DE DATOS BIOMÉDICOS "BAJO DEMANDA"

REMOTE REGISTRY OF BIOMEDICAL DATA ON DEMAND

Realizado por
María Sanzo Díaz.

Tutorizado por
Rafael de Jesús Navas González.
Manuel Jesús Martín Vázquez.

Departamento
Electrónica.

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Septiembre, 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: Este trabajo fin de grado consiste en especificar, diseñar e implementar un sistema básico para la adquisición y el registro remoto de datos biomédicos de pacientes monitorizados in situ, y localizados en diferentes ubicaciones, a medida que esta información es solicitada desde un operador o gestor centralizado. En este sistema, la iniciativa de la comunicación la lleva el operador del sistema centralizado, que, cuando estima oportuno, solicita al sistema de monitorización la realización de una medida y, por tanto, el envío de los datos solicitados.

Así, las tareas de este trabajo comprenden el estudio de las necesidades y requerimientos del sistema, la elección de los elementos hardware, y el desarrollo de las aplicaciones software apropiadas para su implementación. Estas tareas cubren aspectos relativos a la concepción e implantación de un sistema básico de e-Salud. El sistema desarrollado contempla aspectos básicos como son: el registro de un conjunto de medidas biomédicas (temperatura corporal, pulsioximetría, electrocardiografía (ECG), etc,) en una pequeña comunidad de pacientes, gestionada desde un servidor remoto por un operador. El funcionamiento global del sistema podría resumirse de la siguiente manera: El operador selecciona un paciente de los que han sido dados de alta en el sistema, y solicita un registro biomédico, por ejemplo su temperatura corporal actual. La orden es enviada al sistema de monitorización del paciente en su localización. El resultado de la medida es remitido de vuelta al servidor. Éste recibe los valores obtenidos y los almacena, de modo que puedan ser recuperados y analizados posteriormente. Además, desde el puesto del operador es posible realizar una gestión de pacientes y un análisis básico de los datos.

Palabras clave: Bajo demanda, sensores biomédicos, Arduino, e-salud, red inalámbrica, servidor web, monitorización de variables fisiológicas.

Abstract: The thesis here presented aims to specify, design and implement a basic system for the remote acquisition and record of biomedical data of patients with in situ monitoring situated at several locations. This information should be accessible as requested by a centralized agent. Therefore, is the agent the responsible for initiating the communication and demanding to the monitoring system the record and delivery of the data of a specific measurement.

This project comprehends the study of the needs and requirements of the desired system, the selection of the hardware elements, and the development of all software applications needed for the implementation of the system. These tasks involve aspects related to the conception and implementation of a basic e-Health system.

The developed system includes basic aspects such as: the record of a combination of biomedical measurements (body temperature, pulse oximetry, electrocardiogram (ECG)) in a reduced number of patients, controlled by an agent from a remote server. The global operation of the system can be summarized as follows: the agent selects a

name from the list of patients being monitored by the system and requests the record of some biomedical data, e.g. the current body temperature. This command is sent to the monitoring system of the patient, that performs the requested measurements and delivers the results back to the central server. Once the values are received by the server, these are saved so that they can be accessed and analysed at any given time. Furthermore, the agent controlling the central server is able to manage the patients' information and perform a basic analysis of the data.

Keywords: On demand, biomedical sensors, Arduino and eHealth, wireless network, web server, monitoring of physiological variables.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

GLOSARIO

·**ASCII:** American Standard Code for Information Interchange (Código Estándar Estadounidense para el Intercambio de Información).

·**CPU:** Central processing unit (Unidad de Procesamiento Central).

·**DHCP:** Dynamic Host Configuration Protocol (Protocolo de Configuración Dinámica de Host).

·**DNS:** Domain Name System (Sistema de Nombres de Dominio).

·**EEPROM**: Electrically Erasable Programmable Read-Only Memory (ROM programable y borrable eléctricamente).

·**FTP**: File Transfer Protocol (Protocolo de Transferencia de Archivos).

·**GLCD:** Graphic Liquid Crystal Display (Pantalla Gráfica de Cristal Líquido).

·**GPIO**: General Purpose Input/Output (Entrada/Salida de Propósito General).

·**HTML:** HyperText Markup Language (Lenguaje de Marcas de Hipertexto).

·**HTTP:** Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertexto).

·**IDE:** Integrated Development Environment (Entorno de Desarrollo Integrado).

·**JSON:** JavaScript Object Notation (Notación de Objetos de JavaScript).

·**PHP:** Hypertext Preprocessor (Procesador de Hipertexto).

·**PSK**: Pre-Shared Key (Clave Precompartida).

·**Rx:** Receiver (Recepción).

·**SQL:** Structured Query Language (Lenguaje de Consulta Estructurado).

·**SSL**: Secure Sockets Layer (Capa de Puertos Seguros).

·**TCP:** Transmission Control Protocol (Protocolo de Control de Transmisión).

·**TLS**: Transport Layer Security (Seguridad de la Capa de Transporte).

·**Tx:** Transmission (Transmisión).

·**UART:** Universal Asynchronous Receiver-Transmitter (Transmisor-Receptor Asíncrono Universal).

·**UDP:** User Datagram Protocol (Protocolo de Datagrama de Usuario).

·**UTF:** Unicode transformation format (Formato de Transformación Unicode).

·**WPA:** WiFi Protected Access (Acceso WiFi protegido).

Capítulo 1: Introducción.

Este proyecto consiste en el diseño, especificación e implementación de un sistema básico para la adquisición y el registro remoto de datos biomédicos bajo demanda, es decir, realizar un registro remoto a pacientes monitorizados in situ y localizados en diferentes ubicaciones.

El registro se lleva a cabo desde un sistema centralizado que demanda, recibe y almacena los datos en una base de datos.

El sistema implementado está formado por cuatro subsistemas, que son, sistema de adquisición de datos, sistema de recepción y registro, sistema de administración y consulta, y, sistema de control remoto de datos. Cada uno de ellos tiene unas funciones determinadas y hacen uso de tecnologías como son, servidor web, base de datos, comunicación inalámbrica y sistemas basados en microcontrolador, que nos han permitido la implementación del sistema completo.

1.1 Motivaciones

La utilización de procesos y comunicaciones electrónicas en la práctica médica, ha supuesto una revolución muy positiva de los sistemas sanitarios de diversas partes del mundo.
La demanda social sobre las nuevas tecnologías es cada vez mayor, por lo que, creemos que este proyecto es un buen avance, ya que, se ofrece un servicio de bajo coste que permite tener controlados desde un sistema centralizado a distintos pacientes que estén localizados en diferentes ubicaciones, sin necesidad de que un profesional sanitario esté continuamente revisando el estado de sus variables fisiológicas en cada ubicación. De esta forma, el profesional realiza el control desde el sistema centralizado, solicitando los datos biomédicos cuando estime oportuno.

En este proyecto se ha trabajado con Arduino que es una compañía de hardware libre y que dispone de varias placas de desarrollo hardware y software. Cada una de ellas tiene un circuito y microcontrolador integrados, y un entorno de desarrollo donde se programa cada una. En este caso, hemos usado Arduino Uno.
Además, también se ha trabajado con la placa de salud llamada e-Health, que permite conectar diversos sensores y realizar la toma de datos biomédicos. Y con un módulo WiFi, con el que se ha llevado a cabo la transmisión de datos de forma inalámbrica.

El planteamiento del proyecto ha surgido a raíz de la práctica con Arduino y la placa e-Health en una asignatura llamada Instrumentación Biomédica, impartida en el grado de Ingeniería de la Salud.

Algunos proyectos o estudios realizados con Arduino y relacionados con el registro de datos biomédicos son los que se mencionan a continuación:

·Registro de datos en SD con Arduino.

Este estudio se basa en el registro de variables biomédicas sobre una tarjeta SD, concretamente cuenta con sensores de temperatura y presión, a través de los cuales se tomarán dichas medidas y se almacenarán en la tarjeta. La programación se lleva a cabo en Arduino que dispone de la librería SD para acceder a los archivos de la tarjeta o crear nuevos.

·Arquitectura de e-Salud basada en redes inalámbricas de sensores.

Se presenta una propuesta de arquitectura basada en redes inalámbricas de sensores para el cuidado de la salud. Es una arquitectura que permite la monitorización de variables fisiológicas en pacientes que desarrollan sus actividades diarias, tanto en el hogar, como en cualquier edificio que cuente con la arquitectura propuesta.

·Aplicación móvil para monitorizar cualquier variable fisiológica.

Consiste en la monitorización de variables fisiológicas mediante una aplicación móvil desarrollada con tecnología Android y otra en PC desarrollada en Matlab. Las variables serán recogidas por diversos sensores conectados a una plataforma de bajo coste basada en Arduino y son enviados mediante comunicación serial.

En general, hay numerosos proyectos en los que se utiliza la plataforma de Arduino y su unión a la placa de salud para hacer registro de variables fisiológicas. Los nuevos avances están centrados en la transmisión de datos biomédicos a través de redes inalámbricas.

1.2 Objetivos

Teniendo en cuenta las motivaciones mencionadas en el apartado anterior, nos hemos fijado unos objetivos que se esperan alcanzar en este proyecto. Éstos son:

-Estudio, utilización e integración de diferentes componentes, tanto hardware como software, para el desarrollo de una aplicación e-Salud.

- Especificación, diseño e implementación de un sistema para la adquisición y el registro de datos biomédicos que recoja los principales requerimientos en cuanto a: gestión de pacientes en el sistema, recogida remota de datos bajo demanda, visualización y presentación básica de datos para su análisis.

- Establecer un intercambio de información desde el microcontrolador Arduino hasta el Servidor Web mediante conexión WiFi, donde la iniciativa en la comunicación y transmisión de la información la lleva el operador del sistema centralizado.

-Creación y actualización de una base de datos, con el fin de almacenar la información requerida.

-Creación de una interfaz de usuario, donde se pueda visualizar toda la información almacenada en la base de datos.

1.3 Estructura del documento

El documento se encuentra dividido en 6 capítulos principales, donde se explican los distintos aspectos para llevar a cabo la realización de este proyecto.

En el capítulo 1, encontramos una breve introducción del proyecto, así como, las motivaciones que nos han impulsado a la creación de este sistema, junto con algunos proyectos mencionados que están relacionados de una forma u otra con este proyecto. Además se establecen los objetivos fijados que se quieren alcanzar.

En el capítulo 2, se describe la arquitectura del sistema completo, mencionando los bloques implicados y haciendo una breve introducción a ellos sin llegar a profundizar, ya que, de eso se encargan los siguientes capítulos. Además, se citan las herramientas necesarias para la implementación del sistema.

En el capítulo 3, se detalla la arquitectura hardware y la estructura software del sistema de adquisición de datos. Se muestra un diagrama de conexiones y la explicación y funcionalidad de cada uno de los componentes hardware que forman parte del sistema de adquisición, como son, el microcontrolador, la placa de salud, los sensores y el módulo WiFi, y también, un diagrama de procesos y su correspondiente explicación, que recoge las tareas que forman la estructura software.

En el capítulo 4 se trata el sistema de registro de datos. Se muestra un análisis del sistema, el protocolo de comunicación que se usa entre el cliente y el servidor y la implementación del sistema y de cada una de las partes que lo forman.

En el capítulo 5, basado en el sistema de administración y consulta de los datos. Se explica el tipo de base de datos utilizada y los elementos que la forman, así como, la interfaz de usuario empleada para la visualización y solicitud de los datos.

En el capítulo 6 se muestran algunas de las pruebas realizadas y los resultados obtenidos.

También hay un apartado dedicado a conclusiones donde se resumen los principales logros que se han alcanzado.

Encontraremos un apartado dedicado a la bibliografía consultada para la realización del proyecto.

Y por último, en el apartado Apéndices, se aloja información extra que no se ha visto necesaria incluirla a lo largo del documento, como son los códigos de Arduino y de recepción de datos.

Capítulo 2: Arquitectura del sistema

En este capítulo, se presentan de forma general, los principales bloques funcionales que constituyen el sistema, y cómo se integran para conseguir la funcionalidad requerida. Se ha dejado para posteriores capítulos una descripción más detallada de su implementación, tanto de los elementos hardware y software que los constituyen, como del software específico desarrollado.

Los requerimientos del sistema se basan en realizar capturas de datos biomédicos y transmitirlos vía WiFi para almacenarlos en una base de datos. Para ello, la solución generada es la que encontramos representada en el esquema de la figura 1.

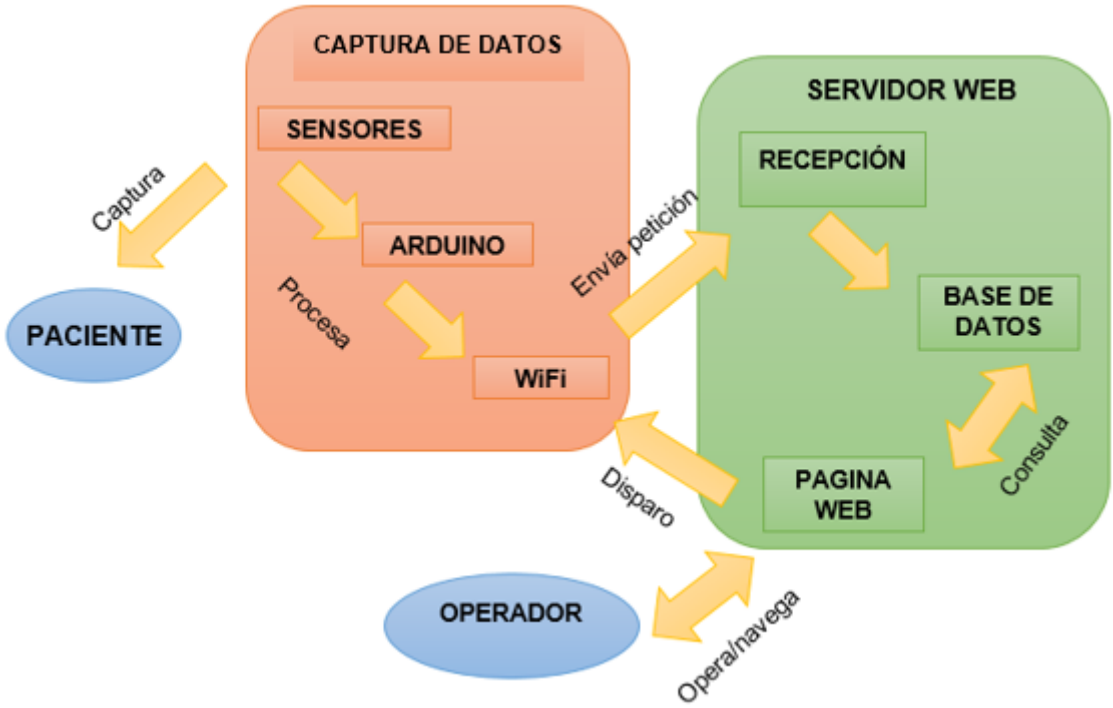

Figura 1: Arquitectura del sistema

En dicha figura, se observan dos bloques principales, por un lado tenemos un sistema de adquisición de datos que tiene interfaz directa con el paciente y, por otro lado, el servidor web donde está el sistema de gestión desde el que se realiza la demanda de datos, su almacenamiento y se proporciona el interfaz necesario para su recuperación y visualización de datos por parte del operador. La conexión entre ambos bloques se realiza de modo inalámbrico mediante una red WiFi.

El sistema de adquisición está formado por un procesador basado en Arduino UNO, un conjunto de sensores biomédicos y su circuitería de acondicionamiento y adquisición reunida en la placa e-Health para arduino, y un módulo WiFi, controlado

desde el procesador y que permite la comunicación con el sistema de gestión y, por tanto, el envío de los datos capturados.

El sistema de gestión está formado por un Servidor Web en conexión con una base de datos.

Los datos recibidos desde el sistema de adquisición son incorporados a la base de datos mediante el proceso de recepción implementado en dicho servidor, y son visualizados a través de una interfaz web.

Cada uno de estos sistemas se introduce en los siguientes apartados.

2.1 Sistema de adquisición de datos

El sistema de adquisición de datos es el encargado de realizar la toma de medidas solicitadas por el operador del sistema centralizado.

En la base de datos encontramos tres variables de estado, correspondientes a cada sensor. Cuando hablamos de variable de estado, nos referimos a un variable que puede adoptar dos estados distintos, 0 lógico o 1 lógico, indicando, según el valor que tenga, su estado inactivo o activo, respectivamente.

Antes de empezar la toma de medidas, hay que comprobar que la variable de estado, está activa. Esta consulta se puede realizar de dos formas distintas. Una de ellas, es un método al que hemos denominado polling, y la otra es un método basado en interrupciones.

El método polling es un método de consulta continua, es decir, está realizando consultas constantes a la base de datos para saber si la variable de estado está activa. En una de esas consultas, encontrará la variable con valor 1, indicando así, que se puede empezar la toma de medidas.

El método basado en interrupciones, se lleva a cabo cuando se manda la orden de solicitud de un dato desde otro sistema, el sistema de control remoto, y esta orden es recibida por el sistema de adquisición de datos. En ese momento, se manda una interrupción y se realiza una consulta a la base de datos para comprobar que la variable de estado está activa, y por tanto, empezar la captura de datos.

La variable que se encuentre activa en el momento de realizar la consulta, es la que indica la medida que se tiene que tomar.

El sistema de toma de datos está formado por la unión de la placa Arduino Uno, la placa e-salud, el módulo WiFi y una serie de sensores.

En la figura 2, se puede observar el conjunto completo que conforma el sistema de toma de datos.

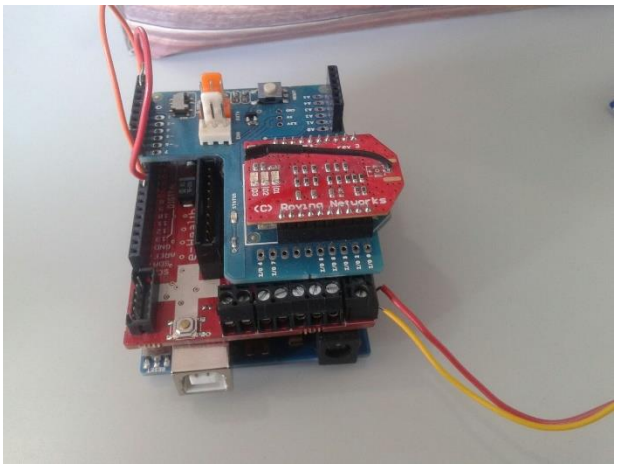

Figura 2: Sistema de adquisición de datos sin sensores conectados.

2.2 Sistema de control remoto

El sistema de control remoto es aquél que actúa en el usuario y en el sistema de adquisición.

El sistema de control remoto en el usuario, es el responsable de iniciar la captura de datos. Este sistema está controlado por un profesional sanitario que, en el momento que quiera saber un dato de un paciente, lo solicita haciendo uso de una interfaz web. Dicha solicitud implica dos acciones, una de ellas activar la variable *flag* correspondiente que se encuentra alojada en la base de datos, y otra, que manda la orden de solicitud al sistema de adquisición.

El sistema de control remoto en el sistema de adquisición, está realizando consultas constantes por polling y, la acción de mandar una solicitud de un dato, es la que adelanta esta consulta actuando como interrupción, es decir, en el momento en que se mande la orden de solicitud, se está mandando una interrupción al sistema de adquisición indicando que realice una consulta sobre la base de datos.

2.3 Sistema de registro de datos

El sistema de registro de datos es el encargado de recibir la información referente a los datos médicos capturados en el sistema de adquisición. Los datos son recibidos y almacenados en la base de datos correspondiente.

De esta forma, permite al operador del sistema consultar cualquier información de un paciente en el momento que desee.

2.4 Sistema de administración y consulta

El sistema de administración y consulta es el que permite, al profesional sanitario que esté haciendo uso de él, consultar la información que desee mediante una interfaz de usuario.

Para ello, hace uso de la base de datos y la información contenida en ella, que ha sido almacenada por el sistema de registro de datos.

Este sistema constará de un control de usuarios, y cada uno de ellos tendrá diferentes permisos para acceder a información determinada de un paciente y para realizar unas u otras acciones.

2.5 Visión general

Así pues, una vez que se ha explicado la función de cada bloque que compone el sistema global, donde la iniciativa de la comunicación la tiene el encargado del sistema de control remoto, se citan a continuación las herramientas que son necesarias para su correcto funcionamiento. Se pueden dividir en herramientas hardware y sus herramientas software correspondiente. Estas herramientas se encuentran detalladas en el apéndice A.

Herramientas hardware:

- Ordenador Pc o compatible.
- Arduino UNO
- Plataforma e-Health V2.0 para arduino y Raspberry-pi
- Kit de sensores para medidas biomédicas para Plataforma e-Health V2.0

18

- “Communication Shield” para Arduino
- Módulo WiFi para Arduino: "Roving RN-XVee"
- Router WiFi
- Herramientas eléctricas (Protoboard, leds, resistencias y cables).

Herramientas software:

- IDE arduino y librerías de comunicaciones.
- Librerías Arduino para Plataforma e-Health V2.0
- PHP, JavaScript, HTML como herramientas de desarrollo web.
- APACHE. Servidor HTTP.
- SQLite como gestor de bases de datos.

Capítulo 3. Sistema de adquisición de datos

A lo largo del capítulo 3, se profundiza en el sistema de toma o captura de datos, citado en el capítulo anterior, al que se hace referencia en la siguiente figura.

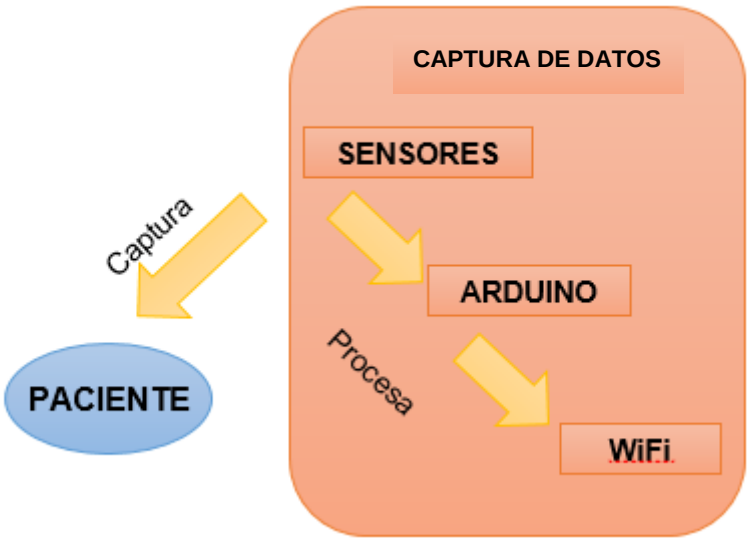

Figura 3: Sistema de adquisición de datos

En la figura 3, se puede ver que el sistema de toma de datos es el dispositivo que, a través de los sensores que tiene conectados, captura los datos y los envía a través de conexión inalámbrica vía WiFi, como ya se ha introducido anteriormente.

3.1 Arquitectura

En este apartado se procede a explicar la descripción del hardware del sistema del sistema de captura de datos.

La parte hardware del sistema de adquisición está compuesta por la unión de la placa Arduino UNO, la placa e-Health, la placa “Communication Shield” y el módulo WiFi, una encima de otra, a través de los pines digitales y analógicos, y de los respectivos conectores para el caso de la conexión de los sensores biomédicos al dispositivo.

El hardware del sistema explicado se puede observar en la figura 4.

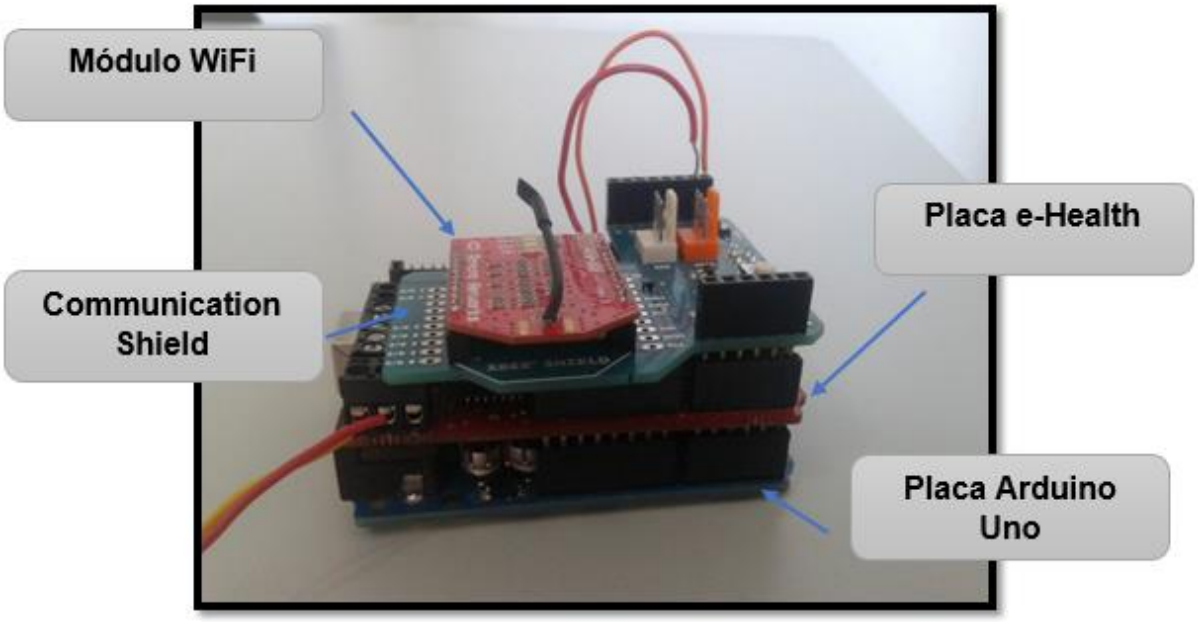

Figura 4: Dispositivo con todas las placas conectadas

La placa cuenta con pines digitales y analógicos. En los pines digitales libres se encuentra conectada una protoboard con un led, que se enciende en el momento que se inicia la captura de un dato (figura 5). La placa Arduino UNO debe estar conectada a una fuente de alimentación, ya sea un ordenador, una batería externa, etc.

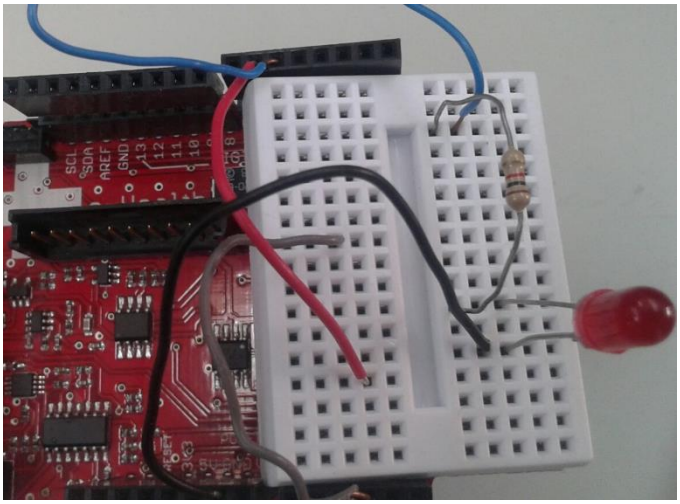

Figura 5: Placa con led

El diagrama de conexiones existente entre las distintas partes del sistema se puede ver a continuación.

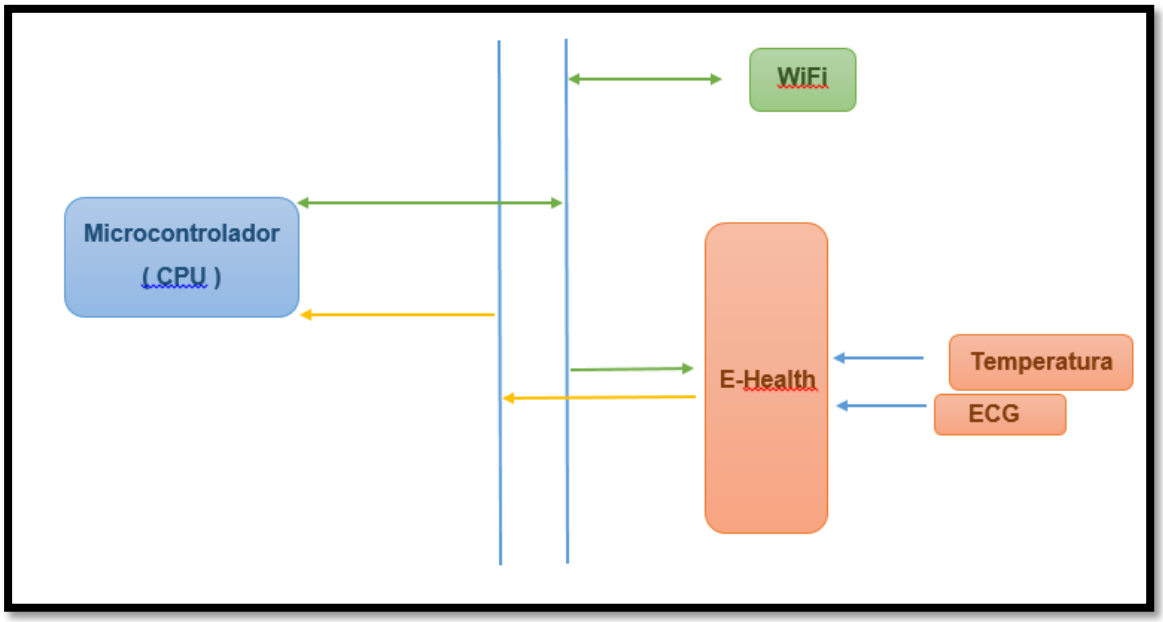

Figura 6: Diagrama de conexiones del dispositivo

En el diagrama de conexiones representado en la figura 6, se presentan dos líneas referentes a pines digitales y a pines analógicos, de manera que el microcontrolador, el módulo WIFI y la placa e-Health están conectados a pines digitales indicados por flechas de color verde, y a pines analógicos indicados con flechas de color amarillo.

El microcontrolador se conecta usando los pines digitales 0-12 y los pines analógicos 0-5, el módulo WIFI usa los pines digitales 0-9 y la placa e-Health los pines digitales 0-13 y analógicos 0-5.

3.1.1 Microcontrolador

El microcontrolador de Arduino está formado por diversos elementos que vemos en la figura 7, que permiten la conexión y unión con las otras partes del sistema.

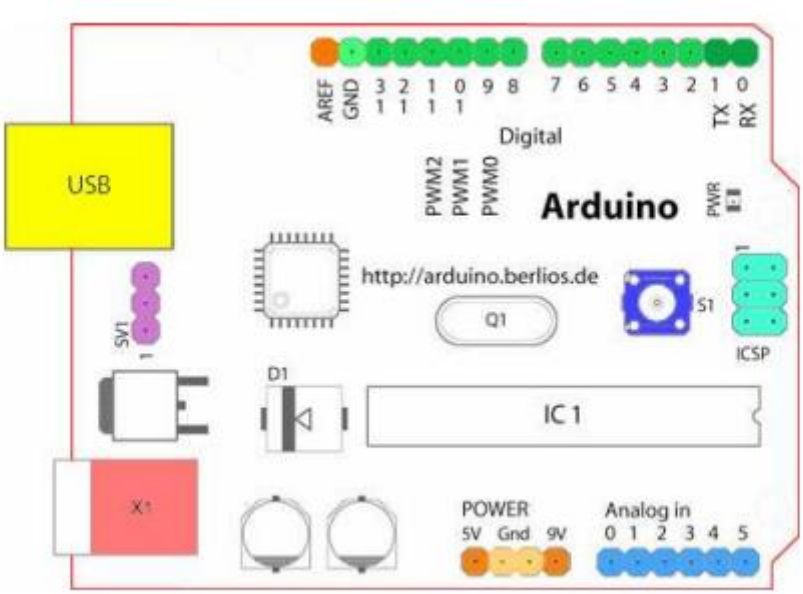

Figura 7: Esquema de conexiones de la placa Arduino

El microcontrolador dispone de un pin de referencia analógica, que se aprecia en la imagen con color naranja, y que se conoce como pin AREF. Los pines digitales van del 0 al 13 teniendo en cuenta que los pines 0 y 1 son pines digitales de entrada y salida del puerto serie, son los pines TX y RX localizados en color verde oscuro. Los pines analógicos son los pines 0-5 de color azul claro.

Además también se puede ver la señal de tierra digital en verde claro (GND), un botón de reset en azul oscuro y la entrada de la fuente de alimentación externa (X1).

En amarillo se muestra el puerto USB, que es el puerto serie que usamos para conectar el dispositivo al ordenador.

El microcontrolador puede recibir información de las entradas, la procesa y escribe un 1 o un 0 (5V ó 0V) en las salidas, actuando sobre el dispositivo que tenemos conectado, como se ve representado en la figura 8.

Figura 8: Esquema entrada y salida del microcontrolador

3.1.2 Placa e-Health

La placa e-Health se encuentra conectada a la placa de Arduino Uno, y dispone de los pines y conectores necesarios para unir a ella los sensores que se quieren usar, como son el de temperatura, pulso y ECG, en este caso.

La conexión con Arduino Uno se realiza a través de pines digitales y analógicos, 0-13 y 0-5, respectivamente.

Además de estos pines, la placa e-Health dispone de un “jumper” de EMG/ECG, un botón de GLCD, un conector para la pantalla LCD y diversos conectores para los distintos sensores, como se aprecia en las figuras 9 y 10.

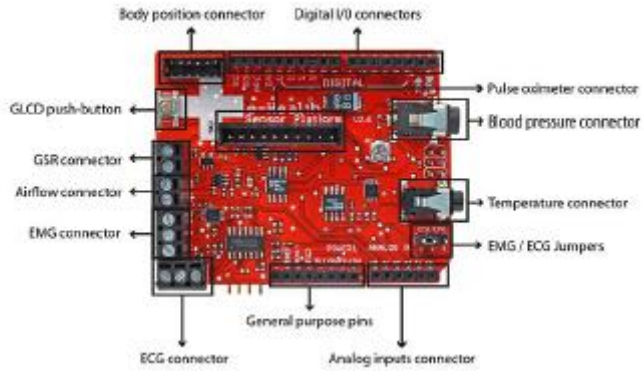

Figura 9: Placa e-Health vista desde arriba

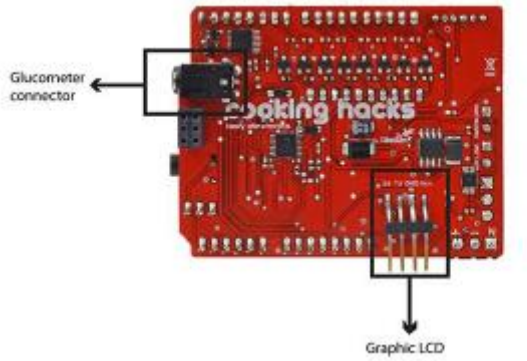

Figura 10: Placa e-Health vista desde abajo

3.1.3 Sensores

En la plataforma que integra el sistema de salud se pueden usar hasta 9 tipos de sensores distintos.

- Sensor de conductividad de la piel.

- Glucómetro.
- Electromiograma (EMG).
- Pulsioxímetro.
- Sensor de temperatura corporal.
- Sensor de presión sanguínea.
- Electrocardiograma (ECG).
- Sensor de posición del paciente.
- Sensor de flujo de aire en la respiración.

El objetivo de todos y cada uno de ellos, es obtener información acerca de una variable fisiológica de un paciente determinado, haciendo uso de las funciones que ofrece la librería e-Health.

A pesar de la gran cantidad de sensores distintos disponibles en la plataforma de salud, se han escogido tres sensores, el sensor de temperatura corporal, el sensor de pulso y el de electrocardiograma. El motivo de la elección de estos sensores es porque nos han permitido probar una transmisión de un dato concreto en el caso del sensor de temperatura, transmisión de un dato que proviene de un dispositivo inteligente, y una transmisión de un conjunto de datos en el caso del ECG. En dichos sensores, se registran datos analógicos que provienen directamente de la circuitería de amplificación y acondicionamiento.

Respecto al sensor de ECG, la placa e-Health dispone de 3 pines para poder realizar su conexión. Cada uno de estos pines pertenecen a los polos positivo, negativo y neutro, pero sólo se usan el negativo y el neutro.

Para un correcto funcionamiento, hay que colocar el “jumper” indicando que vamos a trabajar con ECG, no con EMG. Así que, debemos conectar el puente en los pines 1 y 2, que son los que pertenecen a ECG.

El sensor de temperatura, en cambio, va conectado a través de un cable a una entrada específica para él.

Por lo tanto, cada sensor mencionado está diseñado para que pueda ser usado junto con la placa e-Health, y en la figura 11, se muestran los distintos tipos de sensores y dónde irían conectados.

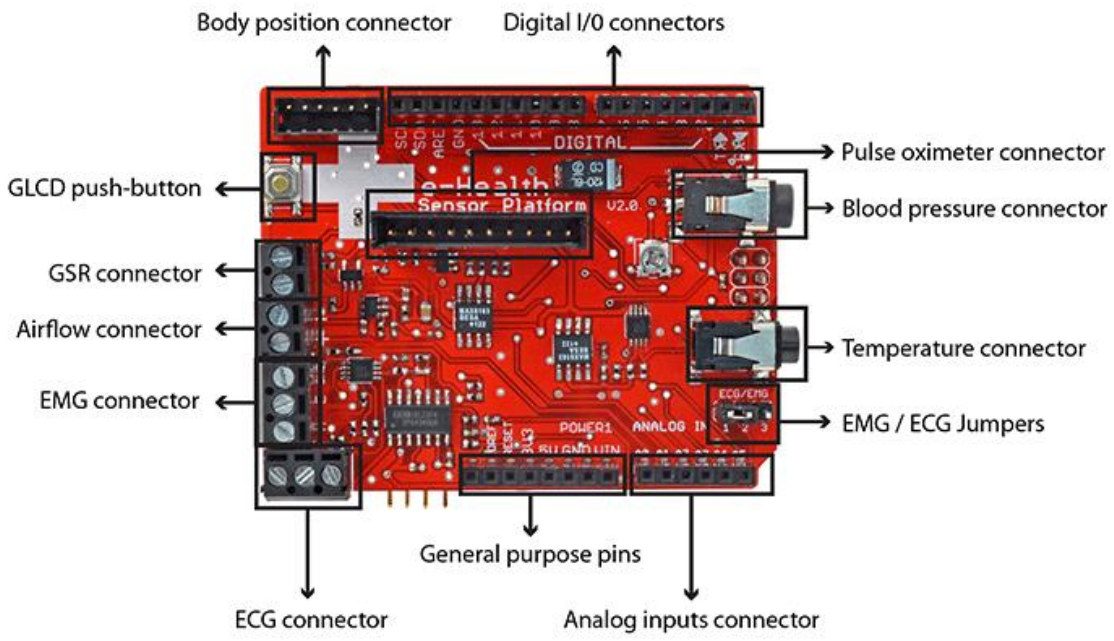

Figura 11: Conexión de los sensores a la placa e-Health

3.1.4 WiFi

El módulo WiFi es el que permite el intercambio de datos entre la placa Arduino y el servidor web mediante comunicación WiFi. Este módulo requiere de una placa de comunicación (figura 12) para unirlo al conjunto formado por la placa de Arduino y la placa e-Health.

Figura 12: Placa "Communication Shield" para Arduino

El módulo está conectado a los pines digitales 0-9 de la placa de comunicación, tiene 8 pines de entrada/salida, 3 entradas analógicas y sólo necesita 4 pines para funcionar (PWR, TX, RX y GND) y crear una conexión inalámbrica abierta.

La estructura del este módulo es la siguiente,

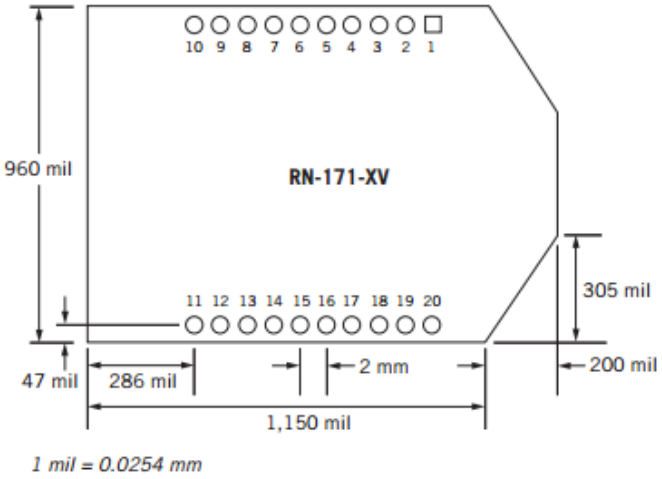

Figura 13: Estructura del módulo WiFi

La placa de comunicación, junto con el módulo WiFi, se coloca encima de la placa e-Health, y el resultado final lo vemos a continuación en la figura 14.

Figura 14: Placa e-Health junto a la placa de comunicación y el módulo WiFi

3.2 Análisis del funcionamiento

Una vez detallada la arquitectura hardware del sistema, vamos a centrarnos en este apartado, en explicar el funcionamiento del mismo.

Como ya se ha comentado, para llevar a cabo la captura de datos, se tiene que realizar una comprobación sobre la base de datos para ver que alguna de las tres variables de estado está activa. Estas variables de estado han sido denominadas como *flagTemp*, *flagPulso* y *flagECG*, correspondientes con los sensores de temperatura, pulso y ECG, respectivamente, y están alojadas en una tabla de la base de datos.

Cuando se realiza la solicitud por parte del profesional sanitario determinado, el sistema de control remoto en el usuario accede a la base de datos, activa el *flag* correspondiente al dato que haya sido solicitado y abre una conexión contra un socket en el sistema de medida, y, en el lado del sistema de medida, periódicamente se está consultando el *flag* al que se accede usando una interfaz web y que mantiene abierto un socket servidor, de forma que, si se produce una conexión sobre él, se interrumpe la espera del polling y se adelanta la consulta del *flag*.

Estas son las dos formas de consulta de *flag* que se habían mencionado, compartiendo ambas el mismo objetivo que es saber qué variable de estado está activa y, por tanto, qué dato hay que capturar.

El proceso *polling*, es una operación de consulta periódica sobre la tabla donde están alojadas las variables de estado. Este método está realizando este tipo de consulta constantemente, y en una de ellas, encontrará una variable con valor 1, indicando que ha sido activada y que se inicie la toma del dato correspondiente.

El proceso basado en interrupciones, es un método que se lleva a cabo cuando el sistema de adquisición recibe la orden de solicitud por parte del sistema de control remoto. En el momento en que recibe la orden, realiza una consulta sobre la tabla donde se encuentran las variables de estado y comprueba cuál está activa.

Una vez que termina el proceso de consulta porque se ha encontrado una variable activa, se pasa al proceso de captura de datos.

El valor de cada una de las variables *flag* es 0, es decir, por defecto están desactivadas.

Ventajas e inconvenientes del proceso *polling*

Es un proceso más seguro porque es más fácil acceder desde el sistema local de Arduino al servidor web, y además, no hay que cambiar el estado de arranque del módulo WiFi, ya que, está siempre configurado como cliente.

El principal inconveniente es que genera un exceso de tráfico en la red, que en el caso de redes con muchos elementos, podría llegar a saturarla. Por otra parte, obliga a estar trabajando a la CPU haciendo que haya un alto nivel de consumo.
Sin embargo, la importancia de este inconveniente dependerá mucho de la medida concreta, y del lapso de tiempo programado entre medidas.
Por ejemplo la consulta de un dato como la temperatura, suele realizarse con una cadencia en general baja, salvo en situaciones o episodios de alarma.

Ventajas e inconvenientes del proceso de interrupciones

Este proceso reduce el tráfico, ya que, sólo está esperando que algún cliente se conecte, y en ese momento empieza a trabajar. El principal inconveniente, es que se puede perder la interrupción y no llega al destino que queremos, pero su incidencia es despreciable.

Finalmente, lo que se pretende es que el tráfico en la red no sea excesivo y que el consumo sea menor, por eso el sistema implementado está formado por una combinación de ambos métodos, donde se usa un polling con un tiempo de espera más largo, y se realiza la consulta, cuando llega una interrupción.

3.3 Estructura del software del sistema

En puntos anteriores se ha hablado del funcionamiento y de la estructura del hardware del sistema, y, ahora se procede a explicar la estructura del software de Arduino. Para ello, se puede observar el siguiente diagrama (figura 15), que es un diagrama de procesos, formados por un bucle principal y un proceso de muestreo continuo. Ambos, son procesos que se realizan de forma paralela.

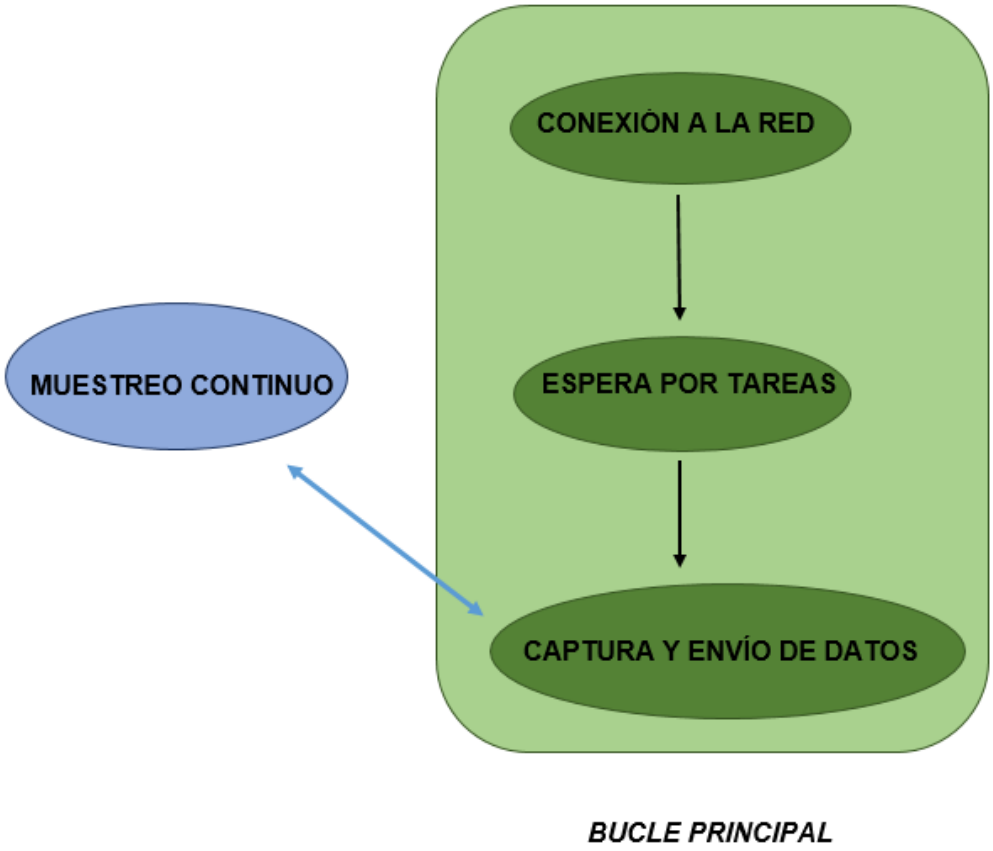

Figura 15: Diagrama de procesos del software del sistema

El bucle principal está implementado usando una máquina de estados, donde la primera tarea es la conexión a la red, de ahí pasa a la espera por tareas y, por último, a un estado de captura y envío de datos.

3.3.1 Bucle principal

El bucle principal, comienza con el estado de conexión a la red, que es lo primero que debe ocurrir para que se lleve a cabo la comunicación entre el cliente y el servidor.

En Arduino se programa la configuración del módulo WiFi, de manera que, el módulo ejecuta las órdenes y se conecta a la red correspondiente. Puede estar configurado para establecer una conexión de forma manual o automática.

La red ha sido proporcionada por un teléfono móvil, con su nombre y contraseña correspondientes. En el momento en que el módulo WiFi detecta esta red, accede a ella de forma automática.

El siguiente estado es la espera por tareas, en el cuál, el sistema se puede configurar de dos formas, como cliente o como servidor. Una vez que el sistema está conectado a una red WiFi, se configura como servidor y espera la conexión de algún cliente. Si no hay ningún cliente disponible, entonces se vuelve a la configuración inicial donde está establecido como cliente y pasa al siguiente estado.

El último estado que compone el bucle principal es la captura y envío de datos. Para que este proceso se lleve a cabo es necesario que la variable *flag* esté actualizada con valor 1, indicando así que se puede empezar la toma de medidas.

Una vez que se comprueba que el estado del *flag* es 1, se procede a capturar la medida correspondiente haciendo uso de las funciones que facilita la librería de salud. Cuando la medida está tomada, Arduino realiza una petición HTTP al servidor web para enviar el dato capturado y, que pueda ser recibido y almacenado en la base de datos por el sistema de registro.

La captura y envío de datos varían según el tipo de dato que se quiere capturar y enviar. En el caso del sensor de temperatura, es un dato simple que se recoge y se envía en una petición *GET* al servidor. Si lo que queremos capturar es una medida de pulso, entonces estamos hablando de dos valores que se tienen que capturar y mandar, uno que recoge la saturación de oxígeno en sangre y otro que recoge las pulsaciones por minuto del paciente. Son dos datos que provienen de un dispositivo inteligente. Y, por último, cuando se trata de datos de ECG se trata de un conjunto de datos los que hay que capturar y mandar.

3.3.2 Implementación del bucle principal

Este apartado trata la implementación del bucle principal, los estados que lo forman y cómo se evoluciona de un estado a otro.

La máquina de estados implementada se muestra en la figura 16.

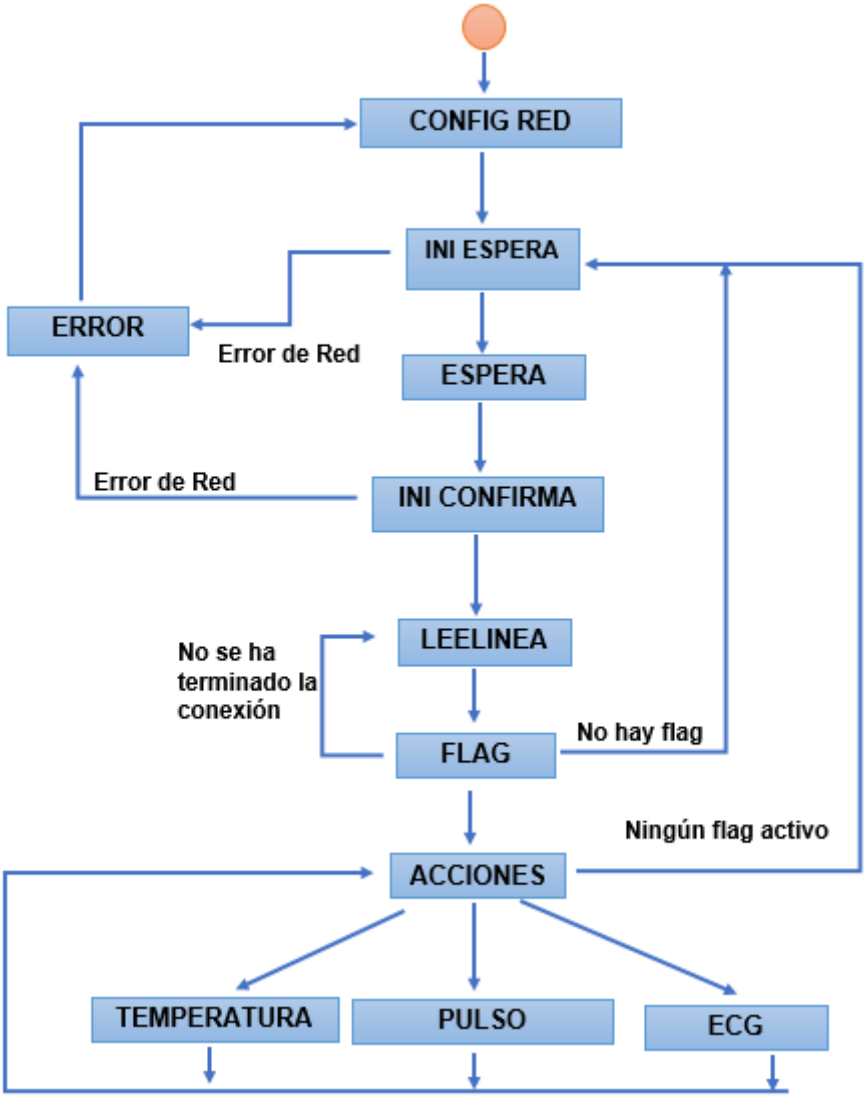

Figura 16: Representación por máquina de estados del bucle principal

Recordando las partes que integraban el bucle principal representado en la figura 15, teníamos tres procesos principales: conexión a la red, espera por tareas y, captura y envío de datos.

A continuación se procede a la explicación de cada estado, desde que se realiza la conexión a la red, hasta que se captura y envía el dato solicitado.

CONFIGURACIÓN RED

Es el primer estado con el que se inicia el proceso, ya que, lo primero que hay que hacer es configurar el módulo WiFi para conectarse a una red y poder establecer una comunicación entre el cliente y el servidor.

Se establece nombre y contraseña de la red a la que queremos conectarnos, y se configura el dispositivo como cliente con la función determinada que ofrece la librería del módulo WiFi.

Si se ha conectado a la red, muestra un mensaje indicando la unión correcta.

```
if (! wifly.isAssociated()) {
  wifly.setProtocol(WIFLY_PROTOCOL_TCP);//Se establece el prootocolo TCP
  wifly.setPort(XLOCALPORT);//Puerto 80 definido en XLOCALPORT
  wifly.setSSID(mySSID);//Nombre de red
  wifly.setPassphrase(myPassword);//Contraseña
  wifly.enableDHCP();
  if (wifly.join()) {
    Serial.println("Joined wifi network");
  }
}

Serial.print("MAC: ");
Serial.println(wifly.getMAC (s, sizeof(s)));
Serial.print("IP: ");
Serial.println(wifly.getIP(s, sizeof(s)));
Serial.print("Netmask: ");
Serial.println(wifly.getNetmask(s, sizeof(s)));
Serial.print("Gateway: ");
Serial.println(wifly.getGateway(s, sizeof(s)));

wifly.setDeviceID("Wifly-WebClient");//Modo cliente
```

Figura 17: Código Arduino correspondiente al estado CONFIGURACIÓN WIFI

En la figura 17, está el código implementado en arduino cuyo nombre de estado es *CONFIGWIFI*.

INI ESPERA

Este estado está implementado para programar lo necesario antes de pasar al siguiente estado de *ESPERA*, esto es: para abrir un socket servidor donde recibir la interrupción y programar el temporizador (timeoutflag).

Este estado está implementado en Arduino de la siguiente forma.

```
//Programamos el temporizador
timeoutflag = millis() + TIMEOUTFLAG;

//Creamos un socket servidor que recibe la interrupción
if( server != NULL ) delete server ; server = NULL ;
server = new EthernetServer(XLOCALPORT) ;
```

Figura 18: Implementación del estado INI ESPERA en el software de Arduino

En la figura 18 están representadas las dos acciones que se realizan en este estado. En la acción referente a programar un temporizador se hace uso de la función "*millis()*" que devuelve la hora actual. De esta forma, la variable que actúa como temporizador se actualiza cada vez que lleguemos a este estado teniendo en cuenta que *TIMEOUTFLAG* es una variable definida que indica el intervalo de tiempo entre dos búsquedas de flag por polling.

La otra acción es crear un socket servidor indicando el puerto donde tiene que recibir la información, es decir, la interrupción provocada cuando se solicita un dato. El puerto que está definido en la variable *XLOCALPORT* es el 80. Además, se realiza un control para saber si el servidor ya contiene información, lo que significa que ya ha recibido una interrupción anteriormente, y en este caso, se elimina la información que contiene para dejarlo libre a una futura interrupción.

Los estados que implementan estas acciones, están definidos como *CONFIGWEBSERVER* y *CONFIGTIMEOUT.*

ERROR

Este estado es a donde van aquellos estados que no han podido realizar su tarea. Por ejemplo, en el caso de no poder configurar la red WiFi, pasamos a este estado donde esperará un tiempo concreto antes de probar de nuevo a realizar la conexión.

Se añade un tiempo de espera con la función *delay()* y vuelve a reiniciarse la configuración.

```
case ERROR_ :
  // Estado en el que se espera un tiempo antes de reiniciar
  // el sistema
  delay( 60000000 ) ; // 1 minuto
```

Figura 19: Código Arduino relacionado con el estado ERROR

El código que se utiliza en arduino para implementar este estado es el que se muestra en la imagen 19.

ESPERA

Una vez que llegamos a este estado, se detecta si ha llegado alguna conexión de forma correcta. En caso negativo, se comprueba si el tiempo de espera programado ha expirado.

```
//Ha llegado una conexión
if (RedServicio.available( ) > 0) {
  Serial.println( F("Conexion entrante")) ;
  //Se muestra una página indicando la conexión
  RedServicio.flushRx() ;
  RedServicio.println(F("HTTP/1.1"));
  RedServicio.println(F("Content-Type: text/html"));
  RedServicio.println(F("Transfer-Encoding: chunked"));
  RedServicio.println();
  RedServicio.sendChunkln(F("<html><head>"));
  RedServicio.sendChunkln(F("<title>Arduino</title>"));
  RedServicio.sendChunkln(F("</head><body>"));
  RedServicio.sendChunk(F("<h1>Conectado a arduino ")) ;
  snprintf( s , 9 , "%d" , cuenta ) ;
  RedServicio.sendChunk( s ) ;
  RedServicio.sendChunkln(F("</h1>"));
  RedServicio.sendChunkln(F("<hr>"));
  RedServicio.sendChunkln(F("</body></html>"));
  RedServicio.sendChunkln();
  RedServicio.close( ) ;

 //expira el tiempo de espera
} else if (millis() > timeoutflag) {
```

Figura 20: Creación de una página web indicando que se ha conectado un cliente y comprobación del tiempo de espera.

En la figura 20, está implementado el código de arduino que hace referencia al estado ESPERA y que recibe el nombre de *TEST.*

INI CONFIRMA

En este estado de inicio de confirmación se comprueba si existe una toma de datos pendiente, que en caso de producirse un error, pasaría al estado *ERROR.*

Para saber si hay una toma de datos pendiente, hay que realizar una petición al servidor para que éste devuelva el valor de los *flags,* y así, si alguno de ellos está activo, sabemos que hay que realizar la toma de una medida.

Esto se muestra en la figura 21.

```
// Abrimos una conexión con el puerto 80, para verificar
// el estado de los flags.
if (RedCliente.open(XCENTRAL, 80)) {
  Serial.print(F("Connected to Apache 1\n"));

  //Se manda la peticion para consultar el valor de los flags
  RedCliente.print(F("GET /Pagina/medico/leeFlag.php?idarduino=3"));
  RedCliente.println(F(" HTTP/1.0"));
  RedCliente.println(F("Host: " XCENTRAL ));
  RedCliente.println(F(""));
```

Figura 21: Implementación del estado INI CONFIRMA

En dicha figura hay que abrir una conexión indicando el puerto y la dirección del servidor que está recogida en una variable definida como *XCENTRAL*, y tomará el valor de la dirección IP a la queramos conectarnos.

Una vez mandada la petición, el sistema está esperando la respuesta por parte del servidor.

Este estado se implementa en arduino en el estado *CONFIGGET.*

LEELINEA

Llegamos a este estado una vez que se ha realizado la petición y lo que queremos es leer línea a línea la salida para determinar el estado de los *flags.*

Para leer la respuesta, se hace uso de un *array* en el que se va almacenando la información recibida.

```
// Se lee linea a linea la salida
if (RedCliente.available() > 0) {
  //Inicialización del array
  nn = 0 ;
  s[nn] = 0 ;
  estado = LEELINEA ;
  //Actualizamos temporizador
  timeout = millis( ) + TIMEOUTCHAR ;
  //Expira el tiempo de espera
} else if ( timeout < millis() ) {

//Guardamos en el array la info que se va leyendo
s[nn] = RedCliente.read();
//Control en caso de llegar a fin de línea
if ((s[nn] == '\n') || (s[nn] == '\r')) {
    Serial.print( "fin de linea\n" ) ;
```

Figura 22: Implementación del estado LEELINEA en Arduino

En la figura 22, se ve como se inicializa el array que usamos posteriormente para guardar la información que se va leyendo. Se actualiza el tiempo de espera de la misma forma que se ha indicado en otros estados y, se comprueba si el tiempo de espera ha expirado, es decir, si no se recibe información tras esperar el tiempo programado.

Una vez que encuentra información disponible, se lee y se almacena en el array definido para ello hasta que llegue a fin de línea. Es entonces cuando habrá leído toda la información y pasa al siguiente estado.

Los estados implementados en arduino reciben el nombre de *ESPERAFLAG y LEELINEA.*

FLAG

En el estado *FLAG* se va a determinar el estado de los *flags.* Para ello se hace uso de la función *strstr* que compara dos cadenas, carácter a carácter. En este caso se

compara la cadena almacenada en el estado anterior en el array *s* con una cadena "*flag*".

De esta forma, puede encontrar tres cadenas, "*flagTemp*", "*flagPulso*" y "*flagECG*", como se muestra en la figura 23.

```
nn = 0 ;
Serial.print("ws: ");
if ( s[0] ) {
  Serial.print( s ) ;
}

{
  //Comparación de cadenas
  char * S = s[0] ? strstr( s , "flag" ) : 0 ;
//Caso en el que encuentra la cadena flagTemp
if ( 1 == sscanf( S , "flagTemp : %d" , &flagTemp ) ) {
   Serial.print( "Detectado :" ) ;
   Serial.print(S);
   Serial.print( "\r\n" ) ;

 //Caso en el que encuentra la cadena flagPulso
if ( 1 == sscanf( S , "flagPulso : %d" , &flagPulso ) ) {
   Serial.print( "Detectado :" ) ;
   Serial.print(S);
   Serial.print( "\r\n" ) ;

//Caso en el que encuentra la cadena flagECG
 if ( 1 == sscanf( S , "flagECG : %d" , &flagECG ) ) {
   Serial.print( "Detectado :" ) ;
   Serial.print(S);
   Serial.print( "\r\n" ) ;
```

Figura 23: Implementación del estado FLAG

En dicha figura, se encuentra implementado el código que realiza la búsqueda del *flag.* Cuando encuentre un flag, pasará al siguiente estado *ACCIONES.*

ACCIONES

Este estado se encarga de llamar a los sensores para realizar la toma de medidas correspondiente al *flag* que se ha encontrado activo en el estado anterior. Si llegamos a este estado y no hay ninguna medida pendiente para realizar, entonces vuelve al estado *INI ESPERA* para esperar que haya una nueva solicitud de medida.

En función del *flag* que se encuentre activo, se realizará una acción u otra. En la figura 24 se puede observar que, según el *flag* que ha encontrado, pasa al estado correspondiente para realizar la toma de la medida solicitada.

```
if (flagTemp == 1) {
  estado = CAPTURADATOSTEMP;
  break;
 } else
f
f CONPULSO
  if (flagPulso == 1) {
    estado = CAPTURADATOSPULSO;
    eHealth.initPulsioximeter();
    pulso_cuenta = 0 ;
    PCintPort::attachInterrupt(6, readPulsioximeter, RISING);
    digitalWrite( ledPulso , HIGH ) ;
    timeout = 30000+millis() ;
    break;
  } else
f
f CONECG
    if (flagECG == 1) {
      estado = CAPTURADATOSECG;
      break;
```

Figura 24: Búsqueda de flag activo

Si encuentra el *flag de temperatura* activo, pasa al estado donde se capturan las medidas de temperatura. Si es *flagPulso* el que está activo, pasa al estado donde se capturan los datos de pulso. Además, en ese momento se inicia el pulsioxímetro y se manda una interrupción, de manera que, al pulsar el botón del dispositivo se activa la captura de datos y el led correspondiente se enciende. Por último, si es *flagECG* el que se encuentra con valor 1, pasamos al estado donde se captura un ECG.

Los dos estados anteriores, FLAG y ACCIONES están recogidos en arduino como un único estado llamado *LEEFLAG,* en el que se realiza la búsqueda del *flag* y según el que encuentre, pasa a uno de los siguientes estados, sin pasar por un estado intermedio.

TEMPERATURA

Se llega a este estado cuando la variable encontrada a 1 es el flag de temperatura, lo que indica que el operador ha solicitado un dato de temperatura de un paciente.

En este estado se realiza la captura del dato de temperatura y se hace una petición de tipo GET, para mandarlo al servidor.

```
//Captura de un dato de temperatura
float temp = eHealth.getTemperature();
Serial.print(F("Temperatura (°C): "));
Serial.print(temp, 2);
Serial.println("");
//Petición donde se envía el dato al servidor
snprintf( s, SLENGTH,
        "GET /Pagina/medico/temperatura.php?flag=%d&temp=%d&idarduino=%d",
          (int) flagTemp , (int) temp, (int) idarduino ) ;
Serial.print( s ) ;
//Se indica la direccion y el puerto donde se envía
if ( Red.open(XCENTRAL, 80) ) {
    Red.println(s);
    Red.println(F("HTTP/1.0\r\n"));
    Red.println(F("\r\n"));
}
```

Figura 25: Captura y envío de un dato de temperatura

En la figura 25, está representada la implementación del código referente a este estado. El dato se captura haciendo uso de la función correspondiente que aporta la librería de salud, y se manda posteriormente a través de una petición HTTP de tipo GET, en la que se incluye, además del dato capturado de temperatura, el valor del *flag* y el *identificador del dispositivo arduino* que esté asociado al paciente determinado.

En arduino este estado recibe el nombre de *CAPTURADATOSTEMP.*

PULSO

Este estado de ejecuta cuando es un dato de pulso lo que el operador ha solicitado, y por tanto se ha encontrado la variable de *flagPulso* con valor 1.

Se comporta de la misma forma que el estado anterior, ya que, realiza la captura del dato y la envía al servidor a través de una petición HTTP de tipo GET.

Al realizar la captura se toman los dos valores, el porcentaje de saturación de oxígeno en sangre y las pulsaciones por minuto.

La única diferencia es que al tener dos valores, en la petición en lugar de una variable, hay que mandar las dos.

```
//Captura de pulsaciones
Serial.print("PRbpm : ");
int bpm = eHealth.getBPM();
Serial.print(bpm);
//Captura de saturacion de oxigeno
Serial.print("  SPo2 : ");
int oxigeno = eHealth.getOxygenSaturation();
Serial.println(oxigeno);
Serial.print("\n");
Serial.println("==============================");

//Envío de datos
snprintf( s, SLENGTH,
        "GET /Pagina/medico/pulso.php?flag=%d&bpm=%d&oxigeno=%d&idarduino=%d"
          (int) flagPulso , (int) bpm, (int) oxigeno, (int) idarduino ) ;
Serial.print( s ) ;
if( Red.open(XCENTRAL, 80) ) {
    Red.println(s);
    Red.println("HTTP/1.0\r\n");
    Red.println("\r\n");
```

Figura 26: Captura y envío de un dato de temperatura

De la misma forma que en el estado *TEMPERATURA,* cuando se ha mandado el dato solicitado, se manda también en la petición el *identificador de arduino,* y el *flag del pulso*.

En arduino este estado recibe el nombre de *CAPTURADATOSPULSO.*

ECG

Este estado de ejecuta cuando es un dato de ECG lo que el operador ha solicitado, y por tanto, se ha encontrado la variable de *flagECG* con valor 1.

En este caso la captura y envío se realizan de forma distinta, ya que, se trata de un conjunto de datos.

En la figura 27 se muestra como se realiza la captura de datos de ECG haciendo uso de la función correspondiente que ofrece la librería de salud. Los datos capturados se multiplican por 100 para obtener un número entero, y se almacenan en un *array*. Todo esto se ha implementado en una función denominada *void timerIsr(),* que será llamada en el momento que se empiece el envío de datos.

```
void timerIsr() {
  if (por_a < 0 || por_a >= ALENGTH) {
    return;
  }
  a[por_a++] = 100 * eHealth.getECG();
}
```

Figura 27: Captura de datos de ECG en Arduino

El envío, también es diferente porque, en este caso, se manda un conjunto de muestras, y no, una muestra individual como en el caso de temperatura.

Para mandar los datos de ECG, ya se ha explicado anteriormente que es necesario un fraccionamiento de estos datos en paquetes, que se envían al servidor uno tras otro.

Par hacer el envío, se utiliza una variable *iteración* que es la que lleva el control de los paquetes que se mandan. Cada vez que se manda un paquete, aumenta el valor de esta variable.

```
// tomamos las muestras
for (por_a = 0; por_a < ALENGTH;) {
    delay(25);
}
{
    //Creamos un string con json para guardar los datos
    String dato = "{\"datos\":[";
    //Número de datos
    int cont = 0;
    //Número de paquetes enviados
    int iteracion = 0;
    //Se guardan en el string creado
    for (int j = 0; j < ECG_LENGTH; j++) {
        dato += a[j];
        cont++;
        //Cuando haya 25 datos, se manda la iteracion 1
        if ((cont == 25) || (j == (ECG_LENGTH-1))) {
            if (j == (ECG_LENGTH-1)) {
                dato += "]}";
            }
```

Figura 28: Implementación de la captura de datos de ECG

En la figura 28, lo que se hace es crear un *string* con *JSON*. El lenguaje *JSON* tiene una estructura formada por dos campos separados por dos puntos. Como se ve en la imagen anterior, el primer campo es “datos” y el segundo campo es el *array* formado por todos los valores tomados.

Los datos que tenemos almacenados en el *array,* se van guardando en el *string* que será mandado al servidor con el conjunto de datos.

El envío de ellos mediante una petición de tipo GET se hace como se muestra en la figura 29.

```
if (Red.open(XCENTRAL, 80)) {
    Serial.print("Connected to Apache\n");
}
for ( int f = 0 ; f < 25 && !Red.available() ; f++ ) delay(10) ;
//Envío de los datos
String envio = "GET /Pagina/medico/ecg.php?idarduino=%d&flag=&cuenta=";
envio += iteracion++;
envio += "&ecg=";
envio += dato;
Serial.println(envio);
Red.println(envio);
Red.println("HTTP/1.0");
dato = ",";
cont = 0;
Red.close();
```

Figura 29: Implementación del envío de datos de ECG

La cadena que aparece en la figura 29, llamada *envio* es la que se manda a través de la petición de tipo GET indicada. En esta petición, al igual que en los casos anteriores, se manda el *identificador de arduino* y el *flag de ECG,* además de una variable *cuenta* que indica el número de paquete que se está mandando. De esta forma, tenemos controlados los datos que van llegando y qué paquete se manda.

En arduino este estado recibe el nombre de *CAPTURADATOSECG.*

3.3.3 Muestreo continuo

El muestreo de los datos solicitados por el operador se realiza de forma continua y se hace cada determinado tiempo. Para ello se ha hecho uso de la librería *TimerOne* y gracias a ella, se ha creado una variable *Timer* para controlar el muestreo de la señal.

```
Timer1.initialize(50000);
Timer1.attachInterrupt(timerIsr);
```

Figura 30: Uso de la librería "TimerOne"

En la figura 30, se ven dos acciones, una de ellas para inicializar la variable que previamente estaba dormida, y la otra para activar la interrupción, llamando a *timerIsr* definida en el apartado anterior (figura 27).

El valor elegido para la frecuencia de muestreo se establece al inicializar. El valor está en microsegundos, por lo que si usamos un valor de 50.000, lo que estamos indicando es que las muestras se toman cada 50 milisegundos. En el apartado de pruebas se pueden consultar varios ejemplos de los resultados obtenidos al usar distintos valores.

Para seleccionar una frecuencia de muestreo, hemos usado el teorema de Nyquist-Shannon, que indica que, la frecuencia de muestreo debe ser superior al doble de la máxima frecuencia a muestrear.

Si nos centramos en el hecho de realizar un ECG a pacientes pediátricos, por pruebas realizadas sabemos que es una frecuencia baja, y debe ser como mínimo de 250 Hz, aumentando así el número de muestras por segundo, es decir, alrededor de 1000. Pero, además, si lo que queremos es ser capaces de detectar enfermedades cardíacas, el número de muestras debe ser mucho mayor, entre 5000 y 10000 muestras por segundo, ya que, a mayor número de muestras que se tomen, más exacto será el electrocardiograma.

El problema es que el dispositivo tiene una capacidad limitada y no nos permite mandar un gran número de muestras, por lo que tenemos que mandar varios paquetes y cada uno de ellos con un número limitado de datos, para poder visualizar una señal de ECG.

Para determinar el número de muestras que se mandan se han hecho varias pruebas, teniendo en cuenta que lo que se quiere es enviar el mayor número de muestras posible y a una velocidad determinada para que el registro y la transmisión sean adecuadas a un electrocardiograma.

Arduino tiene limitaciones de memoria, siendo esta de 2KB de RAM, lo que significa que sólo se pueden mandar 295 muestras como máximo. Como no se pueden enviar todas a la vez, se fracciona en paquetes. Hay que tener en cuenta que, cuanto mayor sea el número de muestras tomadas, menor será la memoria disponible que quede para formar estos paquetes, pero a su vez, cuanto menor sea el número de muestras, más se tarda en enviar todo el conjunto de datos de ECG al servidor, ya que, en las pruebas realizadas, cada paquete tarda en mandarse 10 segundos. En la tabla 1, se comprueba que cuanto mayor sea el número de muestras, mayor es el tiempo que tarda en enviarse.

Número de muestras totales	Tiempo (mm : ss)
295	**50:00**
250	**3:10**
200	**1:35**
150	**0:55**

Tabla 1: Pruebas del número de muestras tomadas

En este caso, el número total de muestras que se pueden mandar es de 120 y se hace fraccionando los datos en paquetes, de manera que, se mandarán 4 paquetes con 25 datos cada uno, y un último paquete con 20 datos.

La forma de fraccionarlos y mandarlos está explicado en el apartado anterior, así como, el código completo se encuentra detallado en el apéndice B.

Capítulo 4. Sistema de registro de datos

A lo largo de este capítulo, se van a presentar los principales aspectos del sistema de control y registro de datos, una vez que sabemos cómo se capturan y envían éstos.

En primer lugar, se hace un análisis general del sistema, luego se verá qué tipos de protocolos de comunicación se usan para el intercambio de datos, y cómo se lleva a cabo la implementación de los procesos que componen el sistema.

4.1 Análisis y arquitectura del sistema

El sistema de registro de datos está formado principalmente por tres procesos. El proceso de disparo de medidas, el proceso de consulta de tareas, y el de registro de los datos que se reciben desde Arduino, desde sistema de adquisición de datos.

Estos procesos se realizan en el servidor y el primero que se lleva a cabo es el disparo de medidas, ya que, es el que indica e inicia la comunicación entre el cliente y el servidor.

En el esquema que aparece en la figura 31 se pueden ver estos procesos representados de forma gráfica.

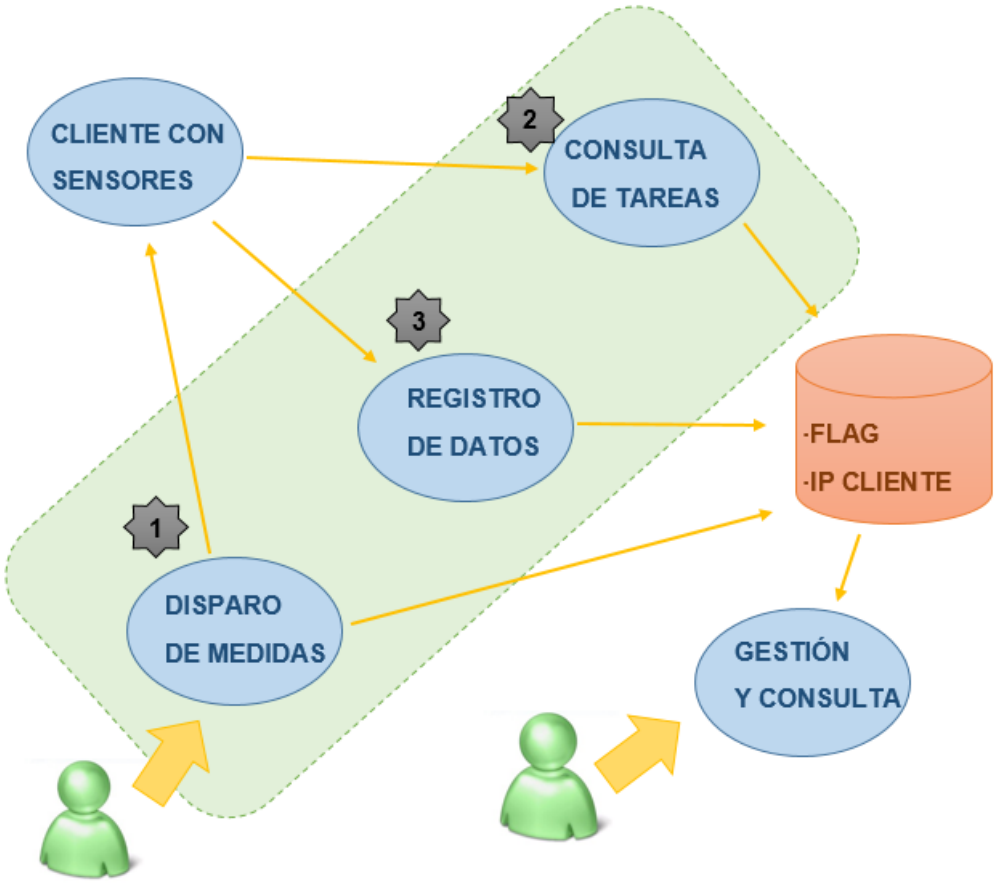

Figura 31: Esquema del sistema de control y registro

El proceso de disparo de medidas consiste en lanzar una petición ejecutada por un operador que solicita un dato a Arduino. El proceso que representa la recepción de esta petición, es el que aparece como cliente con sensores, ya que, es ahí, donde están conectados los sensores y el dispositivo de Arduino a un paciente determinado y, por tanto, donde se realiza la captura de datos.

Además, este proceso necesita interactuar con la base de datos, de manera que actualiza el estado de la variable *flag* a 1 para que comience la toma de datos.

En segundo lugar, tenemos el proceso de consulta de tareas, que permite consultar cualquier dato sobre la base de datos.

El cliente con los sensores conectados, manda una petición al proceso de consulta de tareas para consultar el valor del *flag,* y éste, hace una consulta sobre la base de datos para obtenerlo.

El proceso número 3, registro de datos, es un proceso de recepción de los datos capturados. Los datos que han sido tomados al cliente son mandados al servidor y, recibidos y almacenados en este proceso.

Además, el usuario que lleva a cabo el proceso de gestión y consulta, también accede a la base de datos para obtener la información que desee.

4.2 Protocolo de comunicación entre el cliente y el servidor web

Para que exista comunicación entre el cliente y el servidor, es necesario el uso de un protocolo de petición y respuesta. (Figura 32).

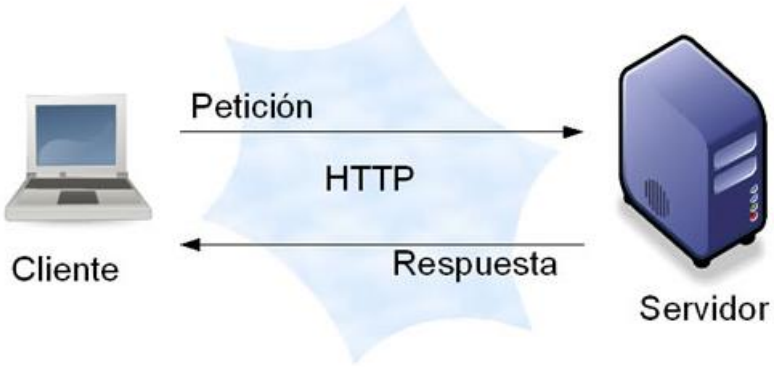

Figura 32: Comunicación Cliente/Servidor

En este caso, el protocolo HTTP es el que se usa, y es un protocolo que nos suministra dos tipos de primitivas para el intercambio de datos entre el cliente y el servidor. Son las órdenes GET y POST.

Tanto GET como POST están compuestos por un envío al servidor conocido como petición y una respuesta a dicha solicitud.

GET significa obtener información del servidor, es decir, traer datos que están en el servidor, ya sea en un archivo o en la base de datos, al cliente, teniendo en cuenta que para recibir esos datos, tenemos que enviar otros que el servidor procesa y envía la respuesta.

POST es enviar información desde el cliente para que sea procesada y actualice o agregue información en el servidor.

Son métodos que, al fin y al cabo, lo que hacen es mandar una petición al servidor y esperar una respuesta de éste, es decir, el objetivo es el mismo.

La diferencia entre los dos métodos radica en la forma de mandar los datos a la página. Mientras que el método GET envía los datos usando la URL, el método POST los envía de forma que no podemos verlos en la URL, quedan ocultos al usuario.

Algo que favorece el método POST es que se puede mandar mucha más cantidad de datos que por GET.

En este proyecto hemos trabajado con método GET, ya que, la cantidad de datos que usamos no es muy grande. Lo hacemos pasando los valores a través de la url o link.

Con el mismo tipo de protocolo HTTP con el que se envía la petición, desde hacerse la recepción de ésta.

En las siguientes dos imágenes se puede apreciar la diferencia entre una petición de tipo GET (figura 33) y una petición de tipo POST (figura 34).

```
GET /cgi/saludar.pl?nombre=pepe&email=pepe@infor.uva.es HTTP/1.0
```

Figura 33: Ejemplo de petición tipo GET

```
POST /cgi/saludar.pl HTTP/1.0
Accept: */*

nombre=pepe&email=pepe@infor.uva.es
```

Figura 34: Ejemplo de petición tipo POST

4.3 Implementación

La implementación del sistema de control y registro de datos se realiza en el servidor, ejecutando distintos scripts que hacen uso del lenguaje PHP.

Este lenguaje nos permite recibir la información que se manda desde el dispositivo arduino, e interpretarla y trabajar con ella.

4.3.1 Implementación del proceso Disparo de medidas

Como se ha comentado, este proceso comienza cuando el operador solicita un dato a Arduino en el servidor, y actualizando la variable de estado a 1.

Esto se realiza con lenguaje PHP como se muestra a continuación.

```php
//Guardamos los valores introducidos desde la página web
$idpacientesolic=(htmlspecialchars($_POST['idpacientesolic']));
$medida=(htmlspecialchars($_POST['valor']));

//Apertura de la base de datos
$db = new SQLite3('C:/xampp/htdocs/Pagina/final.db');
$redireccionamiento="";
//Se comprueba cuál es la medida solicitada
if($medida == "temperatura"){
    //Actualizamos la variable flag a 1.
    $db->exec("UPDATE PacienteArduino SET flagTemp=1 WHERE idpaciente='$idpacientesolic'");
    //Se hace la consulta para comprobar la actualizacion anterior
    $sql = "SELECT*FROM PacienteArduino where idpaciente='$idpacientesolic'";
    $result = $db->query($sql);
    while ($row = $result->fetchArray(SQLITE3_ASSOC)){
        echo $row['idpaciente'] . ': ' . $row['flagTemp'] .'  '. '<br/>';
    }
```

Figura 35: Solicitud de un dato de temperatura en el servidor

Como se observa en la figura 35, lo primero que se hace es, una vez guardados los datos que el usuario introduce a través de la interfaz web, abrir la base de datos y, si el dato es de temperatura, como en este caso, se actualiza a 1 el valor de la variable *flagTemp* alojada en la tabla *PacienteArduino.*

Si el dato solicitado es otro distinto de temperatura, se procede de la misma forma, actualizando la variable asociada a ese dato.

Para que el operador de la interfaz web pueda solicitar un dato, se dispone en el servidor web de script como el que tenemos a continuación.

```html
<html>
    <head>
        <title>TFG</title>
    </head>
    <body>

        <form action="listapacienteSOLO.php" method="POST">
        ID del paciente: <input type="text" name="idpacientesolic" /><br>
        Medida a solicitar: <input type="text" name="valor" /><br>
        <input name="submit" value="Solicitar" type="submit" />
        </form>

    </body>
</html>
```

Figura 36: Solicitud de datos en la Página Web

El código *html* que aparece en la figura 36, es el que permite que el operador indique en la página web el dato que quiere solicitar y de qué paciente.

4.3.2 Implementación del proceso de registro de datos

Después de solicitar un dato al cliente, éste envía la medida capturada al servidor, que debe procesarla y registrarla en la base de datos.

En este caso, se crean dos *scripts* distintos para cada medida solicitada. Un archivo que recibe la información del cliente y la procesa y registra, y otro, que es el que permite la visualización de esta información en la interfaz web.

El proceso de recepción y registro no se hace de la misma forma, si se trata de un dato de temperatura, de pulso, o si es un dato de ECG.

A continuación, se puede ver en los siguientes diagramas de flujo los pasos que sigue desde que se recibe un dato hasta que se almacena, ya sea un dato de temperatura, pulso o ECG, respectivamente.

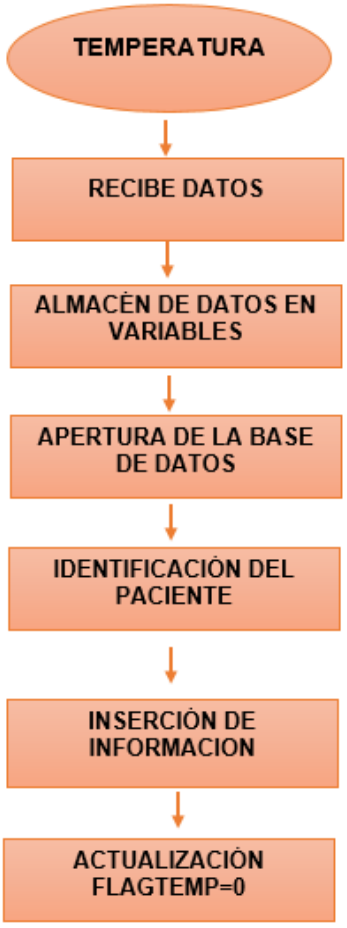

Figura 37: Diagrama de flujo de recepción de datos de temperatura.

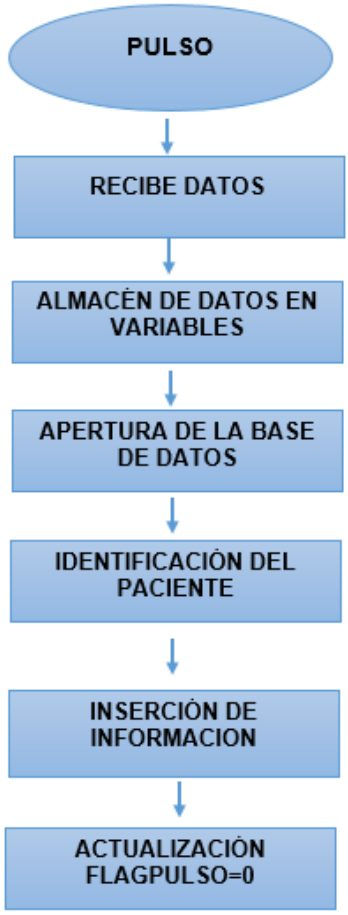

Figura 38: Diagrama de flujo de recepción de datos de pulso

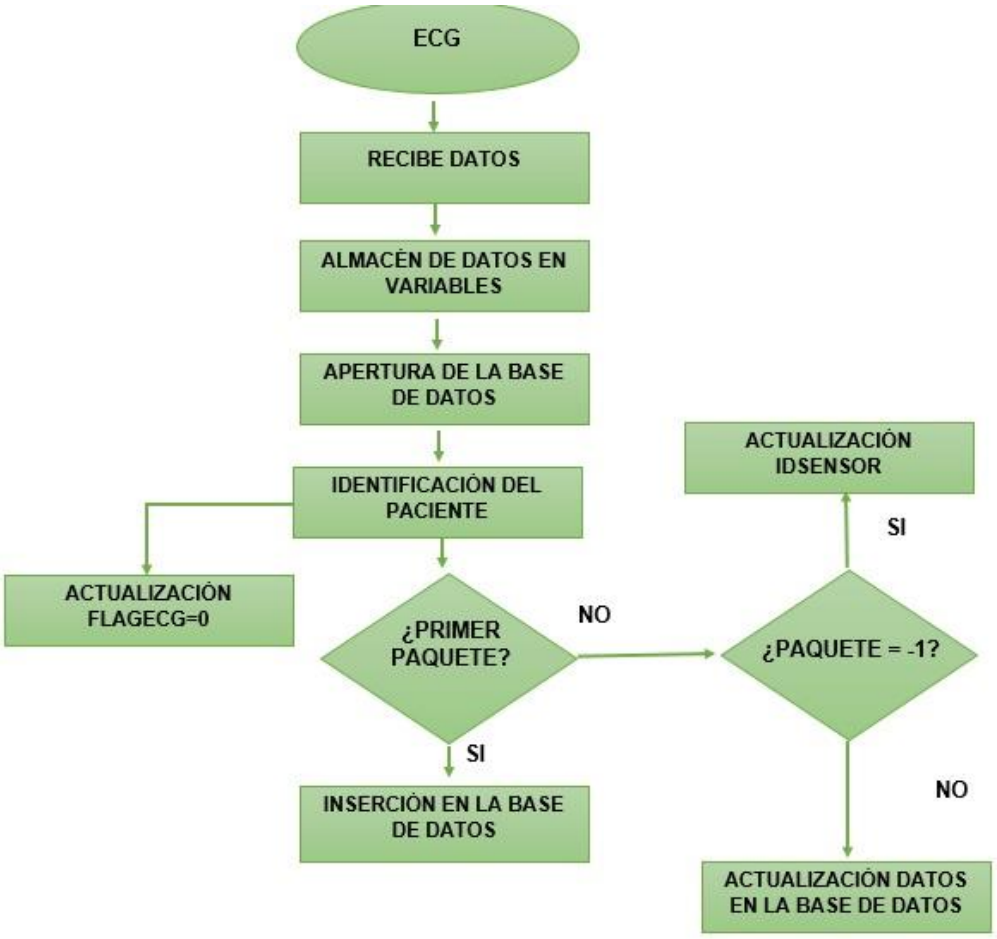

Figura 39: Diagrama de flujo de recepción de datos de ECG.

Como se muestra en los diagramas de flujo de las figuras 37, 38 y 39, temperatura, pulso y ECG comparten los primeros pasos. Una vez que se recibe el dato desde arduino, se guarda en una variable para poder trabajar con ella. Es necesario realizar la apertura de la base de datos indicando la ruta de su localización y se realiza una consulta para obtener el identificador del paciente con el que estamos trabajando.

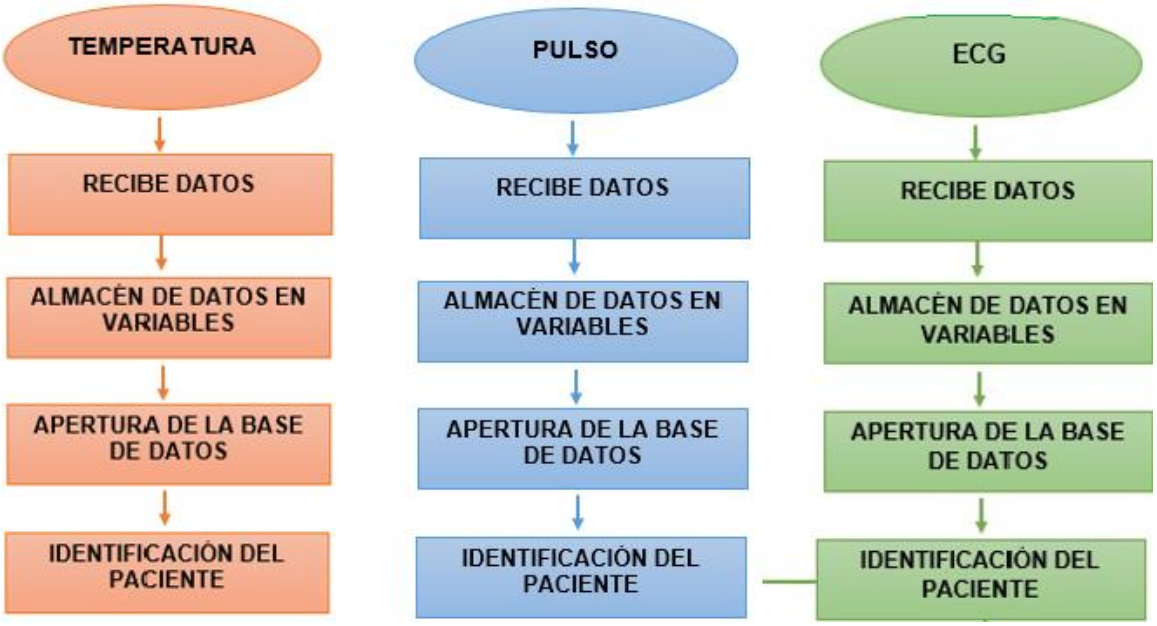

Figura 40: Procesos compartidos en la recepción de datos de temperatura, pulso y ECG

La diferencia reside en la forma de insertar los datos en la base de datos. No sólo se insertan en tablas distintas, sino que, en el caso de temperatura y pulso es una inserción sencilla de datos simples, pero en el caso del ECG, se trata de insertar varios paquetes de datos.

Cada uno de estos paquetes se manda de forma separada, es decir, primero llega un paquete identificado como paquete 0, y luego van llegando los demás. De esta forma, el primer paquete se inserta directamente de forma usual, pero, el resto de paquetes que van llegando no se pueden insertar de la misma manera porque sustituye los datos ya existentes. Así que, en lugar de realizar una inserción, hay que hacer una actualización de la variable en la que se almacenen los datos de cada paquete, teniendo al final, todos los datos de los paquetes recibidos enlazados unos detrás de otros en la misma variable.

El código en lenguaje PHP, localizado en el servidor, correspondiente al registro y almacén de datos, se encuentra en el Apéndice C.

4.3.3 Implementación del proceso de consulta de tareas

Como ya se ha comentado, este proceso es el que permite hacer la consulta del *flag* o la dirección IP que tiene el cliente, sobre la base de datos. El resultado de esta consulta, se puede visualizar en la página web, pero no se manda al cliente.

Desde arduino se envía una petición al servidor, solicitando el estado del flag. Es entonces cuando, el servidor recibe esta petición y la procesa. (Figura 41).

```php
if(isset($_GET['idarduino'])){

//Almacenamos la variable
$idarduino = htmlspecialchars($_GET['idarduino'],ENT_QUOTES);

//Apertura de la base de datos
$db = new SQLite3('C:/xampp/htdocs/Pagina/final.db');

//Se hace la consulta que devuelve el estado de cada flag
$sql =  "SELECT*FROM PacienteArduino where idarduino=$idarduino";

$result = $db->query($sql);
while ($row = $result->fetchArray(SQLITE3_ASSOC)){
    echo ('flagTemp : '.$row['flagTemp'].'<br/>');
    echo ('flagPulso : '.$row['flagPulso'].'<br/>');
    echo ('flagECG : '.$row['flagECG'].'<br/>');
}

//Actualizamos la variable que contiene la direccion IP
$a=$_SERVER['REMOTE_ADDR'];
$db->exec("UPDATE PacienteArduino SET IP='$a' where idarduino='$idarduino'");
unset($db);
```

Figura 41: Código en lenguaje PHP que implementa la consulta del "flag" en el servidor

En la figura 41, se ve como recibe la variable *idarduino* que se envía en la petición mandada desde el dispositivo y, una vez abierta la base de datos, se hace la consulta sobre ella de los *flags*, así como, se actualiza la IP usando la estructura *$_SERVER ['REMOTE_ADDR']* que nos da la IP del cliente que esté conectado en ese momento, y la asignamos en la base de datos.

En definitiva, el sistema de control y registro es el que permite que los datos capturados sean procesados y almacenados en el servidor.

Capítulo 5. Sistema de gestión y consulta

En este capítulo vamos a profundizar en el sistema de gestión y consulta.

Este sistema es el responsable de que el operador pueda llevar a cabo una gestión adecuada, es decir, que pueda realizar acciones concretas sobre la base de datos.

La base de datos está alojada en el servidor APACHE y es de tipo relacional, ya que permite establecer interconexiones entre sí y relaciones entre los datos. Este tipo de base de datos tiene algunas características:

- Los datos se organizan en tablas, donde cada tabla a su vez es un conjunto de campos (columnas) y registros (filas).

- La relación entre dos tablas (tabla padre y tabla hijo) se lleva a cabo a través de las claves primarias y claves foráneas.

- La clave primaria es la clave principal de un registro dentro de una tabla y ésta identifica de forma única a cada fila de una tabla.

- Las claves foráneas se colocan en la tabla hija, contienen el mismo valor que la clave primaria del registro padre.

Para la creación de la base de datos, se introduce la siguiente línea:

Sqlite3 nombre.db. Está línea crea una base de datos *Sqlite* con el nombre "*nombre*" en el directorio que se haya seleccionado.

Se ha creado en el modo comando CMD. Se accede a ella a través del ejecutable, introduciendo una serie de comandos, hasta llegar a su ruta donde está situada. Debe permanecer en una carpeta dentro del servidor al igual que todos los archivos que procesan la recepción de datos. (Figura 42).

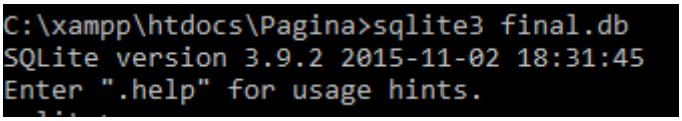

Figura 42: Acceso a la base de datos

Una vez que ya estamos dentro de la base de datos ya creada, se pueden realizar todo tipo de acciones, creación y eliminación de tablas, consultas concretas de una tabla seleccionada, etc.

La base de datos, formada por 5 tablas relacionadas unas con otras, es la que aparece en la figura 43.

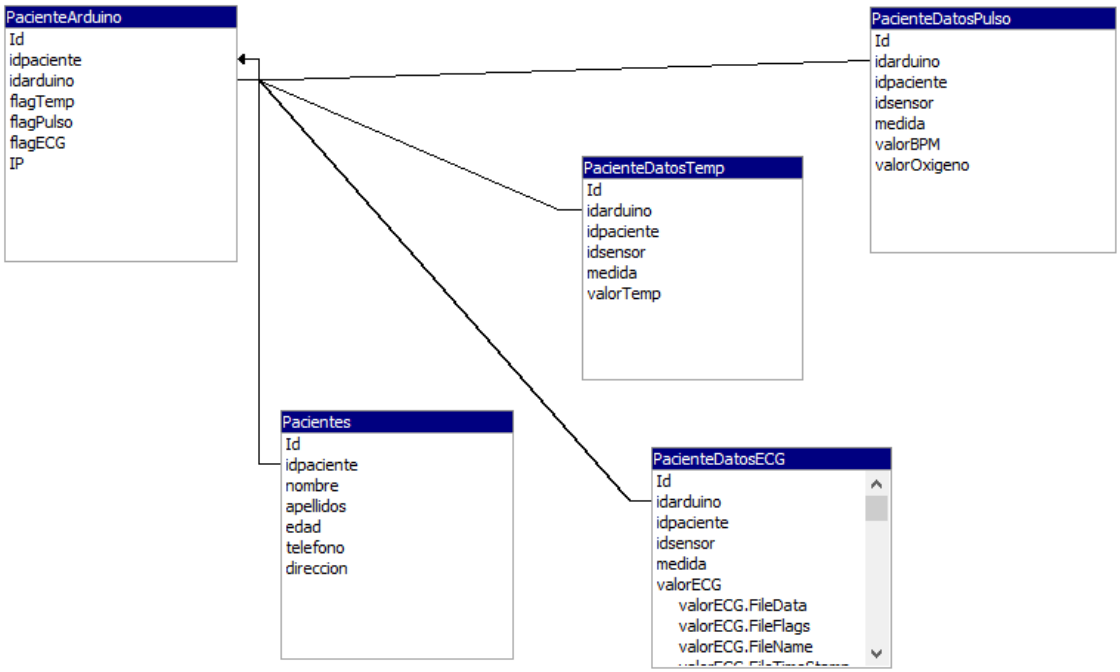

Figura 43: Base de datos relacional del sistema de monitorización

Se puede ver qué tablas tenemos ya creadas en CMD. (Figura 44).

```
sqlite> .tables
PacienteArduino     PacienteDatosPulso  Pacientes
PacienteDatosECG    PacienteDatosTemp
```

Figura 44: Consulta de tablas sobre la base de datos en CMD

Por lo tanto vemos que las tablas que alberga la base de datos son:

- PacienteArduino que relaciona cada paciente registrado en la tabla Pacientes, con el arduino que tenga asignado. Esta tabla alberga la variables *flag* correspondientes a cada sensor, y sobre ella, se realizan las consultas para consultar el estado de la variable. Por tanto, esta tabla tiene la dirección IP del cliente, un identificador de paciente, un identificador de arduino y los flags.

- PacienteDatosECG, PacienteDatosPulso, PacienteDatosTemp. Estas tablas se han creado para almacenar la información relativa a las medidas que se hayan solicitado a un paciente concreto. Cada una de ellas tiene variables relacionadas con el identificador de paciente y del arduino que ha capturado los datos.

Están compuestas por un identificador de paciente, un identificador de arduino, un identificador del sensor, el nombre de la medida que se está tomando y el valor correspondiente.

- Pacientes que almacena los pacientes que sean dados de alta junto con su información personal. Tiene variables para guardar nombre, apellidos, edad, teléfono y dirección.

Todas ellas están relacionadas entre sí, lo que permitiría hacer consultas de varias tablas distintas que tienen variables en común.

5.1 Página web

En el sistema de gestión y consulta no sólo hay que tener en cuenta la base de datos, ya que, una parte importante para ver que todo funciona correctamente, es poder visualizar los datos que se capturan. Para ello, se hace uso de una página web.

La página web alojada en el servidor es una web simple, porque no es el objetivo del proyecto, pero suficiente para una correcta visualización de los datos.

La página principal de la web es la que aparece en la figura 45.

Figura 45: Pantalla principal de acceso a la Página Web

Está caracterizada por tener un control de usuarios, de manera que, si tomamos como escenario un hospital, existen tres tipos de usuarios que pueden operar y navegar con ella:

- Administrador.

El administrador de la página web accede con un nombre de usuario "*admin*" y una contraseña "*12345*".

Al ser el administrador, tiene permisos para realizar todo tipo de acciones sobre ella, como son:

·Añadir un nuevo paciente

·Eliminar un paciente

·Solicitar datos

·Visualizar datos

·Modificar datos

·Mostrar un listado de pacientes ordenados por orden alfabético, por edad o por identificador.

·Asignar arduino

·Información de un paciente

En la figura 46 se puede ver el aspecto que tiene el menú principal del administrador.

Figura 46: Menú principal del administrador de la Página Web

- Médico

El administrador de la página web accede con un nombre de usuario “*medico*” y una contraseña “*22222*”.

·Añadir un nuevo paciente

·Solicitar datos

·Visualizar datos

·Modificar datos

·Mostrar un listado de pacientes ordenados por orden alfabético, por edad o por identificador.

·Información de un paciente

La figura 47 nos muestra el menú del médico cuando accede a la Página Web.

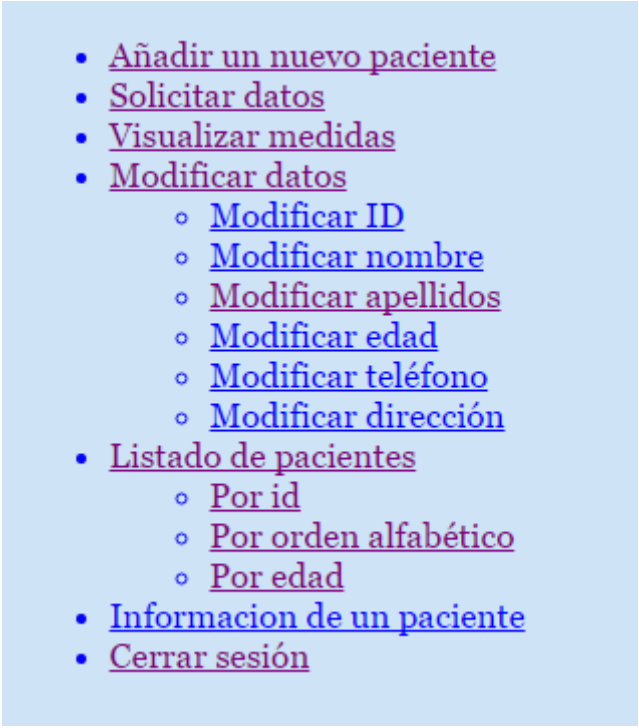

Figura 47: Menú principal del médico que usa la Página Web

- Enfermero

El administrador de la página web accede con un nombre de usuario “*admin*” y una contraseña “*55555*”.

·Mostrar un listado de pacientes ordenados por orden alfabético, por edad o por identificador.

·Visualizar datos

·Información de un paciente

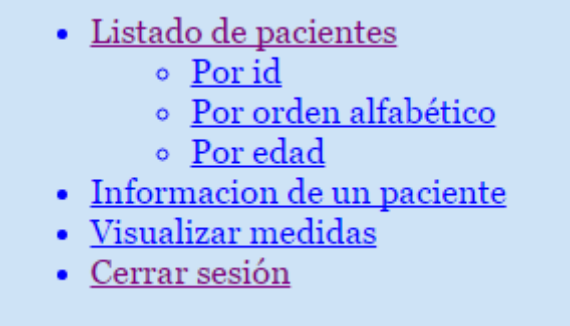

Figura 48: Menú principal del enfermero que usa la Página Web

El menú correspondiente a un enfermero es el mostrado en la figura 48.

5.1.1 Arquitectura de la página web

El siguiente esquema recoge la arquitectura de la página web que se ha ido comentando. En él se encuentran los diferentes usuarios que hacen uso de ella, así como, el conjunto de acciones que puede realizar cada uno de ellos.

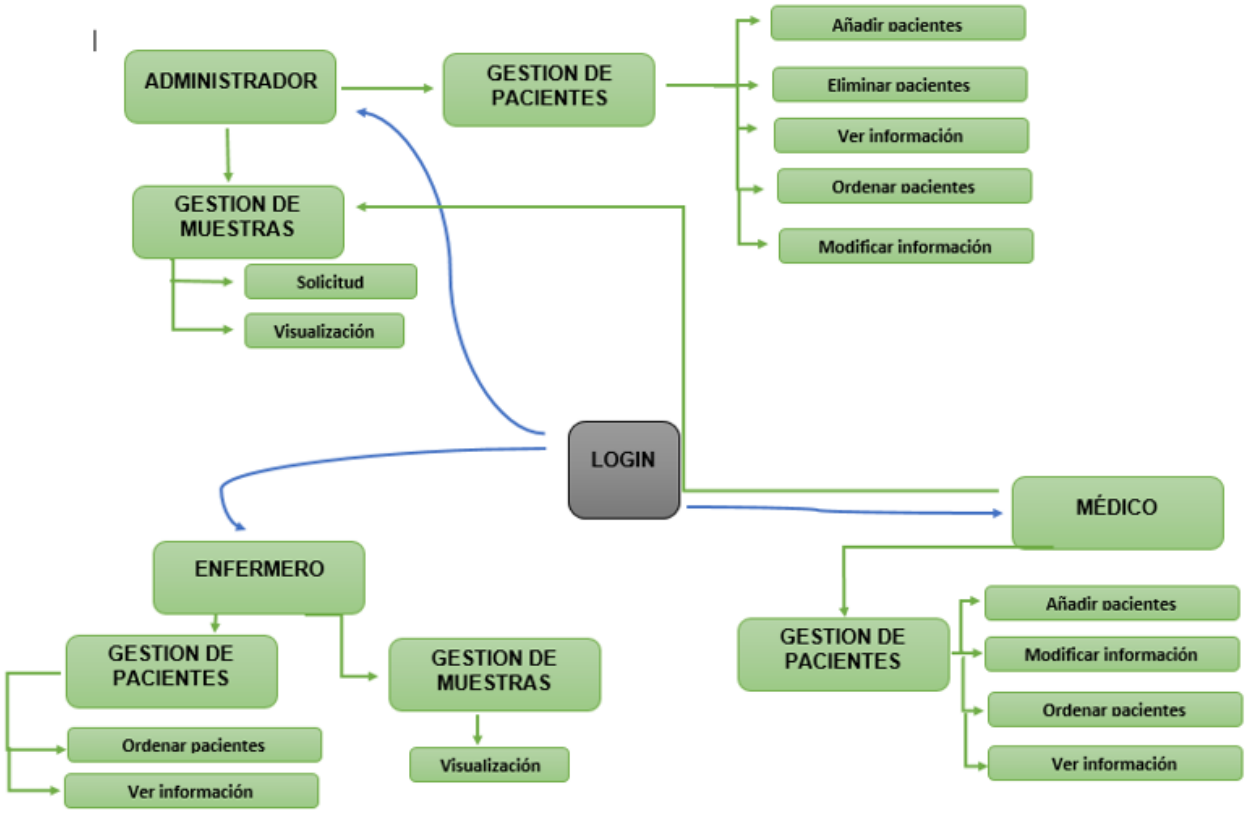

Figura 49: Arquitectura general de la Aplicación Web

Se puede ver en la figura 49 que, Administrador y Médico, son usuarios que comparten gran cantidad de acciones, porque son los que mayor uso darán a la web. En cambio, un enfermero, se considera en este proyecto, que el número de acciones que realiza es mucho más limitado, centrándose en consulta y visualización de datos, básicamente.

Capítulo 6. Pruebas y resultados

Se han realizado diversas pruebas para comprobar el correcto funcionamiento de la base de datos, el de la página web, conexión WiFi y la transmisión de datos y, por tanto, la conexión entre Arduino y la página web.

6.1 Pruebas sobre la base de datos

Consultas para que muestre todos los pacientes de la tabla Pacientes, o bien, si queremos un paciente determinado estableciendo una restricción. Por ejemplo, sobre la tabla PacienteArduino para que muestre qué valor de *flag* está a 1, que es una de las consultas principales en este proyecto, ya que, es lo que indica que se puede comenzar la toma de medidas. La consulta que nos muestra los valores del *flag* sería tal y como aparece en la figura 50.

```
sqlite> SELECT*FROM PacienteArduino where idpaciente='25351385';
25351385|3|1|0|0
```

Figura 50: Consulta sobre la tabla PacienteArduino

Si lo que queremos es consultar una medida concreta, la consulta la haríamos sobre la tabla que contiene la información de los valores medidos. De esta forma, si queremos saber el valor de la temperatura corporal de un paciente indicado: (Figura 51).

```
sqlite> SELECT valorTemp FROM PacienteDatosTemp where idpaciente='25351385';
36
```

Figura 51: Consulta de un dato de temperatura

Y del mismo modo si queremos saber datos de pulso o ECG.

6.2 Pruebas sobre la página web

Pruebas para comprobar que hay un control correcto de usuarios, para que se visualicen los datos, para ver un listado de pacientes, para ver la gráfica del ECG, añadir y eliminar pacientes, etc. En general pruebas realizadas para comprobar que las acciones que se añaden en la página web funcionan correctamente.

Si se accede de forma correcta o incorrecta, muestra un mensaje en la página web indicándolo. Si se accede correctamente, indicará el permiso que tiene. (Figura 52).

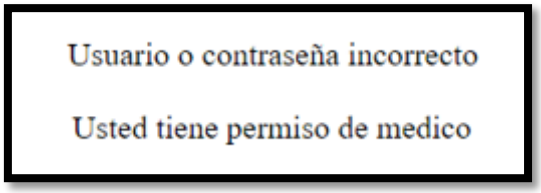

Figura 52: Control correcto o erróneo de usuarios

Cuando añadimos un nuevo paciente, la interfaz web que aparece es la siguiente:

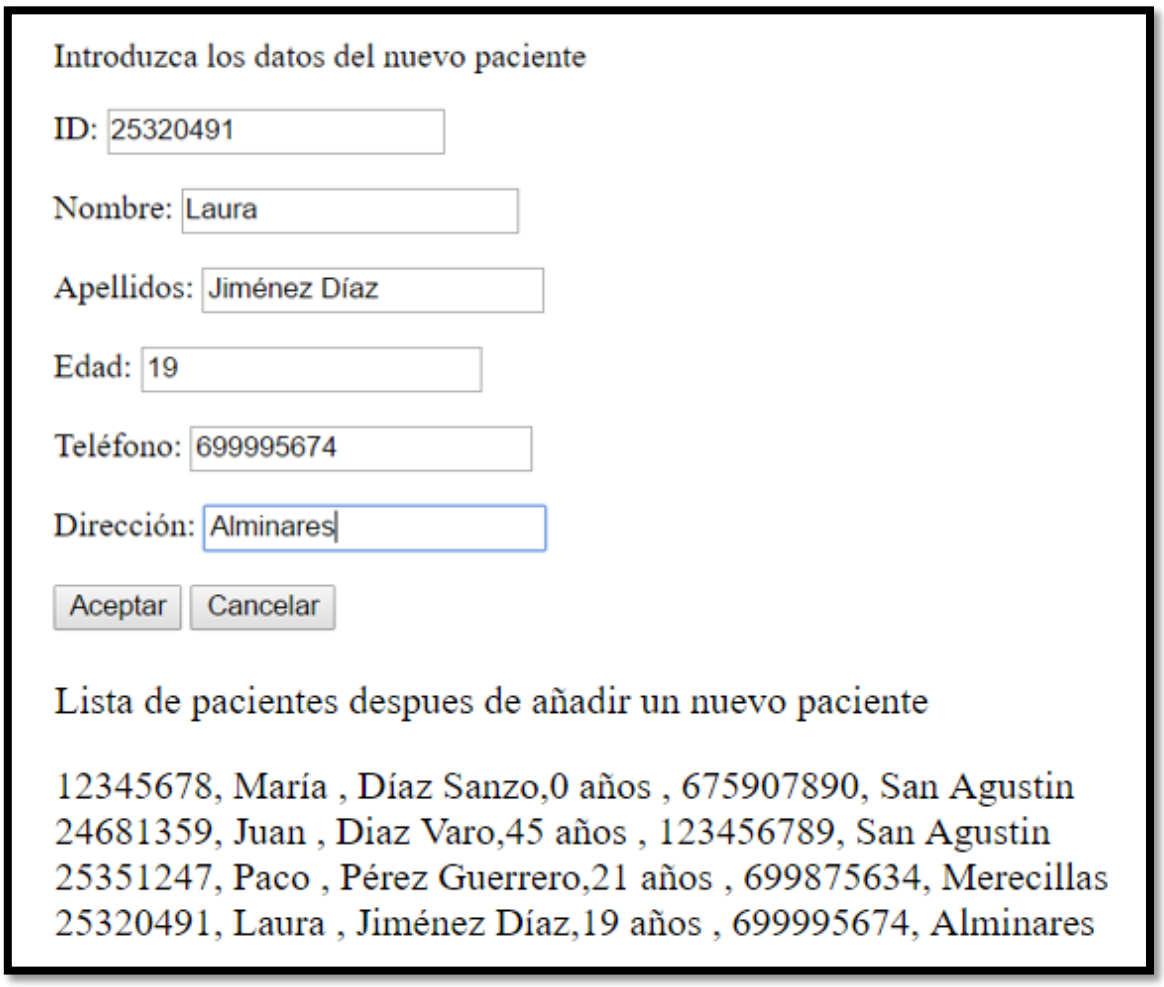

Figura 53: Procedimiento para añadir un nuevo paciente a la base de datos

En la figura 53 nos muestra toda la información que se solicita para que el operador de la página realice un registro de un paciente que no está en la base de datos. Cuando se introducen todos los datos de forma correcta, nos muestra un listado con los pacientes donde se puede comprobar que el último registro se ha realizado correctamente.

A la hora de visualizar una medida, debemos introducir el paciente y la medida que queremos solicitar para que posteriormente nos devuelva lo que hemos pedido. (Figura 54).

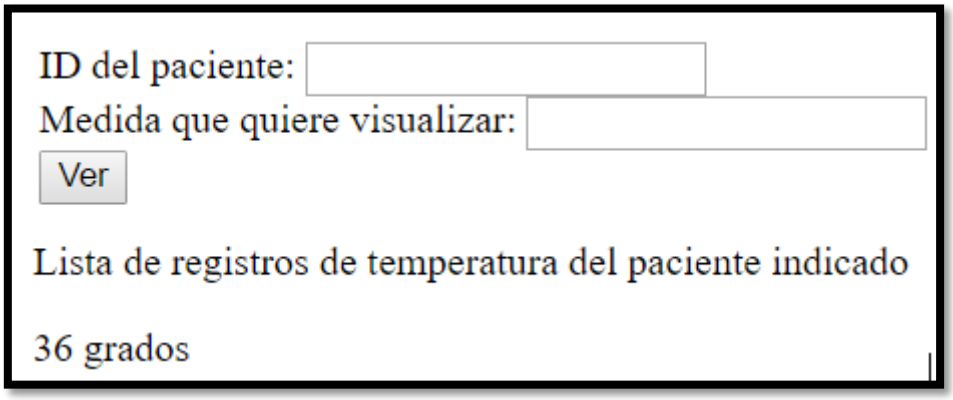

Figura 54: Proceso para visualizar una medida

Una de las pruebas que más se han realizado es la representación gráfica de los datos capturados de ECG.

Para estas pruebas, era necesario atenuar la señal montando un divisor de tensión en una protoboard(figura 55), y con ayuda del generador de señales (figura 56) y del osciloscopio se iba probando a diferentes frecuencias para ver el comportamiento según la frecuencia de muestreo utilizada.

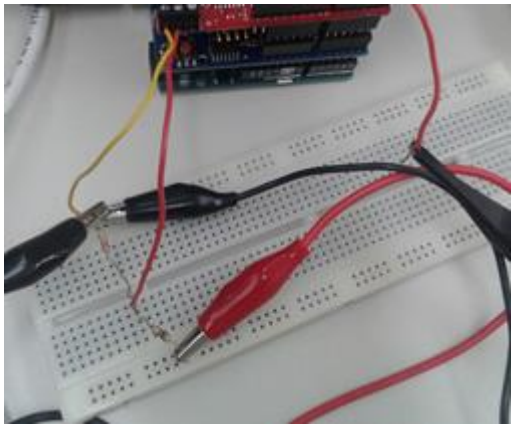

Figura 55: Divisor de tensión en la protoboard

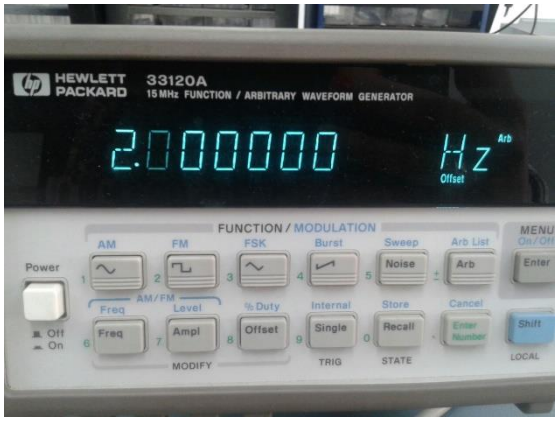

Figura 56: Generador de señales

En la siguiente tabla (Figura 57) se recogen distintas gráficas de las pruebas realizadas. La columna de la derecha es la gráfica que se obtiene en el osciloscopio y la de la izquierda la gráfica que se visualiza a través de la página web.

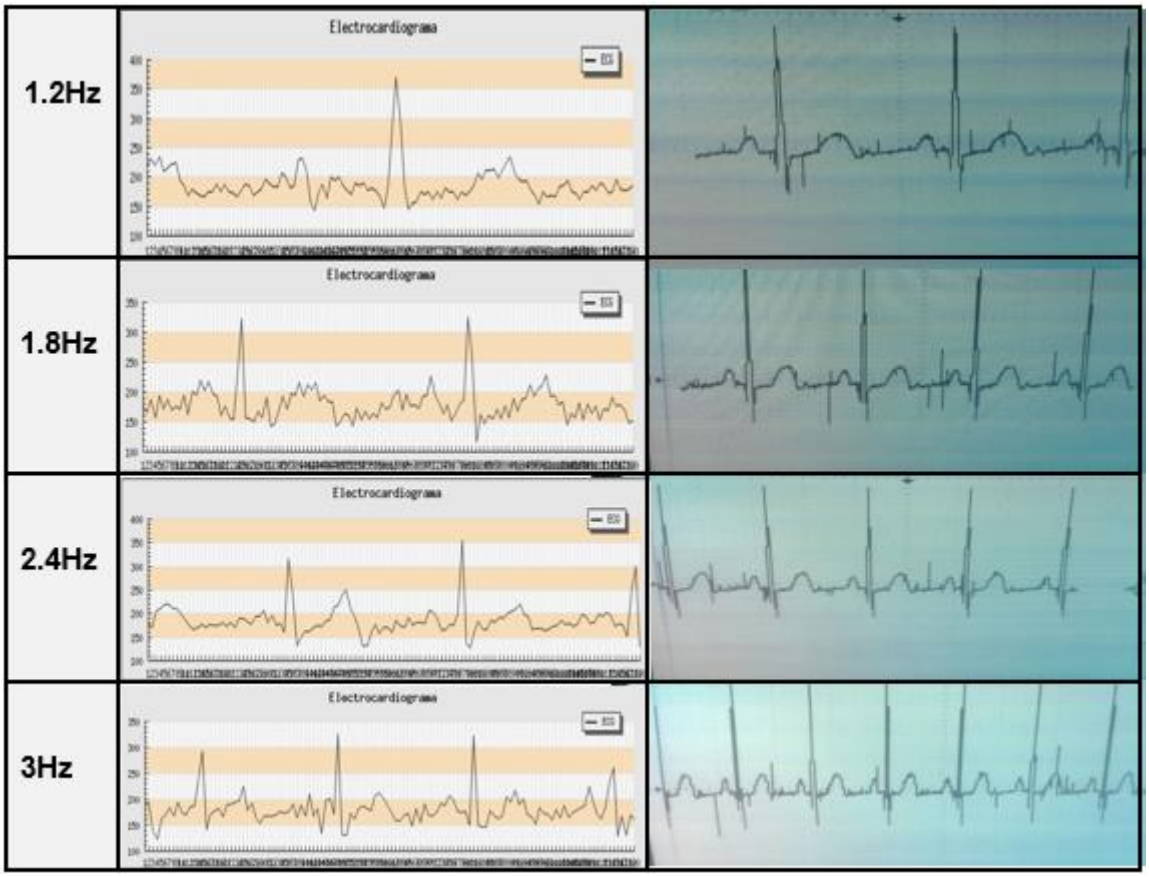

Figura 557: Representaciones gráficas de ECG

En la figura 57, se representan gráficas de ECG para valores de 10, 15, 20 y 25 milisegundos. Estos valores indican que las frecuencias a las que hemos muestreado son de 1.2, 1.8, 2.4 y 3 Hz respectívamente. Para ello, hemos tenido en cuenta que el número máximo de muestras que podemos mandar es de 120.

El intervalo en el que trabajamos estará entre 60 y 130 pulsaciones por minuto, sabiendo que 60 p/m, equivale a 1 p/s.

Con todo esto, observamos que los 50 milisegundos de los que hablamos en capítulos anteriores, no puede ser la elección porque no podríamos ver una señal normal de ECG, ya que la frecuencia sería menor de 1Hz.

Como se puede observar, cuanto mayor es la frecuencia, mayor es el conjunto de ciclos de ECG que se pueden visualizar.

6.3 Pruebas en la conexión WiFi y transmisión de datos

A la hora de realizar la conexión del módulo WiFi hubo que hacer pruebas en relación a los comandos que hay que usar, ya que, en muchos comandos se pueden tomar

distintos valores. Por ejemplo, para que se conecte de forma automática o manual, o para elegir un tipo de protocolo u otro.

Una vez que se conecta a la red, se hicieron pruebas para la transmisión y recepción de datos. Tanto con GET, como con POST, probamos a mandar desde datos simples como un número, hasta con cadenas de caracteres. Sabiendo que el objetivo de los dos métodos es el mismo, pero si queremos mandar una cantidad mayor de datos era mejor con POST.

Además, con el mismo método que se mandaban los datos, teníamos que recibirlos en el servidor y de forma correcta.

Por ejemplo, hacer una transmisión de un dato de temperatura desde Arduino y recibirlo en el servidor,

Wifly.println(GET/temperatura.php?temp=36); //Transmisión

$temperatura = $_GET['temp']; //Recepción

6.4 Pruebas técnicas

En la parte técnica empezamos con la conexión vía USB con la placa Arduino Uno. Para comprobar que todo estaba correcto, el software de Arduino ofrece multitud de ejemplos sencillos. Por ejemplo, encender un led, o simplemente que se encienda una luz de la placa al pulsar una letra, y que se apague al pulsar otra. (Figura 58).

Figura 56: Prueba para encender la luz de la placa

De la misma manera cuando unimos la placa e-Health a la de Arduino Uno y, posteriormente, el módulo WiFi.

En cuanto a sensores, con pruebas simples como captar una temperatura o una toma de pulso y que se muestre en el monitor para comprobar que tomaba los datos de forma correcta.

También para comprobar que encendía un led al tomar un dato concreto.

Con estas pruebas nos dimos cuenta que la placa con la que hemos trabajado sólo dispone de un puerto UART y fue ahí cuando tuvimos que hacer un cambio quitando dos pines y haciendo puente con otros, para que la transmisión de datos se realizara de forma correcta.

Pero, no sólo eso. Finalmente no se pudo conseguir una correcta transmisión de datos entre el cliente y el servidor. Para ello, se optó por usar la placa ***ethernet*** como alternativa. (Figura 59).

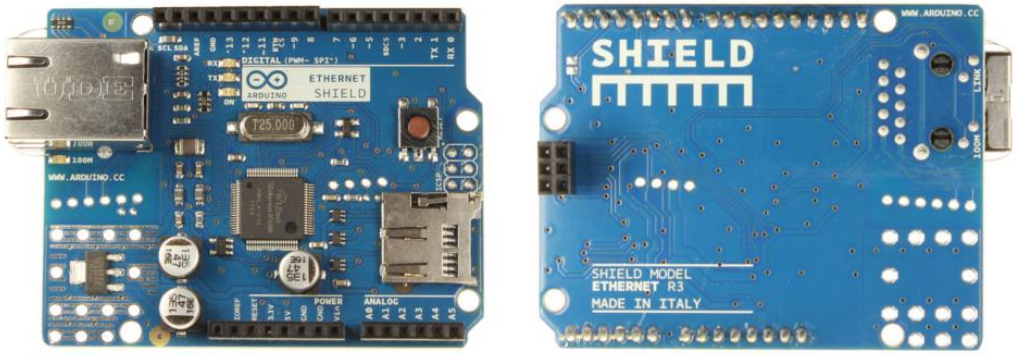

Figura 57: Placa Ethernet

La gran diferencia de lo que habíamos hecho hasta ahora es que la transmisión de datos con esta placa es por cable y no de forma inalámbrica.
Esta placa nos permite conectarnos a Internet y sus principales características son:

·Está basada en el chip W5100.
·Soporta hasta 4 conexiones de sockets simultáneas.
·Usa la librería Ethernet para escribir programas que se conecten a internet usando la Shield.
·Dispone de unos conectores que permiten la unión de otras placas encima.
·Utiliza los pines 10,11, 12 y 13 para comunicarse con el W5100.
·El botón de reset que dispone, resetea ambos, el chip y Arduino.

·Los sketchs se cargan de la misma forma que con la placa Arduino. Una vez subidos, se puede desconectar del ordenador y alimentar la placa con una fuente externa.

Para realizar la conexión, hemos usado un cable cruzado entre el dispositivo y el ordenador.

El código programado en arduino está diseñado para poder trabajar con módulo WiFi y con módulo Ethernet, como se muestra en la figura 60.

```
/* configuración del hardware ************************************************/

#define A_UNO    /* si usamos como base un arduino uno */
//#define A_LEO    /* si usamos como base un arduino leonardo */

#define C_WIFI  /* si la conxión remota se realiza usando un modudo wifi */
#define C_ETH   /* si la conexión remota se realiza usando un modulo ethernet */
```

Figura 58: Configuración del hardware en el IDE de arduino

Según el que queremos usar, se comenta uno u otro. De esta forma, sólo se ejecutarán unas funciones y órdenes si trabajamos con el módulo WiFi y otras, si trabajamos con módulo Ethernet.

La configuración del nuevo módulo se muestra en la figura 61, donde se establecen los parámetros de forma distinta a como se hacía con el módulo WiFi y se crea un servidor y un cliente, con los que se trabaja en el programa para realizar la comunicación y la transmisión de datos.

```
#ifdef C_ETH
//Parametros asociados a ethernet
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 150, 214, 109, 232 };
byte _dns[] = { 150, 214, 57, 7 };
byte gw[] = { 150, 214, 109, 254 };
byte sn[] = { 255, 255, 255, 192 };

//Creación de servidor
EthernetServer * server = NULL ;
//Creación de cliente
EthernetClient cliente ;
```

Figura 59: Configuración Ethernet

Al final, realizando pruebas con esta alternativa, pudimos comprobar que la transmisión se realizaba correctamente, y que, creemos que el problema reside en los conflictos que da la librería asociada al módulo WiFi.

El resultado final es la unión de la placa Arduino Uno, la placa Ethernet y la placa e-Health junto con la unión de una protoboard con LED. (Figura 61).

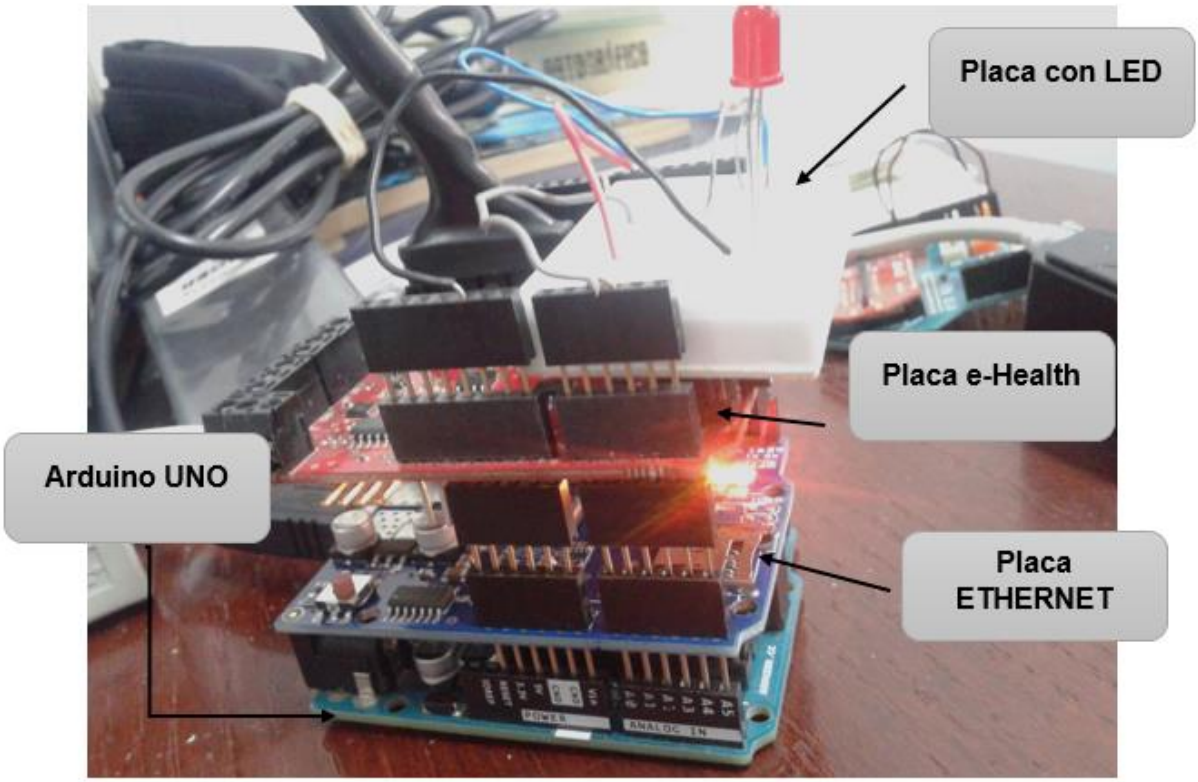

Figura 60: Dispositivo final

7. Conclusiones.

En este Trabajo de Fin de Grado:

·Se ha implementado un sistema de adquisición y registro de datos biomédicos, formado por Arduino Uno, plataforma e-Health con sus respectivos sensores, la placa de comunicación junto con el módulo WiFi Roving RN-XV 171.

·Se ha implementado un sistema de gestión y recepción de datos. Para ello se ha usado el servidor web Apache en el que se ha alojado una base de datos y la página web.

·Se ha conseguido establecer un protocolo de intercambio de información desde el microcontrolador de Arduino hasta el servidor web mediante comunicación WiFi, donde la iniciativa en la comunicación y transmisión de la información la lleva siempre acabo el operador del sistema centralizado.

·La conexión a internet se ha realizado por cable con el uso del módulo Ethernet como alternativa a la inicialmente planteada mediante WiFi.

·Se ha conseguido una correcta recepción de datos enviados desde Arduino al servidor.

·Se ha conseguido un correcto funcionamiento de la página web para poder visualizar los datos.

·Se ha creado una base de datos donde almacenar toda la información.

8. Bibliografía.

[1] Sociedad Internacional para la telemedicina y eSalud. http://www.isfteh.org/working_groups/category/e_hispanic

[2] F.Sebastian et ad.; Real Time E-health System for Continuous Care; Proceedings of the 8th International Conference on Pervasive Computing Technologies for Healthcare, REHAB ICST; pp. 436-439; Brussels, Belgium, 2014 DOI: 10.4108/icst.pervasivehealth.2014.255308.

[3] I.Orha, S.Oniga; Wearable sensors network for health monitoring using e-Health platform; Carpathian Journal of Electronic and Computer Engineering 7/1 (2014) 25-29

[4] H. Cervantes de Ávila y otros; Arquitectura de e-Salud basada en redes inalámbricas de sensores; Revista de Divulgación Científica; Vol. 6, No.2, septiembre-diciembre 2012; ISSN 2007-3585.

[5] W. Chen et ad. ; Wireless Transmission Design for Health Monitoring at Neonatal Intensive Care Units; Procceding of the 2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies, 2009. ISABEL 2009.

[6] E. Shanko and M. G. Papoutsidakis; Real Time Health Monitoring and Wireless Transmission: A Controller Application to Improve Human Medical Needs; The 4th IEEE International Conference on E-Health and Bioengineering - EHB 2013.

[7] Arduino: https://www.arduino.cc/en/Guide/HomePage

[8] Tutoriales plataforma e-Health, disponibles on-line en:

https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical

[9] Librerias Wifi para Arduino, disponibles on-line en: https://www.arduino.cc/en/Reference/WiFi

[10] Guía de comunicaciones Wifi con arduino, disponible on-line en: http://www.ismsolar.com/blog/la-guia-definitiva-de-comunicaciones-wifi-con-el-arduino-wireless-sd-shield-y-wifly-rn-xv

[11] Roving RN-XVee datasheet, disponible on–line en:

http://www.cooking-hacks.com/skin/frontend/default/cooking/pdf/WiFly-RN-XV-DS.pdf

[12] Roving RN-XVee manual, disponible on–line en:

http://www.cooking-hacks.com/skin/frontend/default/cooking/pdf/WiFly-RN-UM.pdf

[13] Learning PHP, MySQL, and JavaScript, CSS & HTML5. Robin Nixon. Sebastopol, Calif., O'Reilly, cop. 2014

[14] The Essential Guide to HTML5 and CSS3 Web Design. Craig Grannell, Victor S., Berkeley, CA, Apress 2012.

[15] Diseño y Administración de Bases de Datos. Gary W. Hansen. Ed. Prentice Hall. 1997, 2ª Ed.

[16] Tutorial (Sociedad) de formularios para creación de páginas web:

http://www.tutorialhtml.net/manualHTML/formularios-adjuntos.php

9. Apéndices.

Apéndice A: Tecnologías implicadas

Elementos software

-IDE arduino y librerías de comunicaciones.

El software está formado por un entorno de desarrollo IDE basado en una parte de procesado y el lenguaje de programación wiring, además del “bootloader” que se ejecuta en la placa y que es el cargador de arranque de Arduino.
En la placa hay un microcontrolador que se programa mediante un ordenador que hace uso de comunicación serial, mediante un convertidor de niveles RS-232 a TTL serial, es decir, que sirve para transmisión y recepción de señales.

Es el entorno de desarrollo principal que hemos usado para trabajar, ya que, es un software que hace que sea fácil escribir código y subirlo a la placa y se puede ejecutar tanto en Windows, Mac OS y Linux.

El entorno que utiliza es java y está basado en procesamiento y otros softwares de código abierto, que además, se puede utilizar con cualquier placa Arduino.

En cuanto a librerías de comunicaciones, se refiere a determinadas librerías que se han tenido que instalar para poder usar funciones y ciertos recursos que éstas ofrecen. Las librerías que hemos usado son:

·Wifly. Esta librería se encuentra alojada en Github y es una de las mejores librerías que existen. Además dispone de una buena documentación, así como muchos ejemplos que nos ayudan a aprender a utilizar esta librería de manera rápida y sencilla.
Existen muchos ramas o versiones de esta librería donde programadores han dotado de mayor potencia y añadido nuevas funcionalidades a la misma.

·NewSoftwareSerial, con esta librería habilitamos unos pines diferentes de los pines que vienen por defecto como *tx* y *rx,* que son los pines que se utilizan para la transmisión y recepción de datos. Como argumentos de entrada del objeto de la clase *SoftwareSerial* hay que escribir los pines a los que se conectan los cables.

·PinChallenge está incluida dentro de la librería eHealth y se utiliza sólo cuando usamos el sensor pulsioxímetro.

-Librerías Arduino para Plataforma e-Health V2.0

La plataforma e-Health nos permite el uso de diferentes sensores pero requiere la instalación de la librería e-Health correspondientes que incluye todos los archivos necesarios en dos carpetas separadas “eHealth” y “PinChangeInt”, mendionada anteriormente.

De esta forma se pueden leer fácilmente todos los sensores y enviar la información mediante el uso de cualquiera de las interfaces disponibles. Posee un sistema de código abierto y fácil de usar.

-PHP, JavaScript, HTML como herramientas de desarrollo web.

PHP es un lenguaje de código abierto usado para el desarrollo web y que se puede utilizar también, incrustado en HTML.

Para poder trabajar con código PHP es necesario que éste, esté encerrado entre las etiquetas de comienzo y final <?php y ?>, que nos van a permitir salir y entrar del modo PHP. Además, contiene numerosas funciones que nos permitirán hacer acciones determinadas.

El código PHP se ejecuta en el servidor, generando HTML y enviándolo al cliente.
El cliente ejecuta el archivo y obtendrá un resultado pero sin ver qué código es el que permite realizar esa acción.

La principal ventaja de usar este lenguaje es que puede ser tan simple y fácil de usar para un principiante, como complejo y con muchas características por ofrecer a u programador profesional.

Como en este caso, el servidor lo encontramos en la persona que solicita un dato desde la página web, accediendo a la base de datos y que, por tanto, mediante lenguaje PHP se hacen los programas necesarios para que el intercambio de información se lleve a cabo de forma correcta.

Por ejemplo, programas como, consultar el listado de pacientes, y cualquier interacción con ellos, desde añadir un nuevo paciente hasta eliminarlo porque se haya dado de alta. Y, otros, como tener un acceso a la página web con un usuario y contraseña determinados.

-APACHE. Servidor HTTP.

El servidor Apache es un servidor HTTP que permite la creación de páginas y servicios web. Es gratuito, robusto y genera mucha seguridad y rendimiento.

Para entender mejor lo que es Apache, vamos a definir lo que es un servidor web. Servidor web es un programa diseñado especialmente para transferir datos de hipertexto, es decir, páginas web con todos sus elementos (textos, imágenes, gráficos, etc). Estos servidores web están instalados en un ordenador, con conexión a internet y que utilizan el protocolo http. La conexión a internet es imprescindible, ya que, está esperando a que un navegador haga una petición, ya sea acceder a una página u otro tipo de petición, y una vez recibida, responde a esa petición con código HTML.

Al escoger y trabajar con este servidor podemos destacar ventajas e inconvenientes.

Ventajas:

· Software de código abierto.
· El coste del servidor web Apache es completamente gratuito.
· Es muy popular y usado por muchos programadores profesionales, lo que implica que se esté mejorando y actualizando constantemente.
· Se puede instalar en muchos sistemas operativos, es compatible con Windows, Linux y MacOS. Es un software multiplataforma.
·Rendimiento alto.
·Soporte de seguridad SSL y TLS.

Inconvenientes:

·Falta de integración
·Posee formatos de configuración NO estándar.
·No posee un buen panel de configuración

-SQLite como gestor de bases de datos.

El programa que se ha usado en este proyecto para la gestión de la base de datos es sqlite3.

SQLite es una herramienta de software libre creada por D. Richard Hip en el año 2000. Proporciona líneas de comandos de fácil utilidad, sin tipos de variables precisos y permitiendo al usuario introducir y ejecutar instrucciones SQL en una base de datos de forma manual.

Una de sus principales características es su fácil portabilidad debido a su compatibilidad del 100 % entre las diversas plataformas, por lo que no hay que realizar procesos complejos de importación y exportación de datos.

Otras de sus principales características son las siguientes:

- Está construida en C.
- La base de datos completa se encuentra en un solo archivo.
- Cuenta con librerías de acceso para muchos lenguajes de programación.
- Soporta texto en formato UTF-8 y UTF-16.
- Soportar desde las funciones más básicas hasta las más complejas del lenguaje SQL
- El código fuente es de dominio público.
- En su versión 3, SQLite permite base de datos de hasta 2 Terabyte de tamaño.

Esta base de datos consta de una tabla para almacenar los pacientes que incorporará variables para guardar datos de cada paciente, así como, nombre, edad, id asignado a ese paciente, id del arduino que se le asigne, ciudad de residencia o número de seguridad social, entre otros.
En un principio, se organizó la base de datos estableciendo una tabla para cada sensor, de manera que cada una de ellas almacenaría la información relativa a la toma de las medidas correspondientes. Pero el inconveniente era que, al tener como una de las variables el flag, cada vez que se quería consultar el flag teníamos que acceder a un php independiente creado para cada flag y nos mostraba toda la información.
Es decir, cuando se solicitaba una nueva medida, lo primero que se hacía era actualizar el valor del flag a 1, para indicar así que se podía llevar a cabo la toma de datos. Según la medida que queríamos, se actualizaba el flag de la tabla del sensor correspondiente.

Así que, el cambio fue hacer la consulta del flag en un único php y para ello, debíamos tener el flag en una única tabla. Por tanto, la nueva base de datos constaba de una tabla PacienteArduino cuya finalidad era almacenar la información de cada paciente en relación con el arduino y el sensor asignado, es decir, esta tabla contiene el id del paciente, el id del arduino que se haya asignado y las variables para consultar el flag de temperatura, el de pulso y el de ECG. También tenemos tablas para guardar los datos del sensor de temperatura, de pulso y de ECG, que serían PacienteDatosTemp, PacienteDatosPulso y PacienteDatosECG, respectivamente.
Y por último, una tabla Pacientes, para guardar los datos de cada paciente, así como, el nombre, la edad, el id del paciente, dirección, etc.

Elementos hardware

-Ordenador Pc o compatible.

El ordenador y sistema operativo con el que se ha trabajado es Windows, pero sin problema se puede trabajar con Linux o Mac OS, ya que el entorno de desarrollo del proyecto es accesible también a ellos.

-Arduino

Arduino es una compañía de hardware libre, la cual desarrolla placas de desarrollo que integran un microcontrolador y un entorno de desarrollo (IDE), diseñado para facilitar el uso de la electrónica en proyectos multidisciplinarios.

Arduino es una compañía que desarrolla placas de hardware libre y que integran un microcontrolador y un entorno de desarrollo IDE, como se ha comentado en un punto anterior.

El microcontrolador está integrado por un circuito que se alimenta con un voltaje de +5 voltios, posee multiplicadores de voltaje y condensadores que permiten la implementación de puertos serie en dispositivos que tengan esta alimentación de 5 voltios.

La placa, además del microcontrolador, posee puertos digitales y analógicos de entrada y de salida que pueden conectarse a otras placas para extender el funcionamiento de ésta.

Las especificaciones de la placa Arduino Uno son las siguientes:

Especificaciones técnicas	
Microcontrolador	ATmega328P
Voltaje Operativo	5V
Voltaje Entrada (recomendado)	7-12V
Voltaje Entrada (límite)	6-20V
Pines E/S digitales	14 (6 proporcionan salida PWM)
Pines Entrada analógica	6
Corriente por pin E/S	20 mA
Corriente pin 3.3V	50 mA
Memoria Flash	32 KB (0,5KB reservados para el bootloader)
SRAM	2 KB
EEPROM	1 KB
Velocidad reloj	16 MHz
Peso	25 g
Dimensiones	68.6 mm x 53.4 mm

Tabla 2: Especificaciones técnicas de Arduino

Y a continuación podemos ver un esquema de la placa en la figura 63.

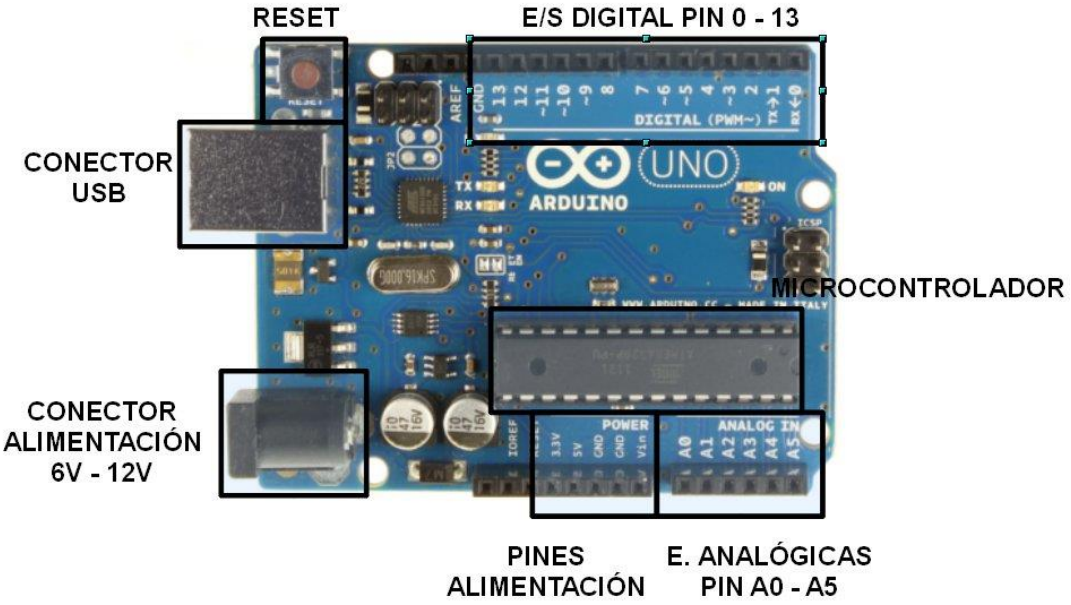

Figura 61: Esquema de la placa Arduino

Como se puede ver en la imagen, dispone de un botón RESET, que nos permitirá resetear la placa y por tanto, el programa que esté cargado en ella y así indicarle que comience de nuevo.

El conector de alimentación que nos permitirá prescindir de la conexión por cable a un ordenador, que se lleva a cabo por le entrada USB que dispone.

-Plataforma e-Health V2.0 para arduino y Raspberry-pi

La placa e-Salud permite a los usuarios de Arduino y Rasperry Pi llevar a cabo aplicaciones médicas donde el monitoreo del cuerpo es necesario mediante el uso de 10 sensores diferentes: pulso , de oxígeno en la sangre (SpO2) , el flujo de aire (respiración) , la temperatura corporal , electrocardiograma (ECG) , glucómetro , la respuesta galvánica de la piel (GSR - sudoración) , la presión arterial (esfigmomanómetro) , la posición del paciente (acelerómetro) y electromiografía (EMG) .

Esta información se puede utilizar para monitorizar en tiempo real el estado de un paciente o para obtener datos sensibles con el fin de ser analizados posteriormente para el diagnóstico médico. La información de los datos se puede enviar de forma inalámbrica utilizando cualquiera de las opciones de conectividad disponibles: Wi-Fi , 3G , GPRS , Bluetooth , ZigBee 802.15.4 y dependiendo de la aplicación.

Si se necesita el diagnóstico por imagen en tiempo real de una cámara se puede conectar al módulo 3G con el fin de enviar fotos y vídeos del paciente a un centro de diagnóstico médico.

En este proyecto los sensores que se han utilizado son el pulsioxímetro, el sensor de temperatura y el sensor de electrocardiografía, y el tipo de conectividad es vía WiFi.

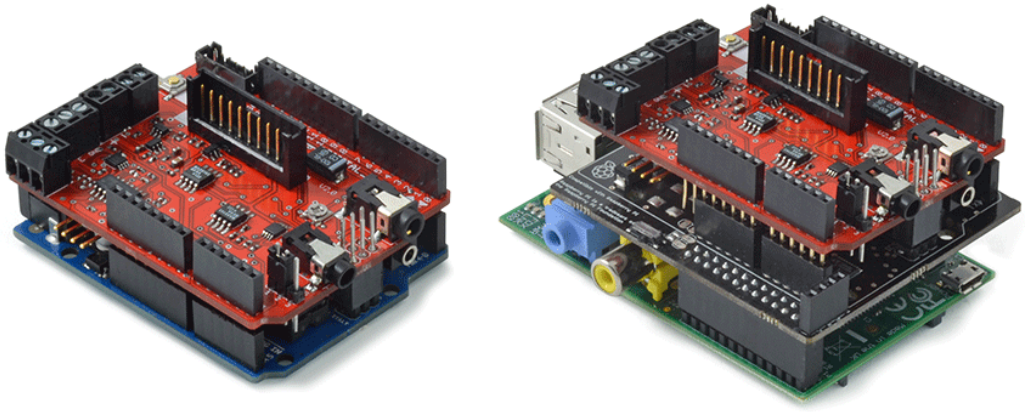

Figura 62: Placa e-Health conectada a Arduino

En la imagen 64, se ve la placa de salud conectada con Arduino para trabajar con él.

- Kit de sensores para medida biomédicas para Plataforma e-Health V2.0.

Sensor de temperatura

Este sensor permite medir la temperatura corporal de un paciente. Hoy en día, una gran cantidad de enfermedades están acompañadas por cambios característicos en la temperatura corporal, por lo tanto, sería de gran utilidad trabajar con él en las condiciones que se especifican a lo largo del documento.

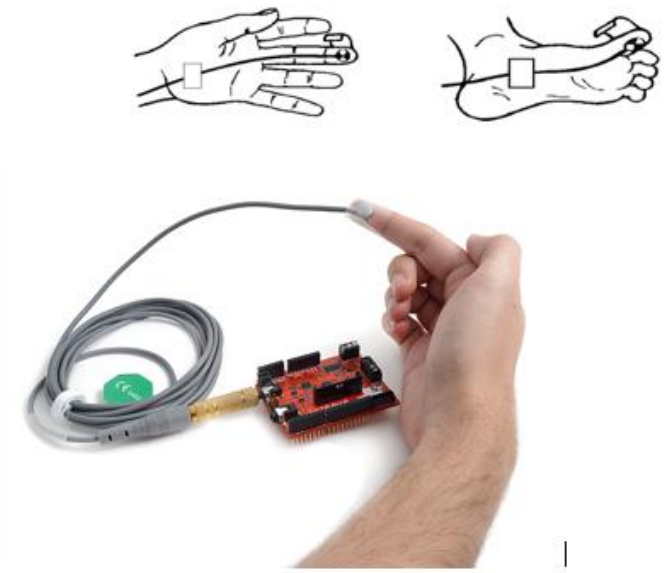

Figura 63: Sensor de temperatura

En la imagen se muestra la realidad del uso de un sensor de temperatura, que se pega en la yema del dedo por la que circulan numerosos vasos y se conecta a la placa, que nos permitirá usar la función determinada para calcular la temperatura corporal.

Los rangos son los siguientes:

·**Hipotermia** <35,0 º C (95,0 º F)

·**Normal** 36,5 a 37,5 º C (97,7 – 99,5 º F)

·**Fiebre o hipertermia** >37,5 a 38,3 º C (99,5 a 100,9 º F)

·**Hiperpirexia** >40,0 a 41,5 º C (104 a 106,7 º F)

Cuando se utiliza el sensor de temperatura, lo que se está midiendo en realidad es una tensión, por cada tensión se obtiene una correspondencia de temperatura, por lo que es muy importante la calibración de este sensor para que mida correctamente el nivel de temperatura corporal.

El circuito asociado a este sensor es el siguiente:

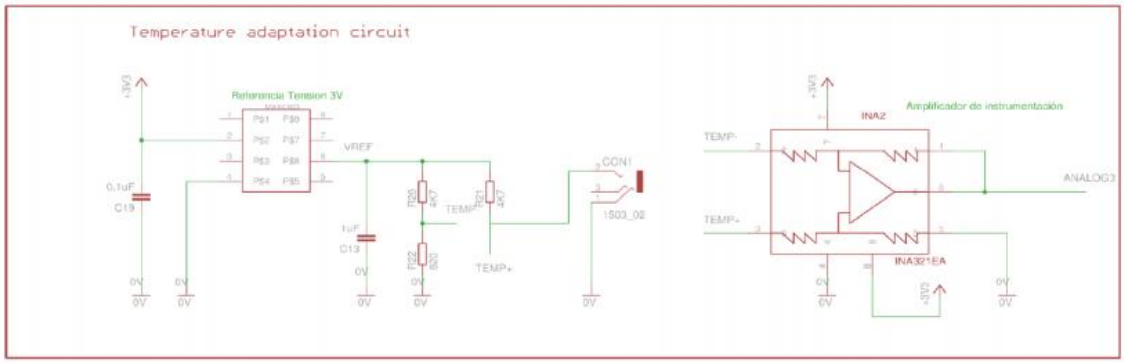

Figura 64: Circuito del funcionamiento del sensor de temperatura

El diseño para el sensor de temperatura se basa en un puente de Wheatstone. El puente de Wheatstone ha sido diseñado para cubrir el rango de temperaturas de interés: entre 25ºC y 50ºC. La tensión de salida diferencial del puente de Wheatstone es amplificada y referenciada a tierra mediante un amplificador de instrumentación. Este sensor se comunica mediante el pin analógico 3 de Arduino.

Sensor de pulso

Este sensor mide la cantidad de oxígeno que hay en la sangre y el recuento de pulsaciones por minuto al que late el corazón.

La saturación de oxígeno se define como la cantidad de oxígeno que hay disuelto en la sangre, sobre la base de la detección de hemoglobina (Hb) y desoxihemoglobina (HbO2).

El funcionamiento es detectar la hemoglobina y desoxihemoglobina. Utiliza dos longitudes de onda de luz diferentes para medir la diferencia real en los espectros de absorción de HbO2 y Hb. La circulación sanguínea se ve afectada por la concentración de Hb y HbO2 en sangre, y sus coeficientes de absorción se miden usando dos longitudes de onda, 660 nm (espectros de luz roja) y 940 nm (espectros de luz infrarroja) . Hemoglobina desoxigenada y oxigenada se encargan de absorber las diferentes longitudes de onda.

Figura 65: Sensor de pulsioximetría

En la imagen 67, se observa que, introduciendo el dedo índice en el pulsioxímetro y pulsando el botón de inicio, a través de luz infraroja, se detecta la hemoglobina en la sangre y muestra en la pantalla los resultados de saturación y pulso.

Este tipo de sensores son útiles cuando se posee un paciente con nivel de oxígeno en sangre inestable, debido a enfermedades crónicas. Este tipo de enfermedades requieren un control de los niveles de oxígeno para comprobar si se necesita suplemento de oxígeno.

Los rangos normales son entre 95 y 99 %.

Sensor de ECG

El electrocardiograma es una herramienta de diagnóstico que se utiliza para evaluar las funciones eléctricas y musculares del corazón. El electrocardiograma (ECG) es uno de los exámenes médicos más usados actualmente.

Se utiliza para diagnosticar patologías cardíacas como isquemia o infarto, siendo esto de gran utilidad para los médicos.

El valor que se obtiene es tensión medida en voltios.

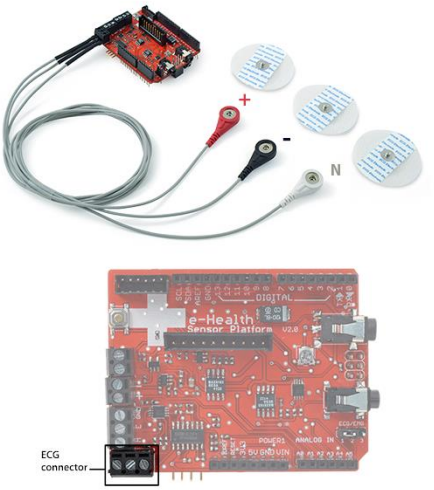

Figura 66: Sensor de ECG

Como se observa, el sensor dispone de tres polos, positivo(rojo), negativo(negro) y neutro(blanco), cada uno de ellos conectado a la placa en el lugar correspondiente.

A continuación, se muestra una gráfica de lo que sería un ECG normal, con intervalos P y T positivos, picos Q y S negativos y un pico característico R. Cada uno de ellos tiene una duración y voltaje determinados para que se pueda analizar y especificar si hay alguna anomalía o está correcto.

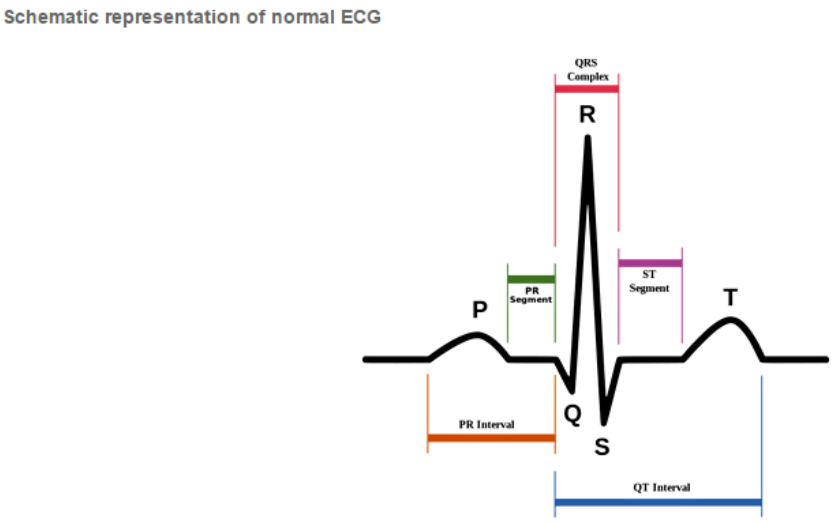

Figura 67: Curva característica de ECG

El circuito correspondiente:

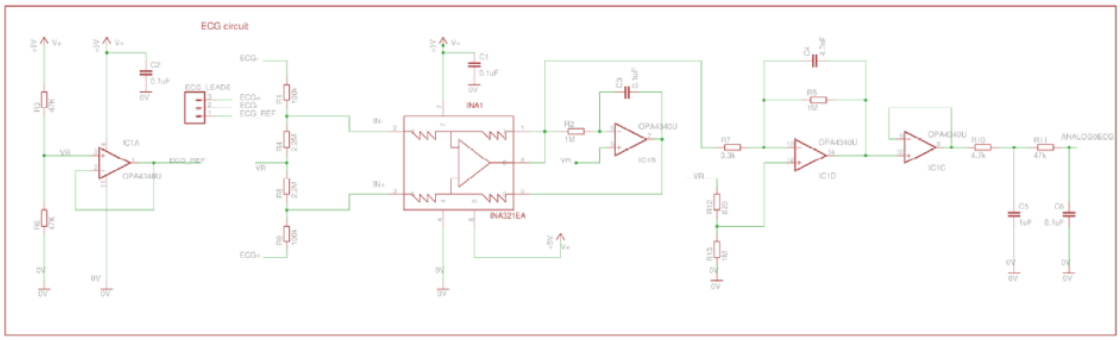

Figura 68: Circuito del funcionamiento de ECG

La electrónica de adaptación necesaria para el electrocardiograma, ha sido basada en las especificaciones técnicas de un amplificador de instrumentación. Este dispositivo es el encargado de amplificar la señal diferencial de entrada proveniente de los electrodos izquierdo y derecho. Se utilizan amplificadores operacionales para completar las distintas etapas necesarias para la medición de la señal de ECG. Este sensor utiliza el pin analógico 0 de Arduino para la comunicación.

-"Communication Shield" para Arduino

La placa "Communication Shield" dispone de un pequeño interruptor con dos posiciones disponibles: una llamada XBEE y otra llamada USB. El esquema que muestra la diferencia entre una configuración u otra se muestra a continuación.

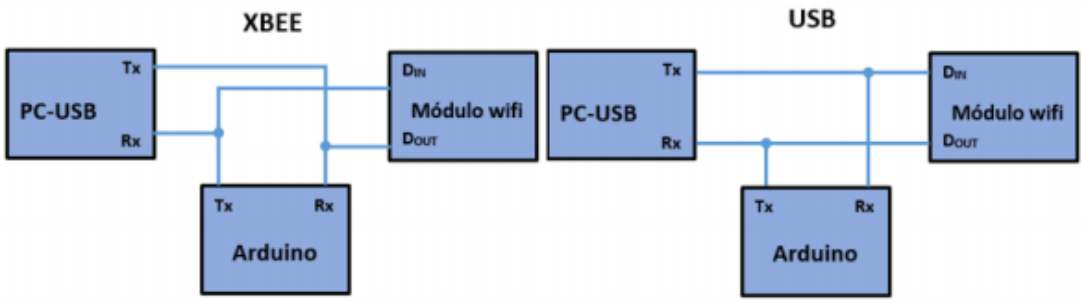

Figura 69: Posiciones que ofrece la placa de comunicación

En la posición XBEE, la salida del módulo WiFi está conectada al pin de escucha del microcontrolador y la entrada del WiFi a la salida del micro, pero hay que tener en cuenta que los pines comunicación del micro siguen conectados al puerto de comunicaciones del Arduino y este al ordenador.

En XBEE no podemos subir ni compilar ningún sketch con código.

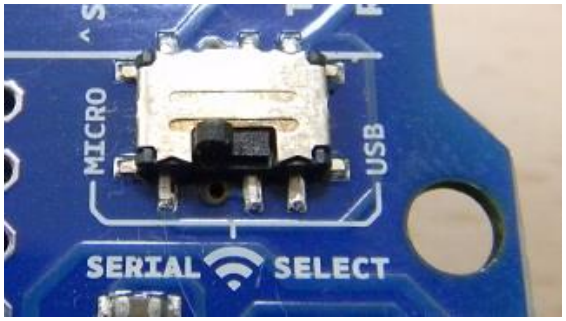

Figura 70: Modo de funcionamiento XBEE

Para poder cargar el código del programa hay que usar el modo USB, donde el módulo WiFi estará conectado al puerto USB, es decir, el módulo se comunica directamente con nuestro ordenador. Una vez que el programa ha sido cargado en Arduino, se puede cambiar la posición del interruptor y ponerlo en XBEE.

Figura 71: Modo de funcionamiento en USB

La solución final fue deshabilitar la conexión física del puerto de comunicaciones entre la placa “Communication Shield” y Arduino, y se creó un puente o canal serie entre el micro y el módulo WiFi.

De esta manera el puerto serie se comunica con el Arduino y el Arduino se comunica con el módulo WiFi y no es necesario estar cambiando el interruptor. Esta comunicación se realiza por otro puerto serie diferente por lo que podemos usar el puerto serie nativo para tareas de depuración.

Lo que hacemos es quitar los pines 0 y 1 de nuestra placa para inhabilitar la comunicación serie y que no entren en contacto con Arduino y creamos un canal virtual serie entre los pines 8 y 9 de Arduino. Para ello, usaremos la librería SoftwareSerial, ya mencionada anteriormente. El interruptor debe estar en la posición XBEE porque

lo que se quiere conseguir es que el módulo WiFi se comunique con el microprocesador para intercambiar comandos.

De este modo Arduino dispondrá de dos puertos serie: el nativo USB que usaremos para programarlo y para enviar información de depuración y el puerto virtual para la comunicación entre el microprocesador y el módulo WiFi.

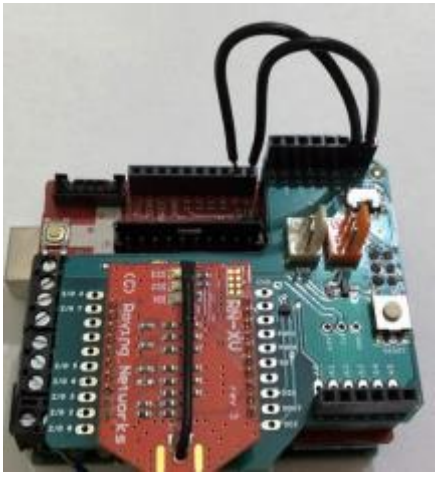

Figura 72: Dispositivo Arduino con la apertura del puerto virtual

Una vez que está listo la configuración del hardware, creamos en Arduino el puerto virtual con las siguientes líneas de comandos:

```
#include <SoftwareSerial.h>
SoftwareSerial wifiSerial(8,9);
```

Figura 73: Creación del puerto virtual en el código de Arduino

Módulo WiFi para Arduino: "Roving RN-XVee"

El módulo "Roving RN-XVee" es el modelo que hemos usado.

El módulo RN-XV integra un RN-171 Wi-Fi, un radio 802.11 b/g, un procesador de 32 bits, un stack TCP/IP, un reloj de tiempo real, un crypto acelerador, una unidad de manejo de potencia y una interfaz análoga. El módulo tiene pre-grabado un firmware que simplifica la integración y minimiza el tiempo de desarrollo teniendo una simple configuración de hardware donde solo se tienen cuatro conexiones PWR, TX, RX y GND.

Características

·Potencia: 0dBm a 12dBm
·Hardware interfaces: TTL UART
·Host data rate: hasta 464Kbps (UART)
·Puertos: 8 GPIO y 3 análogos
·Voltaje: 3.3VDC
·Antena: tipo cable

Figura 74: Módulo WiFi

Antes de empezar a programar, hay que saber que el módulo WiFi puede estar en dos estados: modo comandos o en modo de datos. En el modo de datos el módulo WiFi está preparado para iniciar conexiones externas o aceptar conexiones entrantes. Sin embargo para configurar el módulo hay que estar en modo comandos.

Por defecto el dispositivo estará en modo de datos. Para entrar al modo comandos hay que enviar la siguiente secuencia de tres caracteres: “$$$” y el dispositivo responderá enviando “CMD” indiciando que está en modo Comandos.

Una vez que se está en este modo, es posible enviar secuencias de comandos para configurar el módulo. El dispositivo aceptará bytes ASCII como comandos. Cada comando terminará con un retorno de carro. Para salir del modo comando, se envía: “exit”. El dispositivo responderá con “EXIT” indicando que este ha salido correctamente del modo comando y cambiado a modo Datos. Para la mayoría de los comandos que se envíen, el dispositivo responderá con un “AOK” si son válidos y aceptados o con un “EER” si son inválidos o ha habido algún error.

La sintaxis de los comandos es muy sencilla. Cada comando está compuesto por una palabra clave y varios parámetros adicionales separados por espacios. Los comandos se pueden clasificar en 5 tipos:

- Comandos de Ingreso (SET): modifica los parámetros del módulo. Tienen efecto de inmediato y para que sean permanentes hay que guardarlos.

- Comandos de Petición (GET): Muestra información sobre el módulo y sus parámetros.

- Comandos de Estado (STATUS): Ve que está pasando con la interfaz, estado de IP, etc.

- Comandos de Acción (ACTION): Permite acciones tales como escanear, conectar, desconectar, etc.

- Comandos de Entrada/Salida de Archivos (FILE IO): Actualiza, carga y guarda configuraciones, borra archivos, etc.

Para que las modificaciones sean permanentes hay que enviar el comando “save” para guardarlo, sino el módulo cargará las configuraciones que tenía antes de encender el dispositivo.

Lo primero que se llevó a cabo fue la configuración del módulo wifly que podemos observar a continuación.

-Para habilita o deshabilitar el modo DHCP (Palabra para definir)
set ip dhcp 1

-Configuración del protocolo. Si ponemos valor 1 estamos configurando el protocolo TCP. Si queremos protocolo HTTP, corresponde con el valor 4. Y el 0 corresponde con el protocolo UDP.
set ip protocol 1

-Configuración para asociarse a una red de forma automática o manual. Se pueden establecer cuatro valores. El 0 es una conexión de forma manual, el 1 trata de unirse al punto de acceso que coincida con el SSID, la clave de acceso y el canal almacenados. El 2 trata de unirse al punto de acceso con seguridad que coincida con el modo de autenticación almacenado. Esto ignora el SSID y busca el punto de acceso con la señal más fuerte. El 3 es una opción reservada, no se usa. Y, por último, el 4 corresponde con el modo adhoc, es decir, crea una red adhoc usando el SSID, la IP y la máscara de red. El canal debe ser ajustado.
set wlan join 1

-Configura los password para los modos de seguridad WAP y WAP2. Tiene de 1-64 caracteres. El Password puede ser alfa numéricos, y este es usado junto con el SSID para generar un único valor de 32 bits pre-combinado (PSK), el cual desarrollará en un número de 256 bits. Para contraseñas que contienen espacios se reemplaza con el carácter $.
set wlan phrase <string>

-Adiciona una red <ssid>. Si la red tiene la seguridad habilitada se deberá configurar la contraseña con el comando set wlan phrase previo a la emisión del comando join.
join <SSID

-Configura la dirección IP, en este caso es 192.168.43.235
set ip host <IP address>

-Configura la dirección IP de la puerta de enlace, si DHCP está activado, la dirección IP de la puerta de enlace es asignada y sobre-escrita durante la asociación con el punto de acceso. La puerta de enlace es 192.168.43.1
set ip gateway <addr>

-Configura el número de puerto que el host está escuchando. En este caso es el puerto 80.
set ip remote <port number>

-Configura el String que es enviado al cliente remoto TCP cuando se abre el puerto TCP. Si no es el String deseado, usa el “0” como el parámetro <String>. La longitud máxima del String es de 32 caracteres. Por default es HELLO.
set comm remote <string>

-Configuraciones para el Cliente/Servidor Web HTTP, el valor es un registro de bits combinados. El valor 0 envía automáticamente la cabecera HTML. El 1 envía a los usuarios de datos binarios. El valor 2 prueba los pines DPIO y ADC. El valor 3 anexa &id =<value>, donde el valor es el ID de dispositivo. Y el valor 4 se usa para anexar los pares de valores claves para el mensaje HTTP.
Nosotros usamos el valor 0.
set opt format <value>

-El comando para abrir la conexión después de la configuración anterior sería Open.
Open

-Para guardar la configuración
Save

Router WiFi

Para llevar a cabo el proyecto era necesario disponer de una red a la que conectarse. Ya sea un router particular o usando el móvil como router, como se ha trabajado en este proyecto.

Los únicos datos que hay que usar son el nombre de la red y la contraseña, para que el módulo pueda acceder y conectarse a la red, y una vez configurado, el módulo detectará y se podrá conectar de forma automática a la red indicada.

Apéndice B: Código completo de Arduino

```
/* Registro de medidas fisiologicas remoto bajo demanda */


#define A_UNO
//#define A_LEO

//#define C_WIFI
#define C_ETH

#if defined( A_UNO ) && defined( A_LEO )
#undef A_LEO
#endif

#if defined( C_WIFI ) && defined( C_ETH )
#undef C_ETH
#endif

#if defined( A_UNO ) && defined( C_WIFI )
#warning descomenta la siguiente linea si esta comentada
//#include <SoftwareSerial.h> // Comentar en caso de conflicto con
PinChangeInt
#endif
#ifdef C_WIFI
#include <WiFlyHQ.h>
#endif

#ifdef C_ETH
#include <SPI.h>
#include <EthernetClient.h>
#ifndef C_ETH2
#include <Dns.h>
#include <Dhcp.h>
#include <EthernetServer.h>
#include <EthernetUdp.h>
#include <Ethernet.h>
#endif // C_ETH2
#endif // C_ETH


/* Macros para compilar de forma opcional los modulos
 de control de los distintos sensores */

#define CONTEMP     /* Para usar el sensor de temperatura */
#define CONECG      /* El del elctrocardiografo */
#define CONPULSO    /* El del pulsioximetro */

#if defined( CONTEMP ) ||  defined( CONECG ) ||  defined( CONPULSO )
#include <eHealth.h>
#endif

#ifdef CONPULSO
#define ledPulso 5
#include <PinChangeInt.h>
```

```
#endif

#ifdef CONECG
#include <TimerOne.h>
#endif

#if defined( A_UNO ) && defined( C_WIFI )
SoftwareSerial wifiSerial(8, 9);
#endif
#if defined( A_LEO ) && defined( C_WIFI )
#define wifiSerial Serial1
#endif


#ifdef C_WIFI
WiFly wifly;
#define RedServicio wifly
#define RedCliente wifly
#endif

//Configuración del módulo Ethernet
#ifdef C_ETH
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 150, 214, 109, 232 };
byte _dns[] = { 150, 214, 57, 7 };
byte gw[] = { 150, 214, 109, 254 };
byte sn[] = { 255, 255, 255, 192 };

class tRed : public EthernetClient {
public :
    void flushRx( ) {
        flush( ) ;
    }
    void sendChunkln( const __FlashStringHelper * text ) {
        println( text ) ;
    }

    void sendChunkln( ) {
        println( ) ;
    }
    void sendChunk( const __FlashStringHelper * text ) {
        print( text ) ;
    }
    void sendChunk( const char * text ) {
        print( text ) ;
    }
    void close( ) { stop( ) ; }

    bool open( const char * add , unsigned puerto ) { return
connect(  add , puerto ) ; }

    bool isConnected( ) { return available( ) > 0 ; }

    tRed & operator=( EthernetClient red ) {
      this->EthernetClient::operator=( red ) ;
```

```
      return *this ;
    }

} RedCliente , RedServicio ;

EthernetServer * server = NULL ;

#endif

#ifdef C_WIFI

//Definimos el nombre y contraseña de red
#define mySSID "AndroidAP"
#define myPassword "mariamaria"

#endif // C_WIFI

//Definimos direccion y puerto
#define XLOCALPORT 80 /* Puerto en el que se escuchan las nuevas
conexiones */
//#define XCENTRAL "192.168.43.235"
#define XCENTRAL "150.214.109.240"

//Identificador de arduino al que solicitamos los datos
int idarduino = 3;

//Conjunto de estados que se usan
enum eEstado {
  CONFIGWIFI , CONFIGWEBSERVER , TEST, CONFIGGET,
  ESPERAFLAG, LEELINEA, LEEFLAG2, LEEFLAG, ESPERA1,

#ifdef CONTEMP

  CAPTURADATOSTEMP,
#endif
#ifdef CONPULSO

  CAPTURADATOSPULSO,
  CAPTURADATOSPULSO2 ,
#endif
#ifdef CONECG

  CAPTURADATOSECG,
#endif
  CONFIGTIMEOUT, ESPERA, ERROR_
}
estado , estado_old ;


#ifdef CONTEMP
int flagTemp;
#endif
#ifdef CONPULSO

int flagPulso;
```

```
int pulso_cuenta ;

//Funcion que lee datos del pulsioximetro
void readPulsioximeter(){

  pulso_cuenta ++;

  if (pulso_cuenta == 50) { // solo tomamos una de cada 50 medidas
    eHealth.readPulsioximeter();
    pulso_cuenta = 0;
    if( estado == CAPTURADATOSPULSO ) estado = CAPTURADATOSPULSO2 ;
  }
}

#endif // CONPULSO

unsigned cuenta ;
#ifdef CONECG
int flagECG;

#define ecgLENGTH 10 //Longitud maxima del array s
int ecg[ecgLENGTH]; //Creamos un array para leer datos de ECG

#define ALENGTH 120
int a[ALENGTH];
volatile int por_a; //Me va a indicar por donde va la interrupcion
String dato;
void timerIsr();
#endif

void check(void) ;

#define SLENGTH 100 //Longitud maxima del array s
char s[SLENGTH + 1]; //Creamos un array para leer datos
unsigned long timeoutflag;
unsigned long timeout;

#define TIMEOUTFLAG 100000 /* intervalo de tiempo entre dos
busquedas de los flag por pulling */
#define TIMEOUTCHAR 10000  /* tiempo maximo que se puede esperar la
llegada del proximo byte en una conexion */

void setup()
{
#ifdef A_LEO
  while ( ! Serial ) ;
#endif

//Inicializamos pulsioxímetro
#ifdef CONPULSO
  pinMode(ledPulso, OUTPUT);
  digitalWrite( ledPulso , LOW ) ;
  eHealth.initPulsioximeter();
#endif
```

```
//Inicializamos ECG
#ifdef CONECG
  por_a = -1;
  Timer1.initialize(50000);
  Timer1.attachInterrupt(timerIsr);//Inicio de interrupcion
#endif

  estado = CONFIGWIFI;
  estado_old = ESPERA ;
}

void loop()
{
  if ( estado != estado_old ) {
    switch ( estado ) {
      case CONFIGWIFI : Serial.println( F("CONFIGWIFI") ) ; break ;
      case CONFIGWEBSERVER : Serial.println( F("CONFIGWEBSERVER") )
; break ;
      case TEST : Serial.println( F("TEST") ) ; break ;
      case CONFIGGET : Serial.println( F("CONFIGGET") ) ; break ;
      case ESPERAFLAG: Serial.println( F("ESPERAFLAG") ) ; break ;
      case LEELINEA : Serial.println( F("LEELINEA") ) ; break ;
      case LEEFLAG2 : Serial.println( F("LEEFLAG2") ) ; break ;
      case LEEFLAG : Serial.println( F("LEEFLAG") ) ; break ;
      case ESPERA1 : Serial.println( F("ESPERA1") ) ; break ;

#ifdef CONTEMP
      case CAPTURADATOSTEMP : Serial.println( F("CAPTURADATOSTEMP")
) ; break ;
#endif
#ifdef CONPULSO

      case CAPTURADATOSPULSO : Serial.println(
F("CAPTURADATOSPULSO") ) ; break ;
      case CAPTURADATOSPULSO2 : Serial.println(
F("CAPTURADATOSPULSO2") ) ; break ;
#endif
#ifdef CONECG

      case CAPTURADATOSECG : Serial.println( F("CAPTURADATOSECG") )
; break ;
#endif
      case CONFIGTIMEOUT : Serial.println( F("CONFIGTIMEOUT") ) ;
break ;
      case ESPERA : Serial.println( F("ESPERA") ) ; break ;
      case ERROR_ : Serial.println( F("ERROR_") ) ; break ;
    }
    estado_old = estado ;
  }

  static int nn ;

  switch (estado) {
    case CONFIGWIFI:
```

```
  //Configuramos la conexion a la red
  Serial.begin(9600);
  #ifdef C_WIFI
  wifiSerial.begin(9600);

  if (!wifly.begin(&wifiSerial, &Serial)) {
    Serial.println("Failed to start wifly");
    estado = ERROR_ ;
    break ;
  }

  if (! wifly.isAssociated()) {

    wifly.setProtocol(WIFLY_PROTOCOL_TCP);
    wifly.setPort(XLOCALPORT);

    wifly.setSSID(mySSID);
    wifly.setPassphrase(myPassword);
    wifly.enableDHCP();
    if (wifly.join()) {
      Serial.println("Joined wifi network");
    }
  }
  Serial.print("MAC: ");
  Serial.println(wifly.getMAC (s, sizeof(s)));
  Serial.print("IP: ");
  Serial.println(wifly.getIP(s, sizeof(s)));
  Serial.print("Netmask: ");
  Serial.println(wifly.getNetmask(s, sizeof(s)));
  Serial.print("Gateway: ");
  Serial.println(wifly.getGateway(s, sizeof(s)));
  wifly.setDeviceID("Wifly-WebClient");
  #endif // C_WIFI

  #ifdef C_ETH
  Ethernet.begin( mac , ip , _dns , gw , sn ) ;
  #endif //C_ETH

  estado = CONFIGTIMEOUT;
  break;

case ERROR_ :
  // Estado en el que se espera un tiempo antes de reiniciar
  // el sistema
  delay( 60000000 ) ; // 1 minuto
  estado = CONFIGWIFI ;
  break ;

case CONFIGWEBSERVER:

//Se configura el servidor
#ifdef CONPULSO
digitalWrite( ledPulso , LOW ) ;
PCintPort::detachInterrupt( 6 ) ;
#endif
```

```
    #ifdef C_WIFI
      if (wifly.isConnected()) {
        wifly.close();
      }

      wifly.setProtocol(WIFLY_PROTOCOL_TCP);
      if (wifly.getPort() != XLOCALPORT ) {
        wifly.setPort(XLOCALPORT);
        wifly.save();
        Serial.println(F("Set port to 80, rebooting to make it
work"));
        if ( wifly.reboot() ) {
          Serial.println( "No funciona el reset, lo forzamos" ) ;
          delay( 1000 ) ;
          break;
          asm volatile ("  jmp 0");
        }
        delay(3000);
      } else
    #endif // C_WIFI
    #ifdef C_ETH

      Ethernet.begin( mac , ip , _dns , gw , sn ) ;

      if( server != NULL ) {
          RedCliente.close( ) ;
          delete server ; server = NULL ;
      }
      server = new EthernetServer(XLOCALPORT) ;
    #endif
      {
        estado = TEST ;
      }

      Serial.println(F("Listo"));
      cuenta = 0 ;
      break;

    case TEST:
    //Este estado se corresponde con el estado ESPERA de la maquina
de estados
      cuenta = 0 ;

      #if defined( C_ETH )
      RedServicio = server->available( ) ;
      #endif

      if (RedServicio.available( ) > 0) {

        // Se ha conectado un cliente, devolvemos una pagina
indicandolo
        Serial.println( F("Conexion entrante")) ;

        RedServicio.flushRx() ;
```

```
        RedServicio.println(F("HTTP/1.1"));
        RedServicio.println(F("Content-Type: text/html"));
        RedServicio.println(F("Transfer-Encoding: chunked"));
        RedServicio.println();
        RedServicio.sendChunkln(F("<html><head>"));
        RedServicio.sendChunkln(F("<title>Arduino</title>"));
        RedServicio.sendChunkln(F("</head><body>"));
        RedServicio.sendChunk(F("<h1>Conectado a arduino ")) ;
        snprintf( s , 9 , "%d" , cuenta ) ;
        RedServicio.sendChunk( s ) ;
        RedServicio.sendChunkln(F("</h1>"));
        RedServicio.sendChunkln(F("<hr>"));
        RedServicio.sendChunkln(F("</body></html>"));
        RedServicio.sendChunkln();
        RedServicio.close( ) ;

        estado = CONFIGGET ;
        break;

      } else if (millis() > timeoutflag) {
        // expira el tiempo antes de chequear el estado de los flags
de medidas
        estado = CONFIGGET;
        break;
      } else {
        estado = TEST;
        break;
      }
      break;

    case CONFIGGET:
    //Estado INI CONFIRMA en la máquina de estados
      cuenta ++ ;
      // Abrimos una conexión con el puerto 80, para verificar
      // el estado de los flags.
      RedCliente.close( ) ;
      if (RedCliente.open(XCENTRAL, 80)) {
        Serial.print(F("Connected to Apache 1\n"));
        //Una vez conectados, se manda la peticion para leer el flag
        RedCliente.print(F("GET
/Pagina/medico/leeFlag.php?idarduino=3"));
        RedCliente.println(F(" HTTP/1.0"));
        RedCliente.println(F("Host: " XCENTRAL ));
        RedCliente.println("Connection: close");
        RedCliente.println(F(""));
        estado = ESPERAFLAG;
        timeout = millis( ) + TIMEOUTCHAR ;
#ifdef CONTEMP
        flagTemp = -1;
#endif
#ifdef CONPULSO
        flagPulso = -1;
#endif
#ifdef CONECG
        flagECG = -1;
```

```
#endif
      } else {
        // no esta disponible el servidor

        estado = CONFIGTIMEOUT;
        RedCliente.close( ) ;
      }
      break;

    case ESPERAFLAG:
      // Se lee linea a linea la salida para determinar el estado de
los flags de tares
      if (RedCliente.available() > 0) {
        nn = 0 ;
        s[nn] = 0 ;
        estado = LEELINEA ;
        timeout = millis( ) + TIMEOUTCHAR ;
      } else if ( timeout < millis() ) {
        //Si expira el tiempo de espera
        estado = ESPERA ;
      }
      break;
    case LEELINEA :
    //Entra dentro del estado LEELINEA de la maquina de estados
      {
        bool x = false;

        if ( (nn < SLENGTH) && (RedCliente.available() > 0) ) {
#define ACARACTERES
#ifdef ACARACTERES
          timeout = millis() + TIMEOUTCHAR;
          //Vamos leyendo la informacion y almacenandola en un array
          s[nn] = RedCliente.read(); // Serial.print( s+nn ) ;
delay( 1 ) ;
          if ((s[nn] == '\n') || (s[nn] == '\r')) {
              Serial.print( "fin de linea\n" ) ;
              estado = LEEFLAG2 ;
          } else if (nn > 4 && strncmp("<br/>", &s[nn - 4], 5) == 0)
{
              Serial.println("Llega br\n");
              estado = LEEFLAG2;
          }
#else // ACARACTERES
          nn = wifly.gets( s , SLENGTH , 10000 ) ;
          estado = nn > 0 ? LEEFLAG2 : ESPERA ;
#endif // ACARACTERES
          if( nn >= 0 ) s[++nn] = 0 ;

        } else if ( nn >= SLENGTH ) {

          estado = LEEFLAG2 ;

        } else if ( (RedCliente.available() < 1) || (x = (millis() >
timeout)) ) {
          if (x) {
```

```
            Serial.println("Timeout");
          }
          if (nn > 0) {
            estado = LEEFLAG2;
          } else {
            estado = ESPERA ;
            RedCliente.close( ) ;
          }

          break;
        }
      }
      break ;

    //Cuando termine de leer, llega a este estado que es el estado
    //FLAG en la maquina de estados
    case LEEFLAG2 :
      nn = 0 ;
      Serial.print("ws: ");
      if ( s[0] ) {
        Serial.print( s ) ;
      }

      {
        //Comparamos el contenido del array s con la cadena "flag"
        char * S = s[0] ? strstr( s , "flag" ) : 0 ;
        if ( S > 0) {

#ifdef CONTEMP
          if ( 1 == sscanf( S , "flagTemp : %d" , &flagTemp ) ) {
            Serial.print( "Detectado :" ) ;
            Serial.print(S);
            Serial.print( "\r\n" ) ;

          } else
#endif
#ifdef CONPULSO
          if ( 1 == sscanf( S , "flagPulso : %d" , &flagPulso ) ) {
            Serial.print( "Detectado :" ) ;
            Serial.print(S);
            Serial.print( "\r\n" ) ;
          } else
#endif
#ifdef CONECG
          if ( 1 == sscanf( S , "flagECG : %d" , &flagECG ) ) {
            Serial.print( "Detectado :" ) ;
            Serial.print(S);
            Serial.print( "\r\n" ) ;
          } else
#endif
            Serial.print( "flag")  ; // TODO quitar

        } else {
          estado = ESPERAFLAG;
          break;
```

```
        }
      }

#ifdef CONTEMP
      //Si encuentra el flag de temperatura activo
      if (flagTemp == 1) {
        RedCliente.close( ) ;
        estado = CAPTURADATOSTEMP;
        break;
      } else
#endif
#ifdef CONPULSO
      //Si encuentra el flag de pulso activo
        if (flagPulso == 1) {
          RedCliente.close( ) ;
          estado = CAPTURADATOSPULSO;
          eHealth.initPulsioximeter();
          pulso_cuenta = 0 ;
          PCintPort::attachInterrupt(6, readPulsioximeter, RISING);
          digitalWrite( ledPulso , HIGH ) ;
          timeout = 30000+millis() ;
          break;
        } else
#endif
#ifdef CONECG
      //Si encuentra el flag de ECG activo
          if (flagECG == 1) {
            RedCliente.close( ) ;
            estado = CAPTURADATOSECG;
            break;
          } else
#endif
            ;
      {
        //Si no se encuentra ninguno activo, volvemos a ESPERAFLAG
        //para buscar de nuevo
        estado = ESPERAFLAG;
        break;
      }
    case CONFIGTIMEOUT:
    case ESPERA:

      if (RedCliente.available() > 0) {
        Serial.println( F("Cerrando conexion 1")) ;
        RedCliente.close( ) ;
        Serial.println( F("Cerrada")) ;
        break;
      }

      if (RedCliente.isConnected()) {
        Serial.println( F("Cerrando conexion 2")) ;
        int t = millis() ;
        RedCliente.close();
        Serial.println( F("Cerrada")) ;
```

```
        Serial.println( (millis() -t) / 1000 ) ;
      }
      //Se configura el temporizador timeoutflag
      timeoutflag = millis() + TIMEOUTFLAG; //millis es una funcion
que nos devuelve la hora actual, le sumamos x segundos
      //Pasaría de nuevo a esperar que se conecte algun cliente
      estado = CONFIGWEBSERVER ;
      break;


#ifdef CONTEMP
    case CAPTURADATOSTEMP:
      //Encender LED que indica que se están capturando datos de
temperatura.
      digitalWrite(ledTemp,HIGH);
      if (RedCliente.isConnected()) RedCliente.close();
      {
        //Captura
        float temp = eHealth.getTemperature();
        Serial.print(F("Temperatura (°C): "));
        Serial.print(temp, 2);
        Serial.println("");
        //Envío
        snprintf( s, SLENGTH, "GET
/Pagina/medico/temperatura.php?flag=%d&temp=%d&idarduino=%d",
                  (int) flagTemp , (int) temp, (int) idarduino ) ;
        Serial.print( s ) ;
        if ( RedCliente.open(XCENTRAL, 80) ) {
          RedCliente.println(s);
          RedCliente.println(F("HTTP/1.0\r\n"));
          RedCliente.println(F("\r\n"));
        } else {
          estado = ESPERA ;
          break ;
        }
      }

      estado = CONFIGTIMEOUT;
      break;
#endif


#ifdef CONPULSO
    case CAPTURADATOSPULSO:
    //Si no llega la interrupcion de haber pulsado el boton
      if( millis( ) > timeout ) {
        Serial.println( "Tiempo excedido, cancelada" ) ;
        estado = CONFIGTIMEOUT ;
      }
      break ;
    case CAPTURADATOSPULSO2:

      {
        //Captura
        Serial.print("PRbpm : ");
```

```
        int bpm = eHealth.getBPM();
        Serial.print(bpm);
        //Captura
        Serial.print("  SPo2 : ");
        int oxigeno = eHealth.getOxygenSaturation();
        Serial.println(oxigeno);

        if( oxigeno < 10 ) {
          //Control para que no tome muestras con valor menor de 10
          estado = CAPTURADATOSPULSO ;
          break ;
        }

        Serial.print("\n");
        Serial.println("============================");
        //Envío
        snprintf( s, SLENGTH, "GET
/Pagina/medico/pulso.php?flag=%d&bpm=%d&oxigeno=%d&idarduino=%d",
                  (int) flagPulso , (int) bpm, (int) oxigeno, (int)
idarduino ) ;
        Serial.print( s ) ;
        if( RedServicio.open(XCENTRAL, 80) ) {
          RedServicio.println(s);
          RedServicio.println("HTTP/1.0\r\n");
          RedServicio.println("\r\n");
        }
      }
      estado = CONFIGTIMEOUT;
      break;
#endif

#ifdef CONECG
    case CAPTURADATOSECG:
      if (RedCliente.isConnected()) RedCliente.close();
      {
        // tomamos las muestras
        for (por_a = 0; por_a < ALENGTH;) {
          delay(25);
        }
        {
          //Creamos un string con json para guardar los datos
          String dato = "{\"datos\":[";
          int cont = 0;
          //Me indica el paquete que se envía
          int iteracion = 0;

          //Los datos capturados los almacenamos en el string
          for (int j = 0; j <= 119; j++) {
            dato += a[j];
            cont++;

            //Si llega a 25, se manda un paquete
            //Si llega a 119, ha terminado el envío
            if ((cont == 25) || (j == (119))) {
              if (j == 119) {
```

```
                dato += "]}";

              }
              if (RedCliente.open(XCENTRAL, 80)) {
                Serial.print("Connected to Apache\n");
              }
              for ( int f = 0 ; f < 25 && !RedCliente.available() ;
f++ ) delay(10) ;
              //Envío
              String envio = "GET
/Pagina/medico/ecg.php?idarduino=%d&flag=&cuenta=";
              envio += iteracion++;
              envio += "&ecg=";
              envio += dato;

              Serial.print("$$$"); //check();
              Serial.print("open\r");
              Serial.println(envio);
              RedCliente.println(envio);
              RedCliente.println("HTTP/1.0");
              dato = ",";
              cont = 0;
              RedCliente.close();


            } else {
              dato += ",";

            }


          }

        }
        estado = CONFIGTIMEOUT;
        break;
      }
#endif
  }
}


#ifdef CONECG
//Funcion para capturar datos de ECG
void timerIsr()
{
  if (por_a < 0 || por_a >= ALENGTH) {
    return;
  }
  a[por_a++] = 100 * eHealth.getECG();
}
#endif
```

Apéndice C: Archivos de recepción de datos

leeFlag.php

```
<?php

//Se accede a este archivo cuando se realiza la petición del flag,
para comprobar si alguno ha sido activado


if(isset($_GET['idarduino'])){

//Almacenamos la variable
$idarduino = htmlspecialchars($_GET['idarduino'],ENT_QUOTES);


//Apertura de la base de datos
$db = new SQLite3('C:/xampp/htdocs/Pagina/final.db');

//Se hace la consulta que devuelve el estado de cada flag
$sql =  "SELECT*FROM PacienteArduino where idarduino=$idarduino";

$result = $db->query($sql);
while ($row = $result->fetchArray(SQLITE3_ASSOC)){
      echo ('flagTemp : '.$row['flagTemp'].'<br/>');
      echo ('flagPulso : '.$row['flagPulso'].'<br/>');
      echo ('flagECG : '.$row['flagECG'].'<br/>');
}

//Actualizamos la variable que contiene la direccion IP
$a=$_SERVER['REMOTE_ADDR'];
$db->exec("UPDATE PacienteArduino SET IP='$a' where
idarduino=$idarduino");
unset($db);
}
?>
```

ecg.php

```
<?php

//Se accede a este archivo cuando el dato solicitado es un ECG

if(isset($_GET["ecg"])){

      //Almacenamos las variables que se reciben desde Arduino
      $datos = htmlspecialchars ($_GET['ecg'],ENT_NOQUOTES);
      $paquete = htmlspecialchars($_GET['cuenta'],ENT_QUOTES);
      $idarduino = htmlspecialchars($_GET['idarduino'],ENT_QUOTES);
      $idsensor=3;
      $medida='ecg';

      //Abrimos y accedemos a la base de datos
```

```
      $db = new SQLite3('C:/xampp/htdocs/Pagina/final.db');

      //Realizamos una consulta para obtener el identificador del
paciente
      $sql="select DISTINCT idpaciente from PacienteArduino where
idarduino=$idarduino";
      echo $sql;
      $result = $db->query($sql);
      $row = $result->fetchArray(SQLITE3_ASSOC);
      $idpaciente = $row['idpaciente'];


      echo $paquete;
      if($paquete == 0){

            //Si recibe el primer paquete, lo insertamos en la base
de datos
            $db->exec("insert into PacienteDatosECG(idpaciente,
            idarduino, idsensor, medida,valorECG) values
            ('$idpaciente', $idarduino, -1, '$medida','$datos')");

      }else if($paquete == -1){

            echo $paquete;
            //Si recibe un paquete erróneo
            $sql="UPDATE PacienteDatosECG SET idsensor=$idsensor
            where idsensor= -1";
            echo $sql;
            $db->exec($sql);

      }else {
            //Para el resto de paquetes que llegan, actualizamos la
variable que los almacena
            echo $paquete;
            $sql="UPDATE PacienteDatosECG SET
            valorECG=valorECG||'$datos' where idarduino=$idarduino
            and idsensor=-1";
            echo $sql;
            $db->exec($sql);
      }
      //Actualizamos el flag a 0
      $db->exec( "UPDATE PacienteArduino SET flagECG=0 where
      idarduino=$idarduino");
}
?>
```

Pulso.php

```php
<?php

//Se accede a este archivo cuando el dato solicitado es pulso

if(isset($_GET['bpm'])){

	//Almacenamos las variables que se reciben desde Arduino
	$bpmsolic = htmlspecialchars($_GET['bpm'],ENT_QUOTES);
	$oxigenosolic = htmlspecialchars($_GET['oxigeno'],ENT_QUOTES);
	$idarduino = htmlspecialchars($_GET['idarduino'],ENT_QUOTES);
	$idsensor='2';
	$medida='pulso';

	//Abrimos y accedemos a la base de datos
	$db = new SQLite3('C:/xampp/htdocs/Pagina/final.db');

	//Realizamos una consulta para obtener el identificador del
paciente
	$sql="select DISTINCT idpaciente from PacienteArduino where
	idarduino=$idarduino";
	$result = $db->query($sql);
	$row = $result->fetchArray(SQLITE3_ASSOC);
	$idpaciente = $row['idpaciente'];

	//Insertamos los datos recibidos en la base de datos
	$db->exec("insert into
	PacienteDatosPulso(idpaciente,idarduino,medida, valorBpm,
	valorOxigeno)values
	('$idpaciente',$idarduino,'$medida','$bpmsolic',
	'$oxigenosolic')");

	//Actualizamos el flag a 0 para que pueda ser activado
	$db->exec( "UPDATE PacienteArduino SET flagPulso=0 where
	idarduino=$idarduino");
}
?>
```

Temperatura.php

```php
<?php
//Se accede a este archivo cuando el dato solicitado es temperatura

if(isset($_GET['temp'])){

	//Almacenamos las variables que se reciben desde Arduino
	$tempsolicitada = htmlspecialchars($_GET['temp'],ENT_QUOTES);
	$idarduino = htmlspecialchars($_GET['idarduino'],ENT_QUOTES);
	$idsensor='1';
```

```
      $medida='temperatura';

      //Abrimos y accedemos a la base de datos
      $db = new SQLite3('C:/xampp/htdocs/Pagina/final.db');

      //Realizamos una consulta para obtener el identificador del
paciente
      $sql="select DISTINCT idpaciente from PacienteArduino where
      idarduino=$idarduino";
      $result = $db->query($sql);
      $row = $result->fetchArray(SQLITE3_ASSOC);
      $idpaciente = $row['idpaciente'];

      //Insertamos los datos recibidos en la base de datos
      $db->exec("insert into PacienteDatosTemp(idpaciente,idarduino,
      idsensor,medida,valorTemp)values
      ('$idpaciente',$idarduino,$idsensor,'$medida','$tempsolicitada'
      )");

      //Actualizamos el flag a 0 para que pueda ser activado
      $db->exec( "UPDATE PacienteArduino SET flagTemp=0 where
      idarduino=$idarduino");
}
?>
```

Apéndice D: Árbol de carpetas y archivos de la interfaz web

```
|   ecg.php
|   final.db
|   leeFlag.php
|   leerECG.php
|   leerPulso.php
|   leerTemp.php
|   pulso.php
|   Recepcion.php
|   temperatura.php
|
+---accederbasedatos
|       final.db
|       formulario.php
|       imagen.jpg
|       login.php
|
\---Usuarios
    +---Administrador
    |       addpaciente.php
    |       asigna.php
    |       eliminarpaciente.php
```

```
|       final.db
|       informacionPaciente.php
|       listapacienteordenadoedad.php
|       listapacienteordenadoid.php
|       listapacienteordenadonombre.php
|       modificarpacienteape.php
|       modificarpacientedir.php
|       modificarpacienteedad.php
|       modificarpacienteid.php
|       modificarpacientenombre.php
|       modificarpacientetlfn.php
|       solicitardatos.php
|       visualizar.php
|
+---Enfermero
|       final.db
|       informacionPaciente.php
|       listapacienteordenadoedad.php
|       listapacienteordenadoid.php
|       listapacienteordenadonombre.php
|       visualizar.php
|
\---Medico
        addpaciente.php
        final.db
        informacionPaciente.php
        listapacienteordenadoedad.php
        listapacienteordenadoid.php
        listapacienteordenadonombre.php
        modificarpacienteape.php
        modificarpacientedir.php
        modificarpacienteedad.php
        modificarpacienteid.php
        modificarpacientenombre.php
        modificarpacientetlfn.php
        solicitardatos.php
        visualizar.php
```