



# TensionApp

Sistema de gestión y monitorización de medidas de presión sanguínea, pulso y peso, así como medición de pulso mediante smartphone y Arduino

Miguel Fuertes

# IDEA I: Artículo *Ainara Garde*

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.

- **Artículo**

- Identificar pacientes de riesgo para enfermedades coronarias con solo utilizar un smartphone para medir el pulso.
- Implementar en un dispositivo móvil dicho Sistema de medición.

## IDEA II: Control de Medidas


- Soy hipertenso y ninguna de las aplicaciones en el mercado me convencia para medir esto.
  - Control de Peso
  - Control de Presion Sanguinea

TensionApp

Inicio Documentacion

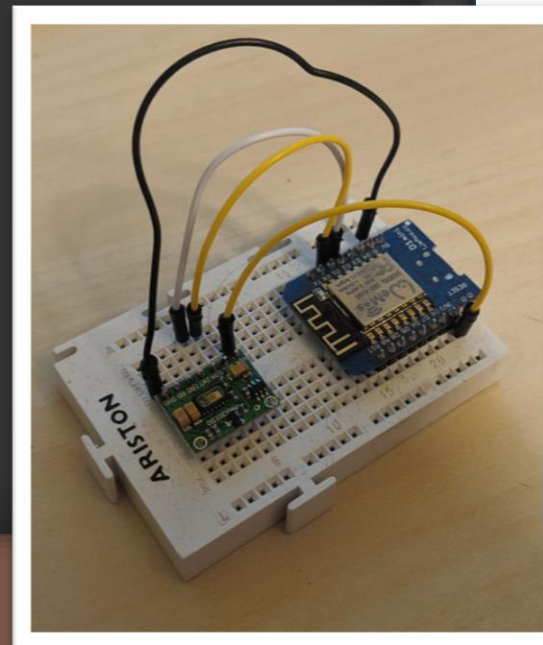
## TensionApp server.

Servidor para la app de control de presión, pulso y peso.



Descarga la aplicación mediante el QR  
(escaneándolo o clickando sobre él)

TFM Master en Ingeniería Biomedica @ UPNA · Miguel Fuertes



MI 5s Plus

2:55 P

Mis Pacientes

Javier Fuertes  
Edad: 25 años Altura: 175 cm

Miguel Fuertes  
Edad: 30 años Altura: 180 cm

Estadísticas

MI 5s Plus

2:56 P

Miguel Fuertes

Solo Pulso	26 Ene 11:36
Pulso: 137	
Solo Pulso	25 Ene 23:12
Pulso: 137	
Solo Pulso	25 Ene 00:35
Pulso: 107	
Peso: 118.0 kg	24 Ene 23:44
Alta: 124 Baja: 80	24 Ene 23:44
Pulso: 70	
Solo Pulso	24 Ene 23:29
Pulso: 146	
Solo Pulso	24 Ene 22:57
Pulso: 101	

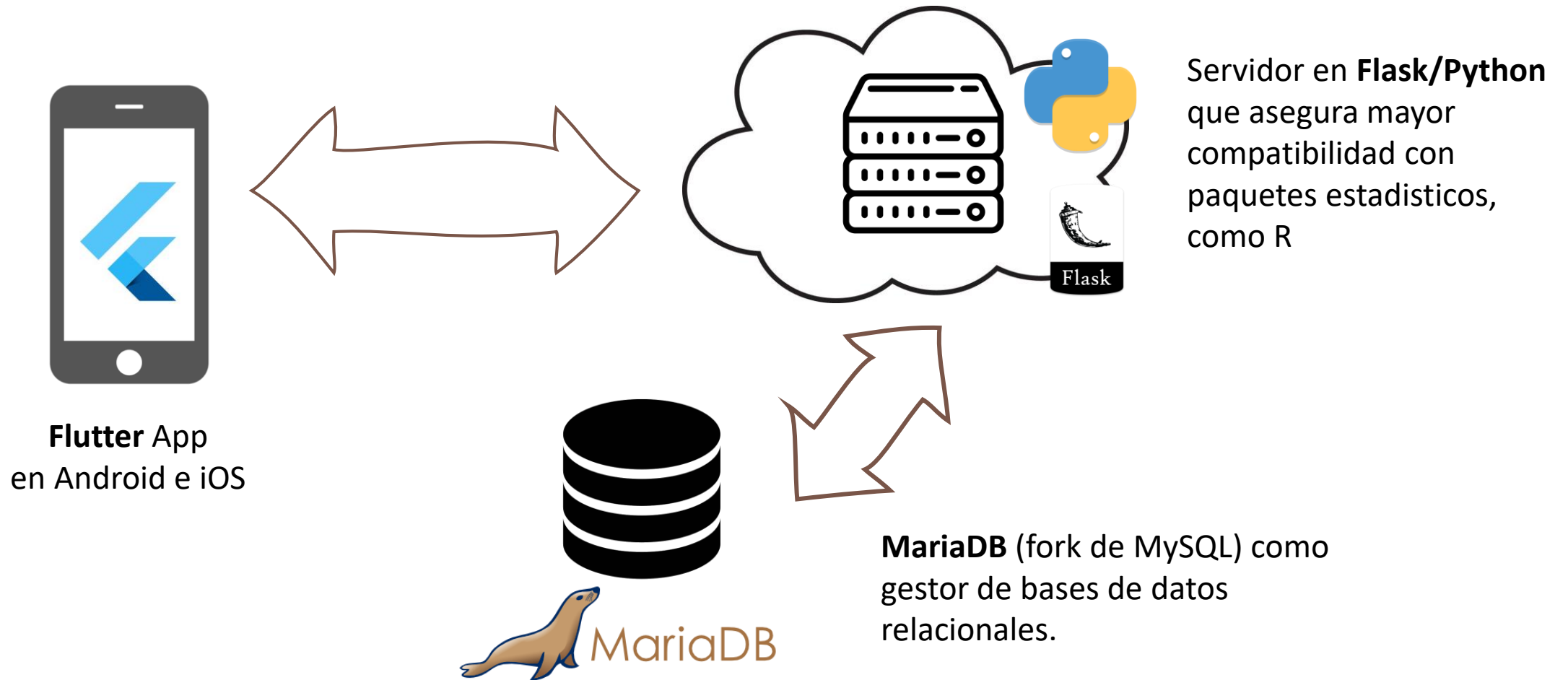
Tensión Peso + Pulso

# PROPUESTA: TensionApp

# TensionApp

- Gestión de Pacientes asignados a un medico
  - Creación de medidas
    - **Presión sanguínea:** introducción manual copiando los datos desde el aparato.
    - **Peso:** Introducción manual copiando los datos desde la bascula.
    - **Pulso:**
      - Introducción manual
      - Lectura utilizando la cámara y el flash del móvil
      - Lectura utilizando un sensor y controlador Arduino conectado mediante el USB del teléfono.
  - Creación/Edición/Borrado de pacientes
- Consulta de gráficas con las medidas anónimas de todos los pacientes.

# Arquitectura



# Servidor



La tecnología utilizada para crear el servidor donde se almacenaran los datos es ***Python***, por su gran cantidad de módulos que facilitan la interconexión del programa con casi todos los sistemas de procesamiento de estadísticas.

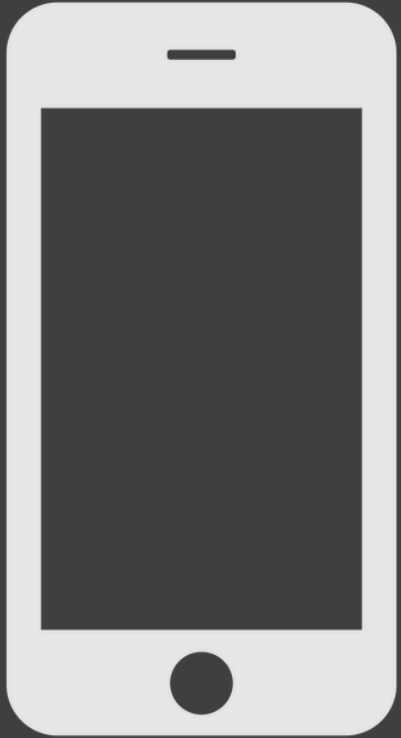
Encima de Python, el framework utilizado es ***Flask***, ya que provee una manera rápida y potente de implementar una API.

La comunicación se hace mediante HTTPS, y la autenticación se hace mediante JWT, que garantiza que los tokens sean auténticos para devolver la información requerida.



[Mas información](#)

# Cliente



La tecnología utilizada para crear el cliente (la app) es **Flutter**. **Flutter** es una tecnología desarrollada por Google, programada en Dart, cuya peculiaridad es que compila el lenguaje Dart al lenguaje nativo de ambas plataformas, *Java/Kotlin* en *Android* y *Objective C/Swift* en *iOS*.

Con un solo código podemos generar ambas aplicaciones. Para la mayoría de las acciones es suficiente, pero si queremos utilizar la cámara para medir el pulso, o comunicarnos con un dispositivo mediante el puerto USB, tenemos que desarrollar esa característica específica en el lenguaje nativo de cada plataforma (actualmente solo desarrollado en Android).



[Mas información](#)



# Extra: Dispositivo Arduino

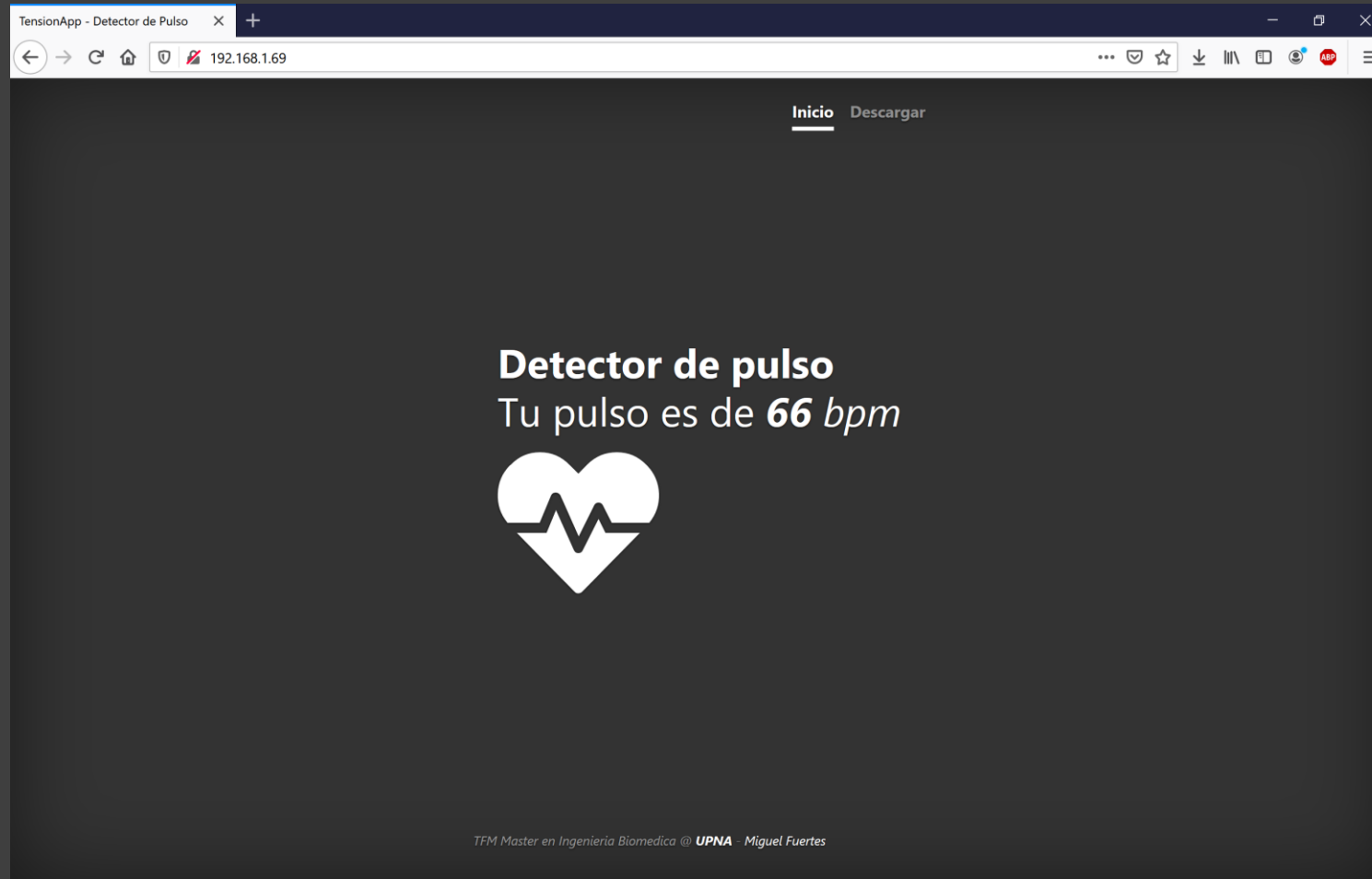
Aprovechando la tecnología **OTG** actualmente presente en todos los smartphone **Android**, es posible programar y conectar un dispositivo **Arduino** y un sensor (en este caso el **MAX30100**) al smartphone mediante un cable USB.

En este caso, el microcontrolador elegido es el **Wemos D1** que es un **ESP8266** mejorado. Este microcontrolador cuenta con conexión WiFi. Podemos ver la lectura en un navegador a la vez que se le envía dicha medida al smartphone mediante el cable USB.

\*La unidad de demo en la exposición tendrá el wifi desactivado, debido a la complejidad de conectarlo a la red de la universidad.



# Extra: Dispositivo Arduino

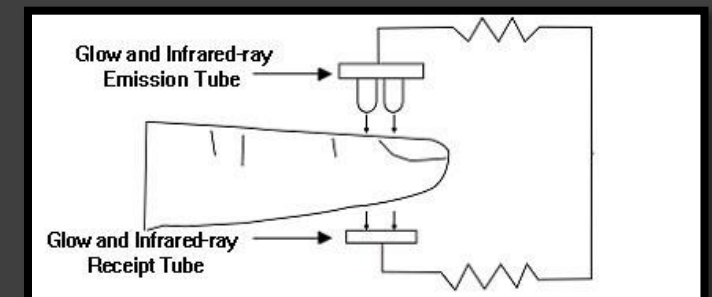


*Captura del pulso: Cámara*

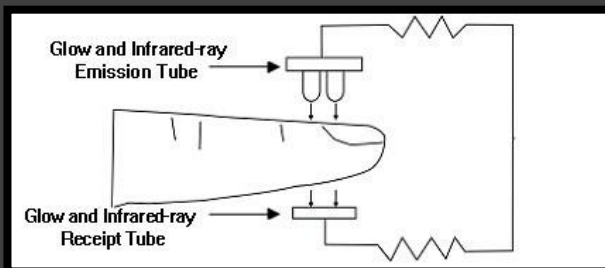
# *Captura del pulso: Cámara*

## ¿Cómo funciona un Pulsioxímetro?

- Entran en juego varias partes, pero en esencia se le hace pasar un haz de luz por el dedo, desde un emisor hasta un receptor.
- Al oxigenarse con cada latido, hay mas cantidad de sangre en el dedo con lo que la luz que recibe el receptor es menos por un intervalo mínimo de tiempo.
- Al repetir esto por un intervalo de tiempo, se calcula viendo el numero de veces que la intensidad registrada ha sido menor.



# *Captura del pulso: Cámara*



## **¿Cómo funciona un Pulsioxímetro en un Smartphone?**

- En este caso 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.

# Captura del pulso: Cámara

## Representación de la imagen:

En informática o en general, la imagen se representa como una colección de tres matrices, mapeando los tres colores principales. En cada matriz (filas y columnas, como anchura y altura de la imagen) se almacena un valor de intensidad entre 0 y 255 para cada uno de los colores:

- El 255,0,0 es el Rojo
- El 0,255,0 es el Verde
- El 0,0,255 es el Azul
- El 255,255,0 es el Amarillo
- ... y así todas las posibles combinaciones.

Como el dedo y la sangre son de color rojo, computar las otras dos aportaciones sería cálculo absurdo que no aportaría valor, con lo que solo trabajaremos con una matriz, la roja.

			165	187	209	58
	14		125	233	201	98
253	144	120	251	41	147	
67	100	32	241	23	165	
209	118	124	27	59	201	
210	236	105	169	19	218	
35	178	199	197	4	14	
115	104	34	111	19	198	
32	69	231	203	74		

# Captura del pulso: Cámara

	165	187	209	58	7
14	125	233	201	98	159
144	120	251	41	147	204
100	32	241	23	165	30
118	124	27	59	201	79
236	105	169	19	218	156
178	199	197	4	14	218
104	34	111	19	196	
69	231	203	74		

Para cada momento calcularemos la media del color rojo de lo calculado por la imagen.

Si justo ha habido un latido, habrá mas sangre, con lo que veremos menos luz, con lo que el color será mas oscuro (mas cerca de 0).

Si no ha habido latido, hay menos sangre, veremos mas luz, y el color será mas claro (mas cerca de 255)

En cualquier caso lo que nos interesa es saber cuando hay una diferencia de color, para mediante calculo de tiempo (milisegundos) llegar a saber las pulsaciones por minuto.

# Captura del pulso: Cámara

```
// Obtenemos la media de los pixeles rojos de de imagen (0 -255)
int imgAvg = ImageProcessing.decodeYUV420SPtoRedAvg(data.clone(), size.height, size.width);
```

Calculamos la media de intensidades para el canal rojo.

```
if (imgAvg == 0 || imgAvg == 255) {
    processing.set(false);
    return;
}
```

```
// Calculamos la media
int rollingAverage = imgArray.calculaMedia();
```

Calculamos la media de los 4 ultimo valores de media de intensidad de rojo.

```
TYPE newType = currentType;
```

```
// Si la media obtenida es menor que la media que teniamos guardada, cambiamos hacia abajo.
```

```
if (imgAvg < rollingAverage) {
    newType = TYPE.SUBIENDO;
    // Sumamos uno al numero de cambio/beats ocurridos
    if (newType != currentType){
        beats++;
    }
}
```

Comprobamos la ultima media de rojo con la media de medias de rojo que teníamos almacenada. De esta manera detectamos si cambia. Si cambiamos a “SUBIENDO” contabilizamos un latido mas.

```
    // Si la media obtenida es mayor que la media que teniamos guardada, cambiamos hacia arriba.
} else if (imgAvg > rollingAverage) {
    newType = TYPE.BAJANDO;
}
```

```
// Metemos la media de la imagen en el array de medias.
imgArray.add(imgAvg);
```

Metemos el valor de la media de rojos actual en la lista de valores medios del rojo. Es una lista circular con los 4 últimos valores. Como solo la usamos para calcular la media, no nos importa el orden.

```
if (newType != currentType) {
    currentType = newType;
}
```



# Captura del pulso: Cámara

```
// Controlamos el tiempo anterior
long endTime = System.currentTimeMillis();
```

Guardamos el inicio y final en milisegundos, para hacer el delta.

```
// Controlamos el tiempo inicial anterior y comprobamos que han pasado 10 segundos antes de hacer nada.
double totalTimeInSecs = (endTime - startTime) / 1000d;
if (totalTimeInSecs >= 10) {
```

Calculamos el delta en segundos y esperamos a que el mismo sea al menos de 10 segundos para empezar a tener datos concluyentes.

```
// Dividimos el numero de beats entre el entre los segundos.
double bps = (beats / totalTimeInSecs);
int dpm = (int) (bps * 60d);
```

Hacemos la conversión a **bps** y luego a **bpm** usando el delta anterior y el numero de latidos en este preciso instante.

```
// Si no esta en un rango entre 30 y 180 lo desechamos
if (dpm < 30 || dpm > 180) {
    startTime = System.currentTimeMillis();
    beats = 0;
    processing.set(false);
    return;
}
```

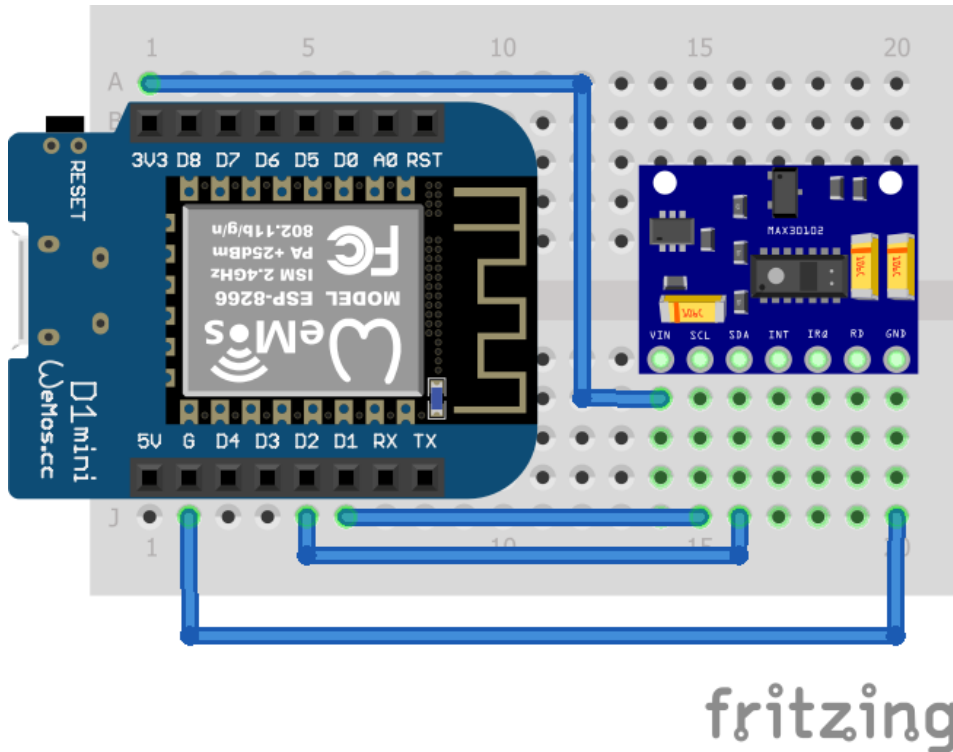
```
// Guardamos en un array los ultimos "beatsArraySize" valores.
beatsArray.add(dpm);

// Calculamos la media de los ultimos "beatsArraySize" valores.
int beatsAvg = beatsArray.calculaMedia();
onFrameChanged.onChangeed(beatsAvg):
startTime = System.currentTimeMillis();
beats = 0;
```

Tenemos un array circular de tamaño 4 guardando los últimos 4 bpm's registrados. Añadimos el nuevo, calculamos la media y ese será el valor que pasaremos al programa principal-

*Captura del pulso: Arduino*

# *Captura del pulso: Arduino*



El Arduino (en nuestro caso el WemosD1) se conecta con el sensor (MAX30100) mediante el protocolo I2C, que es un protocolo serie para interconexión de controladores.

Al ser Serial, solo hacen falta dos hilos para realizar la conexión (D1,D2). Los otros dos son 3.3v y tierra.

La alimentación le llega al Arduino mediante su conexión (micro) USB con el smartphone. Hace falta un adaptador OTG para poder conectarlo al smartphone.

# Captura del pulso: Arduino

```
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();

    // Creamos 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();
}
```

Como todo Sketch de Arduino, el programa consta de dos partes principales, la función `setup()`, y la función `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.

# Captura del pulso: Arduino

```
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);
            }
        }
    }
}
```

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.

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

# Captura del pulso: Arduino

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

Cuando todo esta 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 si, era mas 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.

{ pulse: “120” }



```
@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();
    }
}
```

# *¡Pruébelo usted mismo!*

Usuario: test@upna | Contraseña: test\_upna



## TensionApp 1.0.0 - ¡Primera Versión!

 hkfuertes released this 7 hours ago · 1 [commit](#) to master since this release

Primera version.

Sistema de gestion de mediciones de tension, peso y pulso, con introduccion manual, y medicion de pulso mediante la camara y un pequeño dispositivo arduino.


Mas informacion [aqui](#)

▼ Assets 4

 [TensionApp.arm.07.03.2020.apk](#)

 [TensionApp.arm64.07.03.2020.apk](#)

 [Source code \(zip\)](#)

 [Source code \(tar.gz\)](#)

Pruebe primero la versión arm64, y si esta no instala, entonces la versión arm.

Al no ser una aplicación firmada, deberá [habilitar orígenes desconocidos](#)

MB

MB