

MÁSTER EN INGENIERÍA BIOMÉDICA

**DISEÑO Y DESARROLLO DE UN SISTEMA DE APOYO A LA
DIAGNOSIS E IDENTIFICACIÓN DE PATOLOGÍAS
IMPLEMENTADO CON DISPOSITIVOS DE BAJO COSTE,
PRECISO, FIABLE Y DE USO EN ENTORNOS DOMICILIARIOS,
RESIDENCIALES Y HOSPITALARIOS.**

Oinatz Aspiazu Ituarte

Bilbao,
19 Septiembre 2022



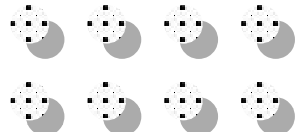
Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

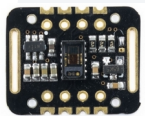
MEDIKUNTZA
ETA ERIZAINNTZA
FAKULTATEA
FACULTAD
DE MEDICINA
Y ENFERMERÍA

Motivación

- Monitorización activa de pacientes
- Detección de síntomas en personas vulnerables
- Datos extensos para investigación de patologías
- Datos precisos durante períodos prolongados
- Extensible a otros ámbitos y sensores
- Diseño basado en procesadores de bajo coste, con un sistema operativo y capaces de ejecutar múltiples programas a la vez



CONCEPTOS



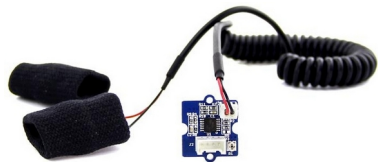
Pulsioxímetro

Dispositivo Max30102 que a través de un led rojo y un led infrarrojo, permite calcular la frecuencia cardíaca y el nivel de saturación de oxígeno en sangre



ADC

Dispositivo ADS1115 que actúa de convertidor de datos recogidos de manera analógica a digital

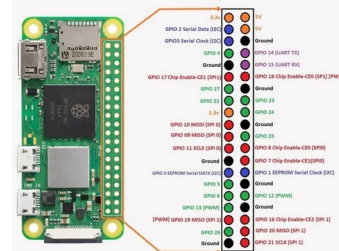


GSR

Mide la respuesta galvánica de la piel (sudoración)



Raspberry Pi



Mini ordenador de bolsillo encargado de recoger los datos de los sensores y enviarlos al servidor de Base de Datos remoto

Servidor Base de Datos

Base de Datos donde se encuentran los datos de los sensores, el ID del dispositivo y la fecha de recogida de los mismos



PostgreSQL

Servidor Web

Se realiza una página Web para mostrar los datos introducidos en la Base de Datos



Flask
web development,
one drop at a time

I2C Y LIBRERÍAS MAX30102 / ADS1115



“

Puerto y protocolo de comunicación serie
Dos líneas:

SCL → línea de los pulsos de reloj
que sincronizan el sistema

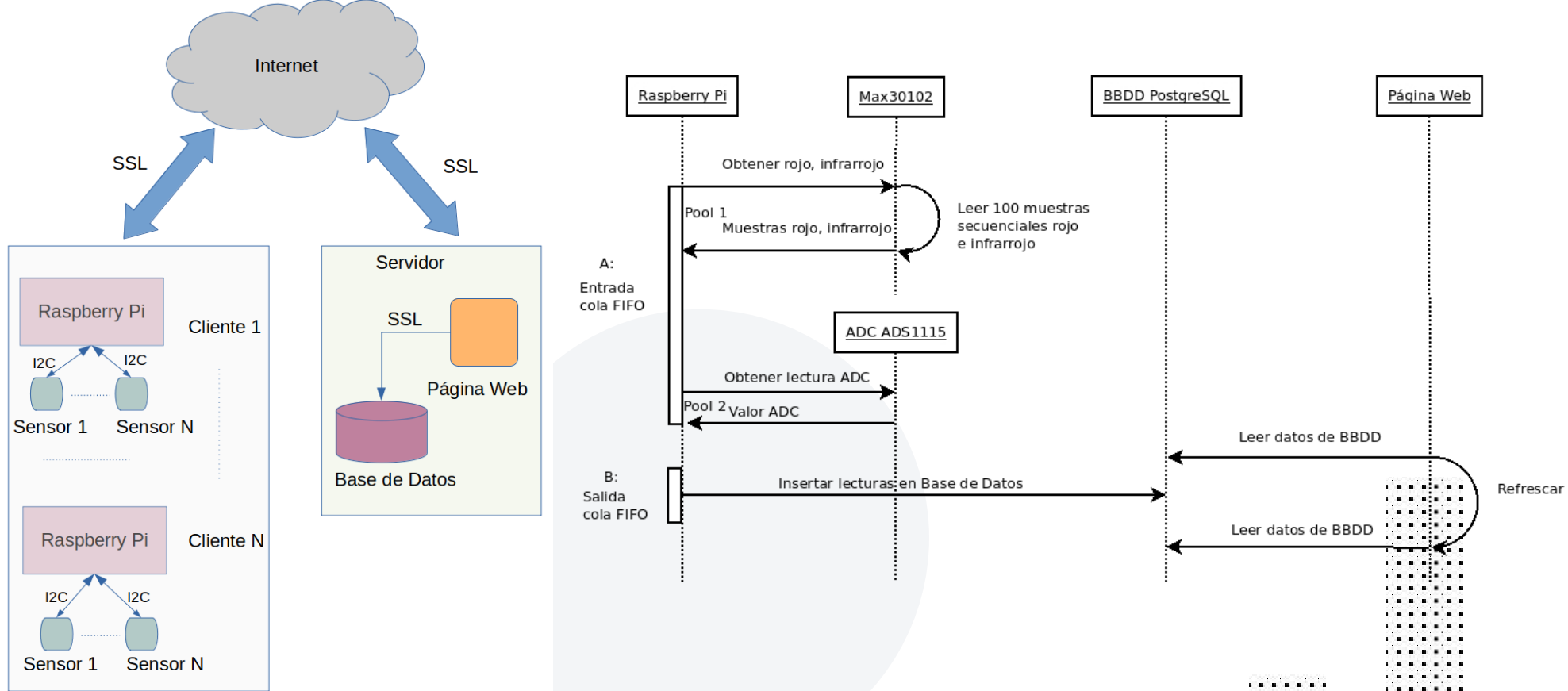
SDA → línea por la que se mueven
los datos entre los dispositivos ”

```
oinatz@raspberrypi:~ $ i2cdetect -l
i2c-1  i2c          bcm2835 (i2c@7e004000)      I2C adapter
i2c-21  i2c          Broadcom STB :      I2C adapter
i2c-20  i2c          Broadcom STB :      I2C adapter
oinatz@raspberrypi:~ $ i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:  -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- 57 -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
oinatz@raspberrypi:~ $
```

eman ta zabal zazu



Arquitectura



eman ta zabal zazu



Multiproceso



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

MEDIKUNTZA
ETA ERIZAINITZA
FAKULTATEA
FACULTAD
DE MEDICINA
Y ENFERMERÍA

01 Pool sensores

Presencia mínima de procesos
levantados

```
#generamos el buffer y lo llenamos el buffer con los datos de los sensores
def generar_buffer(q):

    # Obtenemos el ID, único por dispositivo
    id=str(obtener_id())

    ### Obtenemos valores rojo / infrarrojo en un procesador dedicado
    # Dado que disponemos de varios procesadores, vamos a aprovechar el multiproceso
    # dedicando uno de ellos a las lecturas de PDX y el otro al ADC. Con los datos de ambos, los introduciremos en una cola FIFO (Queue)

    while True:
        # Tenemos que usar pytz para sacar la fecha adecuada ya que datetime sólo trabaja con UTC.
        fecha = datetime.now(pytz.timezone("Europe/Madrid"))

        # Hay que crear la cola FIFO con 6 datos:
        # Usuario, rojo, infrarrojo, Pulso SPO2, sudoracion, fecha
        # De momento introducimos sudoracion como valor ficticio según el valor ficticio obtenido con el ADC
        # El pulso y spo2 corresponden realmente a los valores de rojo e infrarrojo. Se hace media de 100 valores de cada rojo e infrarrojo en cada línea

        # Sirviéndonos del multiproceso, utilizamos un pool de 2 procesos para obtener valores concurrentemente del max30102 y el ADC.
        pool = mpr.Pool(processes=2)

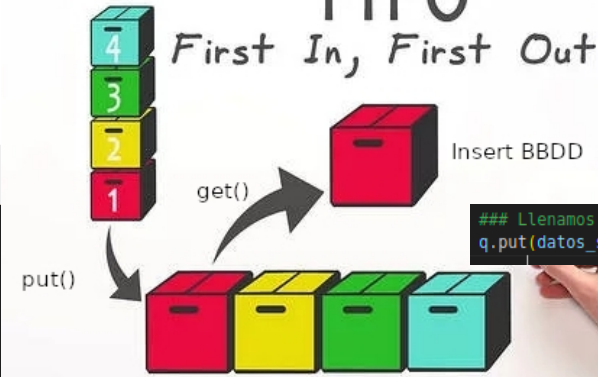
        resultado_async_max30102 = pool.apply_async(obtener_valores.coger_datos).get()
        resultado_async_adc = pool.apply_async(ADC.leer_adc).get()

        # resultado_async_max30102[0] tiene 100 valores de sensor rojo
        # resultado_async_max30102[1] tiene 100 valores de sensor infrarrojo
        # resultado_async_max30102[2] tiene 4 valores: Pulso obtenido de las 100 lecturas, true/false, SPO2 obtenido de las 100 lecturas y true/false.
        # Los valores de true/false son de pulso y spo2 para indicar si ha estado puesto el dedo y se han hecho lecturas.
        # resultado_async_adc tiene los valores recordados del ADC

        datos_sensores = [id, resultado_async_max30102[0], resultado_async_max30102[1], resultado_async_max30102[2], resultado_async_adc, fecha]
        # close() llama a destruir el pool y join() espera a los procesos trabajadores.
        pool.close()
        pool.join()

        ### Llenamos la cola FIFO con los datos de los sensores
        q.put(datos_sensores)
```

Datos sensores



02 Cola FIFO

```
### Llenamos la cola FIFO con los datos de los sensores
q.put(datos_sensores)

def insertar_sensores_bbdd_batch_queue(q):

    x=0
    ## Hacemos que se abra y cierre la conexión a la BBDD una única vez para todas las inserts.
    try:
        connection = psycopg2.connect(user="sensores",
                                       password=contraseña_bbdd.texto_descifrado,
                                       host="sensores.com",
                                       port="5432",
                                       database="sensores",
                                       sslmode = 'require',
                                       sslrootcert = '/home/pi/.certs/postgresql.crt')

        print('Lanzamos la query...')
        cursor = connection.cursor()
        #Recorremos el buffer línea a línea para insertarlas en la BBDD

        while True:
            # Sacamos cada una línea de la cola FIFO
            linea = q.get()

            # Por cada línea, sacamos ahora los elementos individuales
            usuario=str(linea[0])
            rojo=str(linea[1])
            infrarrojo=str(linea[2])
            pulso_spo2=str(linea[3])
            sudoracion=str(linea[4])
            fecha=str(linea[5])

            # Preparamos la query con la insert
            postgresSQL_insert_Query = "INSERT INTO sensores (usuario, rojo, infrarrojo, pulso_spo2, sudoracion, fecha) VALUES (%s,%s,%s,%s,%s,%s);)"
            # Insert con execute batch
            valores=[(usuario, rojo, infrarrojo, pulso_spo2, sudoracion, fecha)]
            psycopg2.extras.execute_batch(cursor, postgresSQL_insert_Query, valores)
            connection.commit()
            x+=1

            if linea is None:
                break

        except (Exception, psycopg2.Error) as error:
            print("Error obteniendo datos de la tabla de PostgreSQL", error)

    finally:
        # Cerramos la conexión a la BBDD
        if connection:
            cursor.close()
            connection.close()
            print("Cerrada la conexión a la BBDD\n")
```

apply_async() devuelve los valores inmediatamente después de que la ejecución se completa. Mantiene el orden de los resultados y soporta concurrencia.

Problemática / Inserts Base de Datos

max30102.read_sequential()

Lectura de 100 muestras secuenciales de led rojo y led infrarrojo → 1 única *insert* en Base de datos tiene ya 100 muestras de cada tipo de led

psycopg2.extras.execute_batch()

Fast execution helpers

The current implementation of executemany() is (using an extremely charitable understatement) not particularly performing. These functions can be used to speed up the repeated execution of a statement against a set of parameters. By reducing the number of server roundtrips the performance can be **orders of magnitude better** than using executemany().

```
def read_sequential(self, amount=100):
    """
    Función que lee el red led y el led ir 100 veces, dado el amount = 100
    Funciona como una función bloqueante
    """
    red_buf = []
    ir_buf = []
    for i in range(amount):
        while(GPIO.input(self.interrupt) == 1):
            # Sólo esperamos a la señal de interrupción, que indica que los datos están disponibles.
            pass

        red, ir = self.read_fifo()

        red_buf.append(red)
        ir_buf.append(ir)

    return red_buf, ir_buf
```



eman ta zabal zazu



```
postgreSQL_insert_Query = "INSERT INTO sensores (usuario, rojo, infrarrojo, pulso_spo2, sudoracion, fecha) VALUES (%s,%s,%s,%s,%s,%s);"
valores=[(usuario, rojo, infrarrojo, pulso_spo2, sudoracion, fecha)]
psycopg2.extras.execute_batch(cursor, postgreSQL_insert_Query, valores)
connection.commit()
```


Validaciones

1:18AM → 8:10AM (412 minutos) → 6123 Inserts.
1 Insert → 100 lecturas led rojo y 100 lecturas led ir → 612300 Inserts
→ 24,769 Inserts por segundo de cada tipo de led.

CPU

Consumos óptimos

Load average

```
top - 08:11:26 up 9:00, 3 users, load average: 1,00, 1,00, 1,00
Tasks: 202 total, 2 running, 200 sleeping, 0 stopped, 0 zombie
%Cpu(s): 24,5 us, 0,7 sy, 0,0 ni, 74,9 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 3838,9 total, 2543,2 free, 667,4 used, 628,3 buff/cache
MiB Swap: 100,0 total, 100,0 free, 0,0 used. 3110,6 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3146	oinatz	20	0	121592	29520	3780	R	43,2	0,8	0:01.30	python
3848	oinatz	20	0	121568	35936	10196	S	0,7	0,9	2:22.71	python
1362	oinatz	20	0	209160	47832	29820	S	0,3	1,2	0:40.46	node
3120	oinatz	20	0	11372	3044	2452	R	0,3	0,1	0:00.12	top
1	root	20	0	33868	8648	6772	S	0,0	0,2	0:03.70	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.04	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
8	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_wq
9	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_rude_
10	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_trace

RAM

```
oinatz@raspberrypi:~ $ free -m
oinatz@raspberrypi:~ $ free -m
              total        used        free      shared  buff/cache   available
Mem:           3838         666         2544          12          628         3111
Swap:              99              0              99
```



Página Web



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

MEDIKUNTZA
ETA ERIZAINNTZA
FAKULTATEA
FACULTAD
DE MEDICINA
Y ENFERMERÍA



Biosignals-SW

GICI-UPV-EHU

Esta página web muestra a modo de concepto los valores recogidos de los dispositivos MAX30102 y ADC ADS1115 en una Raspberry Pi, que han sido enviados a un servidor con una Base de datos PostgreSQL. Los valores se

localhost:3000/usuarios

← → ↺

localhost:3000/usuarios

🔍

🏠

🔖

📄

🔧

JSON

Datos sin procesar

Cabecebras

Guardar

Copiar

Contratar todo

Expandir todo

🔍

Filtrar JSON

❯ 0:

fecha:

"Tue, 23 Aug 2022 09:53:30 GMT"

✖ infarrrejo:

"13346, 1346, 1346, 1346, 1349, 1345, 1349, 1345, 1343, 1350, 1348, 1348, 1348, 1350, 1348, 1353, 1345, 1350, 1347, 1346, 1346, 1346, 1349, 1344, 1351, 1346, 1342, 1349, 1340, 1345, 1350, 1346, 1351, 1350, 1344, 1348, 1346, 1348, 1344, 1350, 1348, 1345, 1348, 1346, 1352, 1344, 1344, 1345, 1344, 1350, 1346, 1350, 1344, 1348, 1347, 1347, 1347, 1346, 1347, 1344, 1349, 1351, 1348, 1345, 1349, 1344, 1343, 1340, 1347, 1345, 1346, 1350, 1351, 1344, 1347, 1345, 1346, 1346, 1345, 1345, 1345, 1347, 1347, 1352, 1346, 1350, 1347, 1349, 1346, 1356, 1346, 1347, 1346, 1347, 1345, 1348, 1346, 1343, 1351, 1342, 1346]"

🔑 pulso_spo2:

"1", "-999", "False", "-999", "False", "1"

🔑 roj:

"13328, 1335, 1332, 1329, 1330, 1330, 1330, 1327, 1332, 1328, 1332, 1330, 1331, 1335, 1332, 1336, 1329, 1329, 1331, 1334, 1332, 1332, 1328, 1327, 1327, 1331, 1332, 1330, 1334, 1330, 1334, 1331, 1332, 1331, 1330, 1327, 1335, 1328, 1332, 1333, 1330, 1331, 1352, 1330, 1332, 1328, 1330, 1332, 1335, 1333, 1328, 1330, 1330, 1330, 1334, 1332, 1333, 1330, 1328, 1329, 1331, 1331, 1333, 1335, 1340, 1330, 1325, 1328, 1333, 1326, 1329, 1334, 1328, 1332, 1333, 1325, 1332, 1332, 1331, 1335, 1330, 1333, 1335, 1334, 1331, 1328, 1330, 1327, 1330, 1329, 1330, 1339, 1339, 1331, 1334, 1326, 1328, 1330, 1329]"

🔑 sueracion:

"4757"

🔑 usuario:

"e4:5f:01:34:7d:1c"

❯ 1:

fecha:

"Tue, 23 Aug 2022 09:53:42 GMT"

✖ infarrrejo:

"13350, 1348, 1346, 1351, 1346, 1350, 1349, 1342, 1345, 1341, 1349, 1348, 1352, 1349, 1346, 1349, 1341, 1349, 1348, 1346, 1344, 1352, 1342, 1349, 1347, 1349, 1353, 1343, 1344, 1350, 1343, 1352, 1348, 1348, 1348, 1346, 1341, 1350, 1348, 1349, 1350, 1343, 1345, 1341, 1345, 1346, 1339, 1346, 1340, 1350, 1343, 1348, 1345, 1343, 1348, 1345, 1343, 1347, 1349, 1344, 1342, 1356, 1346, 1347, 1344, 1348, 1342, 1345, 1346, 1348, 1340, 1351, 1348, 1345, 1349, 1347, 1348, 1337, 1356, 1363, 1360, 1356, 1359, 1357, 1363, 1400, 1579, 2178, 3435, 6038]"

🔑 pulso_spo2:

"1", "250", "True", "94, 129194", "True", "1"

🔑 roj:

"13306, 1332, 1339, 1337, 1336, 1329, 1333, 1329, 1333, 1336, 1329, 1333, 1330, 1336, 1334, 1330, 1331, 1332, 1329, 1335, 1330, 1336, 1330, 1333, 1334, 1333, 1327, 1325, 1325, 1325, 1333, 1327, 1329, 1329, 1330, 1333, 1348, 1330, 1335, 1335, 1335, 1334, 1333, 1339, 1332, 1333, 1333, 1337, 1332, 1328, 1329, 1329, 1333, 1335, 1331, 1339, 1333, 1333, 1331, 1331, 1333, 1337, 1327, 1329, 1335, 1331, 1329, 1325, 1336, 1335, 1329, 1328, 1320, 1329, 1330, 1339, 1334, 1329, 1331, 1330, 1321, 1319, 1337, 1358, 1361, 1359, 1369, 1473, 1705, 2137, 2911, 3932, 7138]"

🔑 sueracion:

"4713"

🔑 usuario:

"e4:5f:01:34:7d:1c"

❯ 2:

fecha:

"Tue, 23 Aug 2022 09:53:46 GMT"

✖ infarrrejo:

"112204, 23394, 51540, 63808, 69408, 76789, 81861, 84381, 88610, 93449, 96133, 96551, 101133, 101495, 96699, 99520, 105818, 109366, 122719, 114362, 116326, 117548, 120807, 124531, 126367, 126294, 125652, 126426, 127894, 128166, 128926, 127771, 127981, 128137, 128212, 128315, 128574, 128792, 128870, 128576, 128366, 128427, 128490, 128543, 128580, 128422, 128548, 128425, 128358, 128407, 128353, 128287, 128282, 128381, 128401, 128374, 128347, 128290, 128200, 128183, 128166, 128177, 128126, 128051, 128012, 127987, 127968, 127949, 127923, 127873, 127866, 127798, 127714, 127616, 126801, 126829, 127998, 128017, 128004, 127986, 128003, 127985, 127971, 127941, 127939, 127941, 127931, 127941, 127953, 128018, 128059, 128015, 128012, 127985, 128016, 127924, 127928, 127925, 127925, 127929]"

🔑 pulso_spo2:

"1", "-999", "True", "-999", "False", "1"

🔑 roj:

"123420, 22182, 45591, 56720, 61556, 70390, 76456, 80405, 86046, 92903, 96612, 96885, 98786, 94679, 91325, 98048, 107993, 113580, 118873, 121004, 124304, 126224, 1

```
app = Flask(__name__)

@app.get('/usuarios/')
def get_users():
    conn = crear_conexion()
    cur = conn.cursor(cursor_factory=extras.RealDictCursor)
    cur.execute("SELECT * FROM sensores")
    usuarios = cur.fetchall()
    cur.close()
    conn.close()
    return jsonify(usuarios)

@app.get('/usuarios/<usuario>')
def get_user(usuario):
    conn = crear_conexion()
    cur = conn.cursor(cursor_factory=extras.RealDictCursor)
    cur.execute("SELECT * FROM sensores where usuario = %s", (usuario,))
    usuario = cur.fetchall()
    cur.close()
    conn.close()

    if usuario is None:
        return jsonify({'message': 'Usuario no encontrado'}), 404

    return jsonify(usuario)

@app.get('/')
def home():
    return send_file('static/index.html')

if __name__ == '__main__':
    app.run(debug=True, port=3000)
```

Otros

Cifrado password y SSL

Obtener_ID()



Contraseña cifrada con *hash*



```
from cryptography.fernet import Fernet
clave = b'pRmgMa8T0INjEAFksaq2aafzoZXEuwKI7wDe4c1F8AY='
cipher_suite = Fernet(clave)
texto_cifrado = b'gAAAAABiJ4zo9YDKl2YmvwJMqXWRTa3GQZoTRCKxnEk3M7xNtTulKK-12qhzyp_LNzWqWHrw_BNarWBvPd4y4y5d3cgLxIdnQ=='
texto_descifrado = (cipher_suite.decrypt(texto_cifrado)).decode("utf-8") # la suite nos devuelve un literal byte y necesitamos un string
```

```
## Introducimos una rutina previa para obtener un ID unico por placa, aprovechando que cada dirección MAC de la WLAN es única por dispositivo
## Esto nos sirve como identificador unico de dispositivo y por tanto de usuario para poder hacer luego las Insert en la BBDD
import subprocess

def obtener_id():
    string="ip addr show wlan0 | grep ether| awk '{print $2}'"
    ID=subprocess.getoutput(string)
    return(ID)
```

SSL + IPs autorizadas

```
# - SSL -

ssl = on
#ssl_ca_file = ''
#ssl_cert_file = '/etc/ssl/certs/ssl-cert-snakeoil.pem'
ssl_cert_file = '/var/lib/postgresql/data/server.crt'
#ssl_crl_file = ''
#ssl_key_file = '/etc/ssl/private/ssl-cert-snakeoil.key'
ssl_key_file = '/var/lib/postgresql/data/server.key'
```

ID por MAC

eman ta zabal zazu



Conclusiones

Se consiguen los siguientes objetivos:

- Monitorización activa continua y precisa
- Uso de dispositivos y sensores de bajo coste
- Uso de multiproceso para realizar lecturas de varios tipos de sensores de manera concurrente (*pool*)
- Uso de multiproceso para lecturas de datos de sensores e inserciones en Base de Datos concurrentes (cola *FIFO*)
- Implementación de ADC
- Inserciones en Base de Datos de manera óptima (*psycopg2.extras.execute_batch()*)
- Cada inserción en Base de Datos lleva 100 lecturas de led rojo y 100 de led infrarrojo (Una inserción equivale a los datos de 100 inserciones de cada tipo de led)
- Seguridad y cifrado
- Página Web
- Validaciones



Preguntas y respuestas



Gracias

eman ta zabal zazu



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

MEDIKUNTZA
ETA ERIZAINNTZA
FAKULTATEA
FACULTAD
DE MEDICINA
Y ENFERMERÍA

