Universidad Pública de Navarra

Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación

Máster en Ingeniería Biomédica

Trabajo Final de Máster

TENSIONAPP:

SISTEMA DE GESTIÓN DE MEDIDAS DE PESO, PRESIÓN SANGUÍNEA Y PULSO; RECOLECCIÓN DE DATOS ESTADÍSTICOS MEDIANTE EL SENSOR FOTOGRÁFICO DE UN SMARTPHONE Y MEDIANTE UN MICROCONTROLADOR.

Autor: Miguel Fuertes Fernández

Tutor: Javier Rodríguez Falces (Universidad Pública de Navarra)

Director:

Tabla de Contenidos

1 Introducción y objetivos 3
1.1 Resumen 3
1.2 Contexto 3
1.3 Objetivos 4
2 Implementación 5
2.1 Requisitos 5
2.2 Arquitectura 5
2.3 Modelo de datos 6
2.4 Cliente 7
2.4.1 Diseño y funcionalidades 7
2.5 Servidor 7
2.6 Medición de frecuencia cardiaca 9
2.6.1 Smartphone: Cámara y flash 9
2.6.2 Microcontrolador: WemosD1 + MAX30100 12
3 Conclusiones y líneas futuras 16
4 Referencias 18
5 Anexos 19

1 Introducción y objetivos

1.1 Resumen

La detección de la frecuencia cardiaca tradicionalmente se realiza mediante los dispositivos médicos conocidos como pulsioxímetros, los cuales realizan la medición en base a la cantidad de veces que cambia la intensidad de rojo al hacer pasar la luz por la punta del dedo.

Este proyecto entre otras cosas pretende implementar dicha técnica utilizando el flash de un teléfono inteligente, así como la cámara, para a la vez, iluminar el dedo y captar la diferencia de rojo del mismo. Adicionalmente se propone la utilización de un microcontrolador y un sensor de pulso para capturar la frecuencia cardiaca.

A la vez, y como forma de almacenar dichas mediciones se ha realizado una plataforma de gestión de mediciones varias, como la frecuencia cardiaca, la presión sanguínea (sistólica y diastólica) y el peso, además de varios indicadores del paciente. La idea es que esta plataforma sirva al medico como punto de recogida de datos para posibles análisis o estudios. De esta manera, se enseña a modo de ejemplo unas simples graficas de frecuencia, presión y peso frente a la edad, así como la posibilidad de generar y exportar un fichero csv anónimo con todas las mediciones del sistema.

1.2 Contexto

La idea surge de un artículo publicado en un medio de noticias, donde destacaban la labor de Ainara Garde en el campo del diagnóstico precoz mediante las medidas de saturación de oxígeno y frecuencia cardiaca:

> Ainara Garde, al frente de un equipo de diferentes disciplinas y cuatro países, desarrolló un instrumento que permite a los trabajadores de salud de primera línea detectar rápidamente la necesidad de los niños de ser hospitalizados y agregan que una característica común de la mayoría de las enfermedades infantiles tratables es la falta de oxígeno.
>
> Para medir este factor de riesgo, el proyecto utilizó un sensor de dedo, el oxímetro de teléfono. De hecho, recopila datos en una aplicación de teléfono inteligente para controlar la saturación de oxígeno en la sangre y la frecuencia cardíaca de una persona. Esta información se combina con una medición de la frecuencia respiratoria.
>
> Garde desarrolló un modelo predictivo que identifica datos anormales de forma fácil y automática.
>
> Ainara Garde

Adicionalmente se propuso por iniciativa del que realiza el proyecto, almacenar estas medidas en un sistema, como control de tensión (hipertenso) y peso. Dicho sistema diferencia como ya veremos entre médico y pacientes, y entre presión sanguínea, peso y pulso.

Posteriormente y aunando ambas iniciativas, surge la idea de exportar todas las medidas para la elaboración de estadísticas posteriores o para enriquecer modelos de Machine Learning que faciliten la predicción en futuras investigaciones.

1.3 Objetivos

Así pues los objetivos son claros; capacidad de medir el pulso, bien sea con la cámara y el flash, bien sea con un microcontrolador y un sensor mediante el USB del teléfono; capacidad de guardar dicha información, así como la presión sanguínea y el peso; capacidad de guardar información relevante del paciente para su posterior estudio estadístico; capacidad de exportar dichas medidas.

2 Implementación

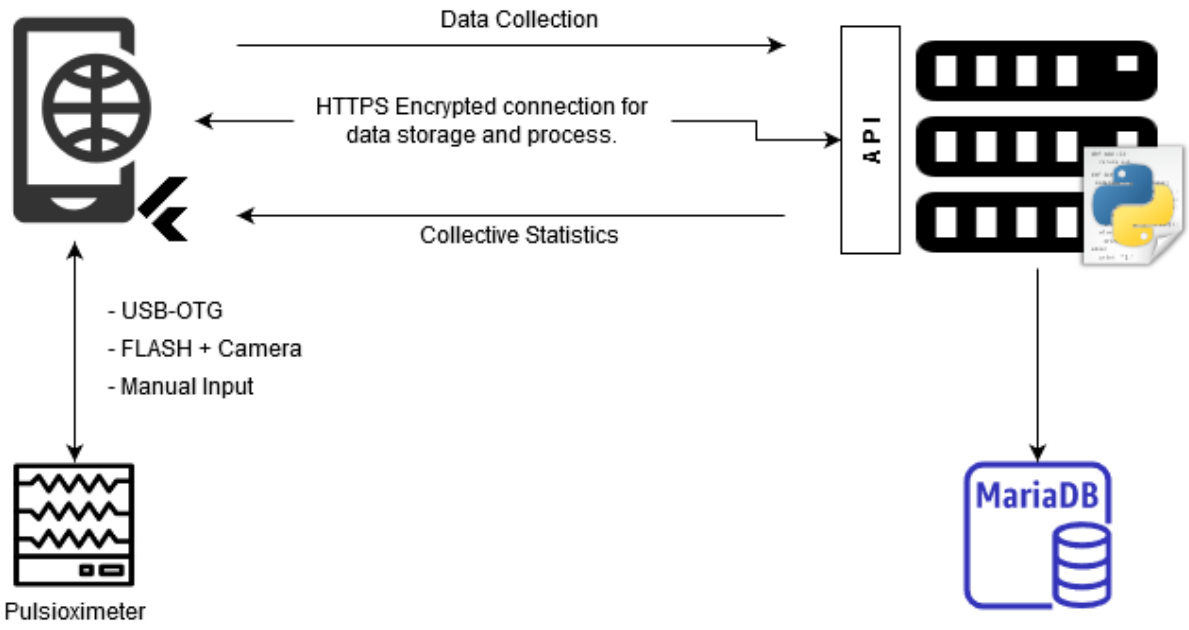
2.1 Requisitos

Los requisitos (o casos de uso, en su forma mas generalizada) es la forma que se emplea en el desarrollo de software para ver que necesidades sugiere el cliente y exponerlos de manera clara en una tabla. Los requisitos recogidos para este proyecto se exponen a continuación:

- Como Médico, quiero poder registrar la edad y el género de un paciente
- Para un paciente, quiero poder introducir variables físicas observadas durante la exploración de dicho paciente, esto es además de la presión sanguínea, el pulso o el peso, los siguientes indicadores: “erc”, “erc fg”, “Asma”, “EPOC”, “Diabetes”, “Dislipemia”, “cardiopatía Isquémica”, “Insuficiencia cardiaca previa”.
- Para un paciente, quiero poder realizar una medida de pulso (frecuencia cardiaca) utilizando la cámara y el flash del smartphone.
- Para un paciente, quiero poder realizar una medida de pulso (frecuencia cardiaca) utilizando un pequeño sensor conectado por USB (Arduino).
- Como médico, quiero poder guardar los datos de esta exploración en un servidor para alimentar un modelo predictivo.
- Como Médico, quiero poder hacer uso de ese modelo predictivo para mostrar una valoración automática por pantalla.
- Como Médico, quiero poder exportar los datos de la base de datos, anonimizados para el posterior análisis en un ordenador.
- Como Medico, quiero poder acceder al mismo sistema (base de datos) desde cualquier smartphone utilizando una cuenta personal (email y contraseña).
- Como Medico, me gustaría poder utilizar la huella y demás sensores biométricos para utilizar la aplicación, sin depender de meter la contraseña cada vez.

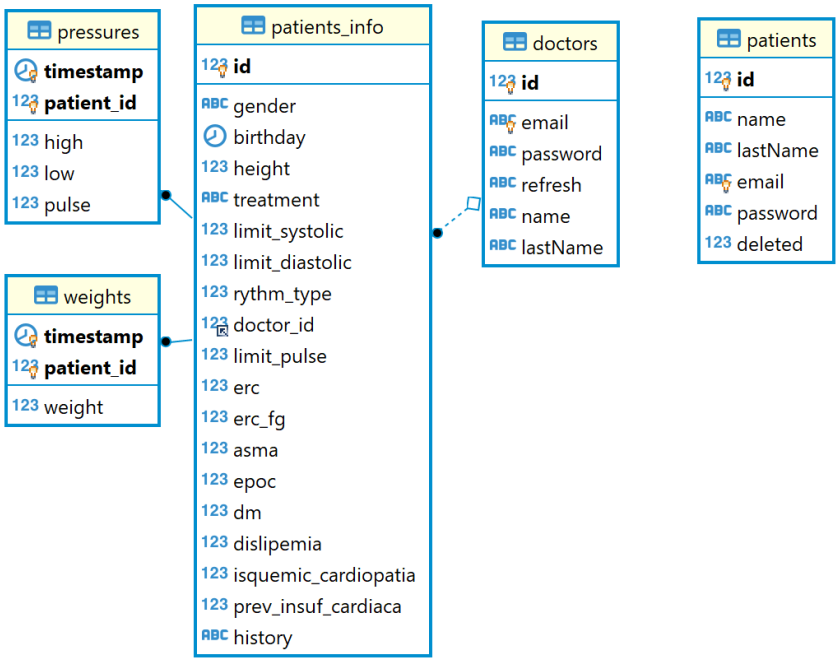
2.2 Arquitectura

Se define arquitectura de software, al diagrama y documentación de los distintos elementos que intervienen de manera física en el sistema; como son bases de datos, servidores, clientes y canales de comunicación. En este caso son tres principalmente, la base de datos (MariaDB en este caso), el servidor (Python + Flask en este caso) y el cliente móvil (Flutter en este caso)



2.3 Modelo de datos

La aplicación consta de dos partes diferenciadas. Una es la toma de medidas y su posterior análisis, exportación de datos, etc. Otra es la gestión de pacientes. Por lo tanto, el modelo que se propone es el siguiente.



Podemos diferenciar cuatro tablas principales, y una quinta separada y sin relación que es la información sensible del paciente. La idea detrás de esto es poder externalizar en un futuro la tabla de pacientes, la información sensible, de manera que nuestro sistema no albergue información sensible y sea más fácil de mantener e institucionalizar.

Las relaciones son de "muchos a muchos" entre la presión sanguínea y la información anónima de paciente, y entre el peso y la información anónima del paciente, ya que son datos. El resto de los datos, como se pueden ver solo se almacenará una copia, la última.

2.4 Cliente

La programación del cliente (o teléfono) de este sistema lo único que requiere es acceso a bajo nivel al hardware del teléfono, con lo que tendremos que ir a lenguajes de programación nativos si queremos obtener datos fiables de la cámara, puesto que necesitaremos diferencias de color en tiempo real para poder extrapolar la gráfica del latido del corazón. Quedan descartadas entonces las soluciones hibridas, Cordova/PhoneGap así como las soluciones web PWA o aplicaciones progresivas en HTML5.

Para la programación nativa lo ideal será realizarla en las dos plataformas más extendidas como son Android e iOS. Sin embargo, últimamente han surgido soluciones muy interesantes por parte de Google y Facebook para facilitar la compilación a código nativo en estas dos grandes plataformas, partiendo de un único código común. Estas soluciones son Flutter y ReactNative respectivamente.

Mi propuesta inicial, y siempre y cuando estos frameworks no supongan una limitación, es utilizar Flutter para el desarrollo de la aplicación de cliente, que como ya he dicho antes se traducirá en Android/Java cuando compilemos para Android y en ObjectiveC/Swift cuando compilemos para iOS.

Para las partes donde hace falta acceso a más bajo nivel, se han tenido que programar piezas especificas en Android (no se dispone de un ordenador Mac para la programación en iOS), por la ya comentada carencia en el framework de Flutter.

Flutter utiliza un lenguaje de programación de alto nivel desarrollado por Google, llamado Dart. Emplea la orientación a Objetos y es muy flexible.

2.4.1 Diseño y funcionalidades

Esta es la aplicación móvil que se encarga de hablar con el servidor. la parte que el usuario/medico va a ver. Entre sus funcionalidades encontramos:

- Consultar el histórico de pacientes
- Consultar estadísticas de todas las mediciones
- Crear nueva medición para un paciente concreto:
- medición de peso *(Entrada manual)*
- Medición de presión sanguínea *(Entrada manual)*
- medición de pulso *(Entrada manual, y usando Cámara y flash del teléfono Android)*

2.5 Servidor

Para la implementación en servidor se plantea el uso de Python como lenguaje de programación web. No es el más versátil, pero si es el que mejor va a manejar todo el procesamiento de datos y ofrece la posibilidad si hiciera falta de interactuar con R en el servidor en el que decidamos desplegarlo.

Para la parte de la API, con el framework Flask, estaríamos cubiertos, puesto que no queremos mostrar nada vía web, ni mostrar ningún contenido html, no necesitamos django para nada. Flask es un framework simple que nos permitirá devolver en formato json la información para que el cliente la reciba y pinte la información en el teléfono.

Adicionalmente se propone el formato CSV para descargar el listado de las mediciones, como se plantea en uno de los requisitos.

La manera que tienen las aplicaciones cliente de conectarse con el servidor es mediante el uso de API. La API o “Application Programin Interface” es la interfaz que un servidor expone a internet para que se le pueda preguntar por información. Anexo, expongo los diferentes “endpoints” disponibles y para que se utilizan, así como el modelo de datos (DTO).

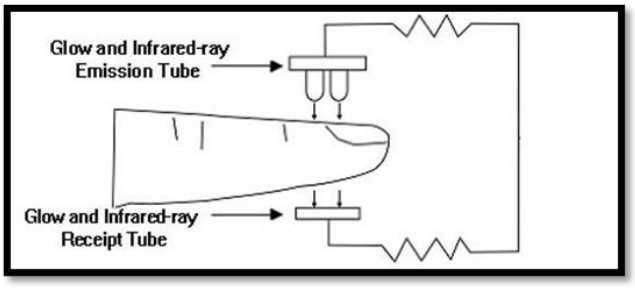
La autenticación se realiza mediante tokens JWT o “Json Web Tokens”. Estos tokens se envían en la cabecera de la llamada, y contienen el identificador del médico logueado, la fecha hasta que ese token es válido, y todo ello firmado con una clave privada que solo conoce el servidor. Si un atacante quisiera suplantar a un médico, la firma fallaría y se vería que el token no es válido. El servidor devolverá un 401-403 indicando que no está autorizado. Es el standard hoy día

2.6 Medición de frecuencia cardiaca

Como se ha indicado en la introducción, se presentan dos métodos para la obtención de la frecuencia cardiaca, el teléfono con el flash y la Cámara, y un pequeño dispositivo Arduino (un microcontrolador) con un sensor de frecuencia MAX30100.

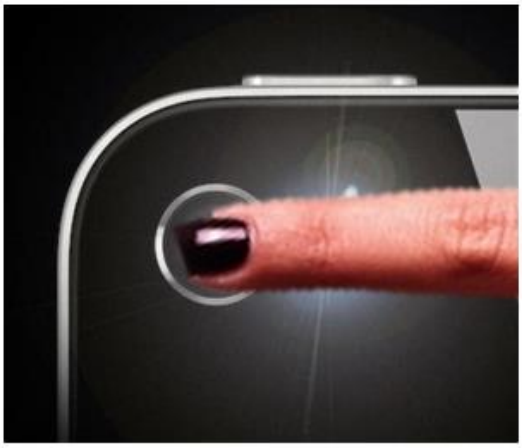
2.6.1 Smartphone: Cámara y flash

La manera en que un pulsioxímetro realiza la medición es mediante la intensidad de luz que llega a un receptor tras pasar a través del dedo del paciente.



Al oxigenarse el dedo con cada latido, hay más sangre circulando por los vasos del dedo, lo que impide que más cantidad de luz llegue definitivamente al sensor. Midiendo la cantidad de esas variaciones en un tiempo concreto, podemos determinar el número de latidos por minuto. Vamos a replicar este mecanismo con el flash y la Cámara de un teléfono inteligente.

En el caso del smartphone la idea es similar, solo que en vez de tener el emisor de luz y el receptor alineados en la misma vertical, usaremos el flash del smartphone como la fuente de luz, y la cámara como el sensor de luz.



Mediremos diferencia de color (en este caso el rojo) para ver cuando hay un flujo de sangre (debido al latido) y cuando no.

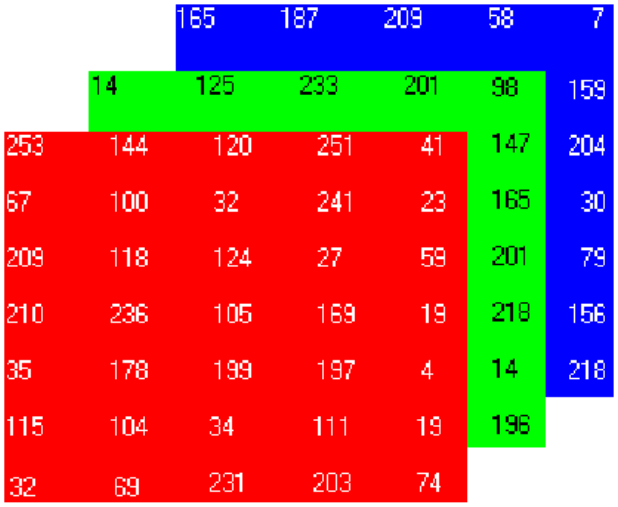
Repitiendo esto varias veces podemos obtener las pulsaciones por minuto.

Esta diferencia de color la mediremos solamente en el canal rojo, ya que la sangre es de ese color, y sería inútil contemplar los otros canales ya que añadirían coste computacional para no obtener ningún beneficio.

Antes de seguir con los algoritmos implementado, es interesante esbozar como se representan en memoria y en un ordenador las imágenes, para entender el proceso aplicado al cálculo de la variación de la intensidad.

Esencialmente en un ordenador la forma en que se representan los diferentes colores en una pantalla es mediante la combinación con distinta intensidad de los tres colores principales, Rojo, Verde y Azul. Así un pixel (o Picture Element) es la unidad mínima representable en un sistema lógico, y tendrá como si de un sistema de coordenadas se tratara, tres valores asociados, una intensidad para el rojo, una intensidad para el verde y una intensidad para el azul.

Podemos entender entonces que una imagen es una repetición de estos pixeles ordenados en dos dimensiones, y un video es un array en el tiempo de estas imágenes. Así podemos entender una imagen como una matriz de tantas columnas como el ancho de la imagen, tantas filas como el alto de la imagen y con tanta profundidad como el numero de canales, normalmente tres, rojo, verde y azul.

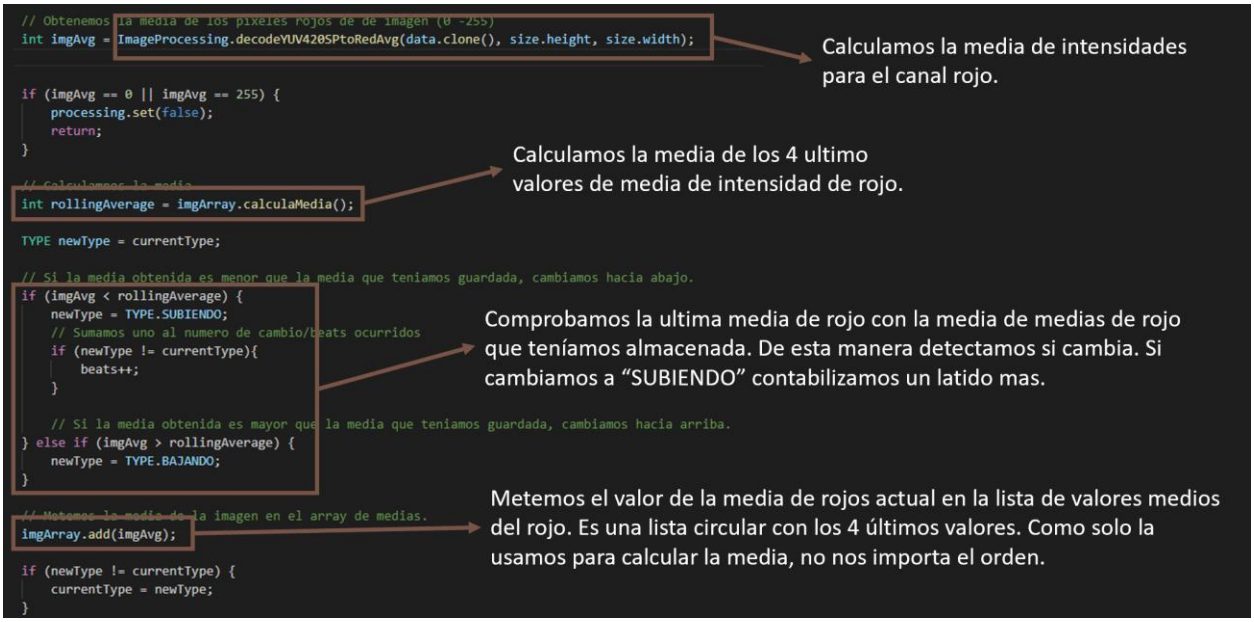


De esta manera y siguiendo la figura en la posición 1,1 de la imagen, tendremos el valor (253,14,165) lo que equivale a este color: COLOR. La forma en la que se almacenan los colores puede variar, dando más o menos definición a la intensidad que se puede alcanzar, pero el estándar común aceptado por la red es de 32bits de información por canal o lo que es lo mismo 256 valores de intensidad (o FF valores si lo representamos en hexadecimal, que es muy común).

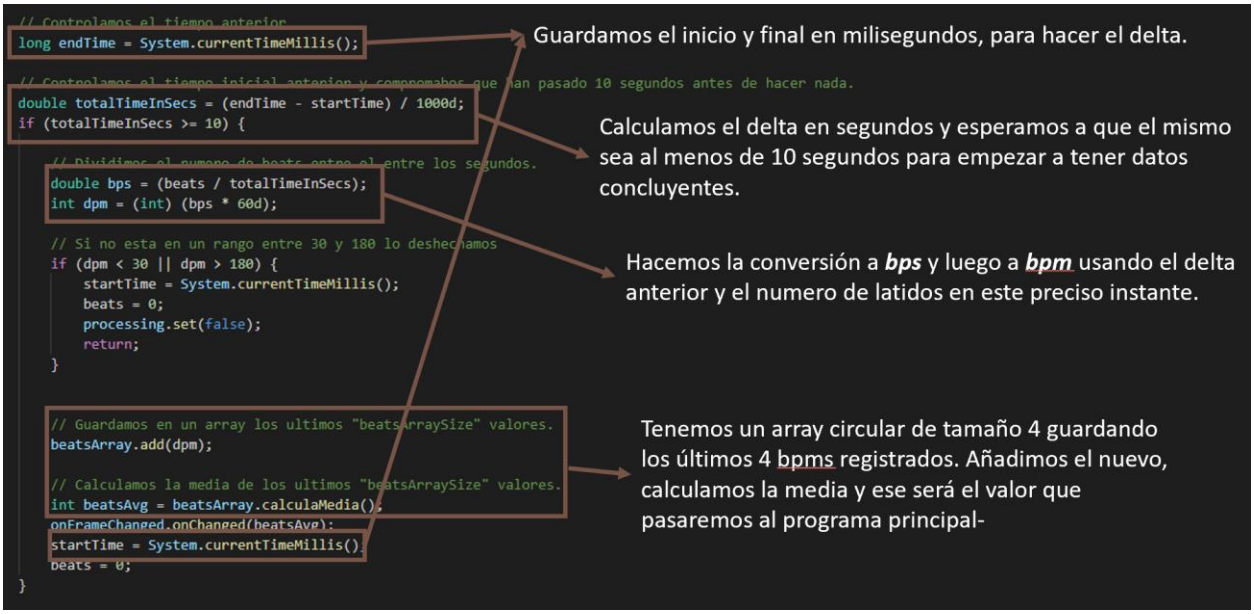
Al capturar una imagen de un objeto eminentemente rojo, los otros dos canales (verde y azul) no aportarán nada a la imagen, con lo que tendrán valores próximos a 0 y desechables. Convirtiendo una matriz de AxAx3 en una matriz de AxA (donde A y A son alto y ancho respectivamente), reduciendo enormemente la complejidad.

El cálculo ahora se limita a agregar la matriz de colores como la media de las intensidades, a repetirlo en un tiempo establecido y a tomar la variación de dicha agregación (en este caso la media) como un latido del corazón.

Se adjunta el código completo. Sin embargo, quiero resaltar aquí las partes más importantes. Una primera parte en la que calculamos la media de intensidades de la imagen y lo guardamos en un array de medias. Con ese array podemos ver cuando estamos subiendo y cuando bajando para detectar un latido.



En el siguiente paso, cronometramos cuantos latidos ha habido en un periodo de tiempo y devolvemos el numero de pulsaciones por minuto o bpm.

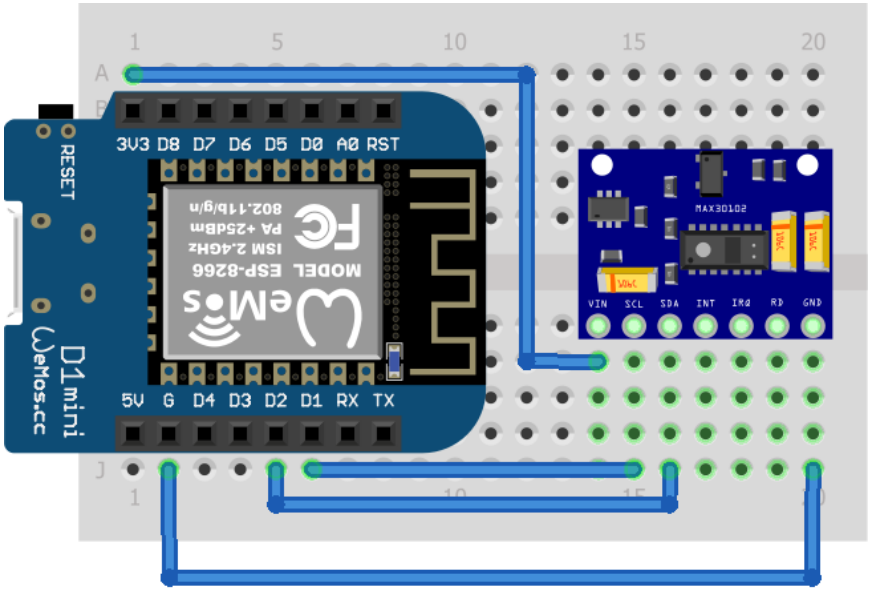


2.6.2 Microcontrolador: WemosD1 + MAX30100

Adicionalmente y como alternativa a la cámara del teléfono, y puesto que este puede no tener flash o estar en una posición imposible la cámara con respecto del flash, se añade la posibilidad de utilizar un microcontrolador y un sensor de frecuencia para realizar la medición. Este controlador se conectará al teléfono mediante el puerto USB del mismo, haciendo uso de la tecnología OTG existente en todos los teléfonos actuales.

Para este proyecto el microcontrolador elegido es un WemosD1, una implementación del conocido ESP8266 con más ROM y conexión serial mediante usb. El ESP8266 es un microprocesador que surgió hace ya unos años con la peculiaridad de implementar toda la pila de red y conexión inalámbrica 802.11b/g/n. Los drivers permiten la conexión con el PC de la misma manera que lo haría un Arduino, la programación es C, similar al Arduino y se puede programar desde el mismo entorno de desarrollo de Arduino. A todos los efectos y con la configuración adecuada es un Arduino con conexión inalámbrica. Por ello en adelante se referirá a él como Arduino.

Como sensor de frecuencia cardiaca se eligió el MAX 30100, un sensor económico que se conecta de manera sencilla con cualquier microcontrolador mediante el protocolo I2C. Trabaja a 3.3v y solo necesita de dos entradas digitales en el Arduino para funcionar.



Por otro lado, el estándar usb proporciona por dos de sus líneas alimentación de 5v y tierra, lo que facilita que el propio smartphone mediante su conexión micro-usb o usb-c proporcione alimentación al circuito. Como he dicho antes el WemosD1 implementa el microcontrolador ESP8266 pero le añade varias funcionalidades. Una de estas funcionalidades es un regulador de voltaje que transforma los 5 voltios continuos en 3.3 voltios continuos.

De la misma manera que con la medición mediante la cámara, se proporciona el código completo en los anexos del proyecto. Sin embargo, aquí se destaca lo más relevante.

Al ser un programa de Arduino, el "Sketch" consta de dos funciones principales, una de configuración y otra de bucle infinito donde a cada tick de reloj se repetirá una función, es un bucle infinito. Estas funciones son las de setup() y la de loop().

En la función setup inicializamos el WifiManager (se encarga de crear una red wifi para que configuremos la conexión al router, en caso de no poder conectarse a ninguna red al arrancar) e inicializamos el sensor de pulso ("***particleSensor***").

La conexión con el sensor de pulso se hace mediante el protocolo I2C. Al utilizar los puertos D1 y D2 del GPIO del Arduino, que son los que él espera para una conexión I2C, podemos directamente cargar el objeto ***Wire*** y pasárselo al ***particleSensor*** para inicializar el sensor.

También abrimos la conexión "***Serial***" con el que se conecte por usb. En la fase de desarrollo es el ordenador y una terminal serial, y una vez terminado será el smartphone mediante el usb-otg.

```
void InitServer()
{

    SPIFFS.begin(); // Start the SPI Flash Files System

    server.serveStatic("/", SPIFFS, "/").setDefaultFile("index.html");

    server.begin();
    Serial.println("HTTP server started");
}

void setup()
{
    Serial.begin(115200);
    // Creamos una instancia de la clase WiFiManager
    AsyncWiFiManager wifiManager(&server, &dns);

    // Descomentar para resetear configuración
    //wifiManager.resetSettings();

    // Cremos AP y portal cautivo
    wifiManager.autoConnect("ESP8266Temp");
    //Serial.println("Ya estás conectado: "+ WiFi.localIP());
    webSocket.begin();
    //webSocket.onEvent(webSocketEvent);

    // Initialize sensor
    particleSensor.begin(Wire, I2C_SPEED_FAST); //Use default I2C port, 400kHz speed
    particleSensor.setup();                     //Configure sensor with default settings
    particleSensor.setPulseAmplitudeRed(0x0A);  //Turn Red LED to low to indicate sensor is running

    InitServer();
}
```

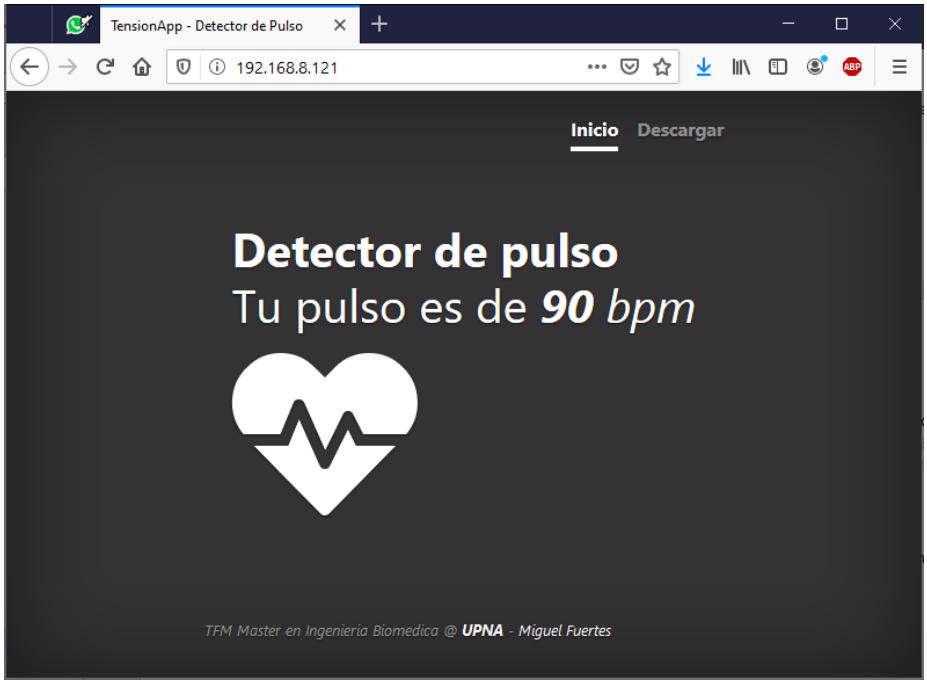
Por otro lado, en la función loop() empezamos por comprobar si la intensidad de infrarrojos es mayor que un umbral (dedo encima del sensor) y esperamos que haya un pulso ("***checkForBeat()***").

Cuando lo encontramos (valor booleano de si o no) guardamos ese milisegundo y hacemos cálculos con los milisegundos anteriores para calcular la media de pulsaciones. Volvemos a estar en una situación en la que tenemos un si o no a “hay un pulso”, y es el recuento en un delta de tiempo el que nos da las pulsaciones por minuto.

Al final mandamos el pulso por el USB (“***Serial.println()***”).

```
void loop()
{
    webSocket.loop();

    long irValue = particleSensor.getIR();
    if (irValue > 7000)
    {
        if (checkForBeat(irValue) == true)
        {

            long delta = millis() - lastBeat;
            lastBeat = millis();

            beatsPerMinute = 60 / (delta / 1000.0);

            if (beatsPerMinute < 255 && beatsPerMinute > 20)
            {
                rates[rateSpot++] = (byte)beatsPerMinute;
                rateSpot %= RATE_SIZE;


                beatAvg = 0;
                for (byte x = 0; x < RATE_SIZE; x++)
                    beatAvg += rates[x];
                beatAvg /= RATE_SIZE;
            }

            sprintf(datagram, "{ \"pulse\": \"%lu\" }", beatAvg);
            webSocket.broadcastTXT(datagram);
            Serial.println(datagram);
        }
    }
}
```

Adicionalmente, aunque solo sirve de manera orientativa ya que no interactúa con la app, se manda la información del pulso obtenida mediante websockets a una pagina web simple que el propio microcontrolador sirve.



La cadena de texto con la información en formato JSON es luego capturada por la clase Java nativa de Android y procesada.

Cuando todo está correcto y el usuario pulsa el botón “Guardar”, se le devuelve el valor “_beat” al widget de ***Flutter*** para su posterior envío al servidor.

Como dije al principio, no todo se puede hacer con ***Flutter***, y aunque esto sí, era más claro desarrollarlo nativamente en Android y conectarlo a ***Flutter***. En cualquier caso, iOS es un sistema muy cerrado y no permite el acceso al USB a cualquier aplicación.

```
@Override
public void onNewData(byte[] data) {
    try {
        JSONObject pulse = new JSONObject(new String(data));

        _beat = pulse.getInt("pulse");
        runOnUiThread(() -> {
            //_heart.setColorFilter(_red ? Color.RED : Color.BLACK);
            _text.setText(_beat + " bpm");
        });
    } catch (JSONException e) {
        e.printStackTrace();
    }

}
```

Siendo ***data = “{pulse:’120’}”***

3 Conclusiones y líneas futuras

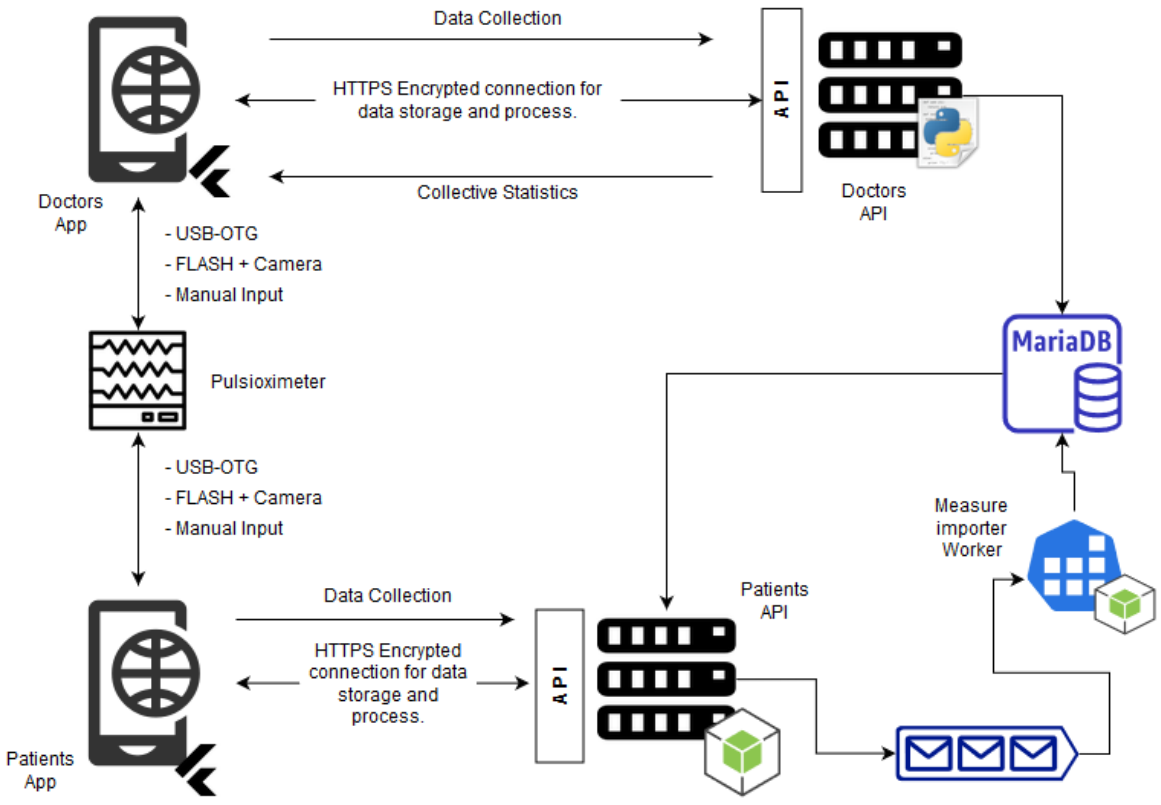
La tecnología ha avanzado a pasos agigantados, y lo que antes suponía un problema, como es la gestión de datos o la medición de los mismos ahora se puede resolver con mucha facilidad y desde la distancia si hiciera falta.

Hoy día los móviles disponen de la tecnología suficiente como para realizar gran cantidad de seguimiento médico, inmediatez de diagnóstico, y posibilidad de hacer todo esto desde la comodidad de un aparato que siempre llevas en el bolsillo.

Tras enseñar a una cardióloga el proyecto, y atender a sus recomendaciones, le veo futuro a la aplicación en el campo de seguimiento de pacientes, y en el campo de la investigación. Esta aplicación le es funcional a la vez al médico para gestionar a sus pacientes y ver rápidamente una posible patología, como para almacenar una gran cantidad de datos que poder utilizar luego anónimamente para realizar estudios estadísticos.

Una de las carencias que la cardióloga detectó es que la aplicación a día de hoy solo le es funcional al médico, y solo tiene sentido en el tiempo en que el paciente esta en consulta. A futuro tiene sentido dividir la aplicación en dos, una parte para el paciente y otra para el médico. Así la toma de mediciones quedara del lado del paciente (y escondida en un submenú para el medico) y la estadística de datos o posible mensaje de diagnostico de parte del médico.

Visto que cada medico puede atender a muchos pacientes y para no sobrecargar el sistema, la arquitectura que se propone a futuro es la siguiente.



La idea es separar la aplicación en dos, como ya se ha comentado, e incorporar un servidor más que dependa de la misma base de datos. Las mediciones realizadas por los pacientes se encolarán y se procesarán a medida que el encargado de procesar los mensajes tenga tiempo de ejecución disponible. Así aceleramos el proceso y hacemos todo mas confiable. A futuro podemos añadirle más colas y replicación de mensajes para asegurar el servicio.

Aunque a día de hoy ya se cuenta con un campo “tratamiento” donde el medico puede guardar información relevante al paciente y a su tratamiento, a día de hoy solo se puede ver y editar desde la aplicación del médico, la única que hay. Con el nuevo sistema, se espera que el medico vea y edite el campo, y que el paciente, instantáneamente vea lo que el doctor le ha pautado.

Adicionalmente se le puede incorporar un servicio de mensajería entre el paciente y el doctor.

En el campo de la adquisición de datos, se podría perfeccionar el sistema de captura de datos con la cámara, haciéndolo mas fiable con más pruebas contra pulsioxímetros reales, así como integrar una comunicación persistente entre la aplicación del paciente y el posible wearable que este pueda poseer.

4 Referencias

(Meter aquí el articulo que vi que explicaba como medir con el smartphone)

5 Anexos