Universidad de Alcalá

Escuela Politécnica Superior

Universidad de Alcalá

Computación ubicua

PECL2 – Especificación del proyecto

Viernes 15:00 – 17:00

Grado en Ingeniería Informática – Curso 2019/2020

Eduardo Graván Serrano – 03212337L

Marcos Barranquero Fernández – 51129104N

Adrián Montesinos González – 51139629A

Jorge Guillamón Brotóns – 492266337N

ÍNDICE

INTRODUCCIÓN

El sueño es una actividad importante para nuestro cuerpo, y en muchas ocasiones el ritmo de vida que llevamos dificulta la creación de hábitos de sueño saludables. Esto puede derivar en problemas tales como no rendir lo suficiente a lo largo del día, afectar a nuestra salud mental, incrementa el riesgo de accidentes, y limitar la capacidad cognitiva.

Por todo ello, se ha fabricado un sistema capaz de monitorizar las horas de sueño que se pasan en la cama. El objetivo es facilitar la creación de hábitos saludables de horas de sueño mediante la adquisición y procesamiento de datos, extracción del conocimiento de estos y generar consejos e informes sobre cómo aumentar el confort y buenos hábitos de nuestras horas de sueño.

El sistema está compuesto de cuatro partes diferenciables:

1. Ordenador o microcontrolador base que interconecta todos los elementos.
2. La infraestructura que permite la recolección de datos mientras se duerme
3. La base de datos que almacena los datos recogidos y los informes procesados.
4. El script o algoritmo que procesa los datos y genera los informes.
5. El servidor web local empleado para mostrar dichos informes al usuario.

Se pasa a describir los siguientes apartados individualmente.

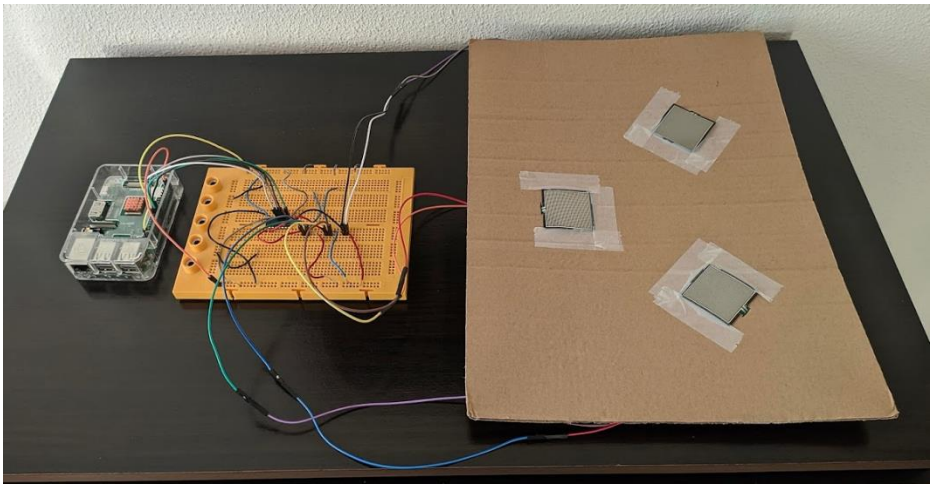
ORDENADOR BASE

Se ha empleado una **Raspberry Pi 3B+** con Raspbian instalado como SSOO. La Raspberry es la encargada de ejecutar los scripts, procesar los datos generando los informes y mantener el servidor web.

Para ello, se apoya en **Python3** con los módulos de **Flask** para el servidor web, **Bluepy** para conectarse a la Mi Band por bluetooth, y **AdaFruit** y **SPI** para la interfaz de lectura de los sensores de presión.

Nota: para visualizar los informes se debe ejecutar el servidor y acceder desde la versión web.

Adjuntamos una foto de la arquitectura:

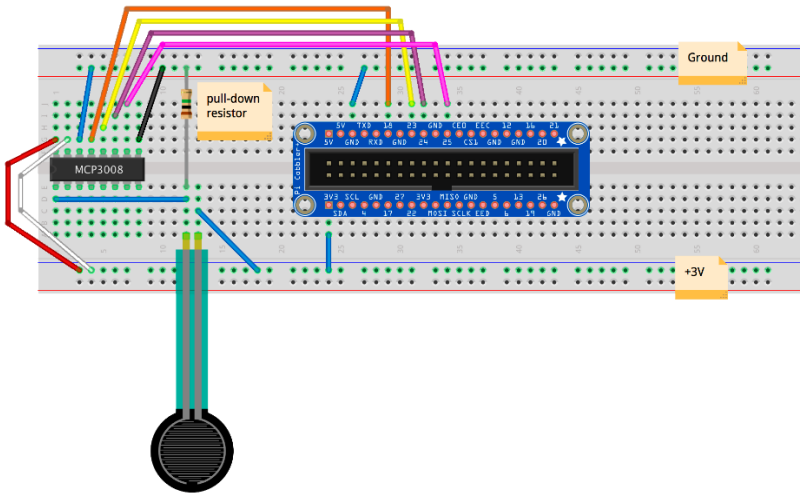


RECOLECCIÓN DE DATOS

La recolección de datos se realiza con los sensores de presión y el de pulsaciones de la Mi Band. Una vez lanzado el script, se escriben las lecturas asíncronas de estos sensores cada 3 segundos.

SENSORES DE PRESIÓN

La lectura de los datos con sensores de presión se basa en la siguiente arquitectura:



Los componentes son los siguientes:

- **Sensor de presión** que fluctúa su resistencia en función del peso que recibe. Se emplea una resistencia de 10k Ohmnios. Se han utilizado 3 de estos sensores.
- **MCP3008** como conversor analógico-digital, ya que las señales del sensor de presión son analógicas y la raspberry solo lee señales digitales.

Sin embargo, se han utilizado cables para conectar directamente la Raspberry a los pines del MCP3008.

Para la lectura de los sensores se han escrito los siguientes métodos:

```
import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
from time import sleep

def inicializar():
    # Creo bus SPI
    spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)

    # Cargo Chip Select
    cs = digitalio.DigitalInOut(board.D5)

    # Creo el objeto MCP asociado
    mcp = MCP.MCP3008(spi, cs)
    # y lo devuelvo
    return mcp

def get_canales(mcp):
    # creo canales
    canal1 = AnalogIn(mcp, MCP.P0)
```

```
    canal2 = AnalogIn(mcp, MCP.P1)
    canal3 = AnalogIn(mcp, MCP.P2)
    canales = [canal1, canal2, canal3]
    return canales

def get_valor(numero_canal, canales):
    return canales[numero_canal].value
```

SENSOR DE PULSACIONES

Para leer las pulsaciones, nos hemos basado en un script ya creado por el usuario *yogeshojha*. Su script original estaba escrito en Python2, y lo hemos traducido a Python3 para poder unificarlo con el resto de los scripts. También se ha recortado la funcionalidad para utilizarse de forma más cómoda.

Se han añadido los siguientes métodos en el script encargado de leer los datos de la mi band:

```
def inicializar_mi_band():
    MAC_ADDR = "D1:51:75:37:49:B4"
    print('Attempting to connect to ', MAC_ADDR)

    band = MiBand3(MAC_ADDR, debug=True)
    band.setSecurityLevel(level = "medium")
    band.authenticate()
    return band

def iniciar_lectura_bpm(band):
    print("Iniciando lectura bpm")
    heart_beat(band)

def get_bpm():
    return ultimo_bpm
```

Cabe destacar que el sensor está limitado a realizar una actualización de los BPM cada 3 segundos, por lo que ese será nuestro tiempo de muestreo para todos los sensores.

Es interesante comentar que se ha modificado el script de lectura de pulsaciones para detenerse si el usuario se quita la pulsera:

```
            # ...
            # stop heart monitor continues & manual
            char_ctrl.write(b'\x15\x02\x00', True)
            char_ctrl.write(b'\x15\x01\x00', True)
            # enabling accelerometer & heart monitor raw data notifications
            char_sensor.write(b'\x01\x03\x19')
            # enablee heart monitor notifications
            char_d.write(b'\x01\x00', True)
            # start hear monitor continues
            char_ctrl.write(b'\x15\x01\x01', True)
            char_sensor.write(b'\x02')
            t = time.time()
            # leo mientras el usuario no se quite la mi band
            while (self.ultimo_bpm != 0):
                self.waitForNotifications(0.5)
                # leo y transformo el bloque bluetooth
                self._parse_queue()
                # send ping request every 12 sec
                if (time.time() - t) >= 12:
                    char_ctrl.write(b'\x16', True)
                    t = time.time()
```

```
            # detengo modo lectura
            self.stop_realtime()
            # finalizo ejecución
            sys.exit()
```

LECTURA Y ESCRITURA EN BBDD

Finalmente, se tiene un método principal encargado de inicializar las lecturas de sensores y escribir en la base de datos cada 3 segundos:

```
from lector_miband import *
from lector_presion import *
from threading import Thread
from Consultas import *
import sys
# Inicializo lector de presión
mcp = inicializar()

canales = get_canales(mcp)
# Inicializo mi band en un hilo aparte porque la lectura es bloqueante
band = inicializar_mi_band()
lectura = Thread(target=iniciar_lectura_bpm, args=[band])
lectura.start()

# Cargo bbdd
bbdd = BaseDatos()

# La mi band tarda un poco en arrancar, así que hasta que no lo haga no empezamos a leer
...
while(band.ultimo_bpm < 0):
    sleep(5)

# Una vez cargada, obtengo nueva id de sesion
try:
    id_sesion = bbdd.getUltimaSesion() + 1
except:
    id_sesion = 1

while(band.ultimo_bpm != 0):
    presion1 = get_valor(0, canales)
    presion2 = get_valor(1, canales)
    presion3 = get_valor(2, canales)
    bbdd.insertarLectura(presion1, presion2, presion3, band.ultimo_bpm, id_sesion)
    sleep(3)
print("El usuario se ha quitado la pulsera. Fin de la recogida de datos. ")
```

Cabe reseñar que la ejecución se detiene cuando el usuario se quita la pulsera, y por tanto las pulsaciones pasan a ser 0.

MODELO DE DATOS

De cara al almacenamiento permanente de los datos, se ha decidido utilizar el motor SQLite, ya que tenemos un sistema de base de datos bastante sencillo y ligero, por lo que SQLite se adapta perfectamente a nuestras necesidades, liberando de carga a la Raspberry Pi.

ARQUITECTURA

Se cuenta con, únicamente, dos tablas en nuestra base de datos:

- **LecturasSensores:** en ella se almacenan los datos de presión y ritmo cardiaco leídos durante uno de los sondeos que haga el sistema. Se almacena también la fecha (timestamp) del momento en que se recogieron estos datos para poder trabajar con ellos más adelante, y un número de sesión que determina a qué sesión de sueño corresponde cada uno de estos datos. Para cada sesión de sueño, todas las tuplas de la base de datos con diferentes lecturas de sensores compartirán este ID de sesión.
- **Informes:** en esta tabla se almacenan las rutas a los archivos HTML generados que guardan los informes una vez ya se han procesado todos los datos para una sesión dada. Para ello, se almacenan también la fecha de generación de este informe, y el número de sesión correspondiente.

En ambos casos, la columna de fecha es la Primary Key y no se permiten valores nulos en ninguna de las columnas.

En cuanto al tratamiento de las fechas, SQLite no cuenta con un tipo específico para la gestión de fechas, por lo que se puede escoger entre tres tipos para almacenar este tipo de datos. El tipo "texto" almacena la fecha en formato legible directamente, pero dificulta el trabajo con las fechas; el tipo "real" almacena números siguiendo el calendario juliano; y por último el tipo "integer" almacena los números usando el timestamp en tiempo de Unix.

Para ambas tablas, debido a que se consideraba que sería mas fácil trabajar con ello, no solo desde la propia base de datos, sino también desde Python, se ha utilizado el tipo "integer". SQLite cuenta también con varias funciones integradas para hacer la conversión a un formato de fecha y hora legibles, por lo que la gestión de timestamps tampoco se hace cuesta arriba. En nuestro caso, la función "datetime(columna, 'unixepoch')", nos transformará el timestamp en formato tiempo Unix a un formato legible.

CREACIÓN DE BDD Y TABLAS

Con todo esto, nos queda el siguiente archivo SQL para la generación de la base de datos:

```
drop table if exists Informes;
drop table if exists LecturasSensores;

create table LecturasSensores (
    fecha integer not null PRIMARY KEY,
    presion1 integer not null,
```

```
    presion2 integer not null,
    presion3 integer not null,
    pulsaciones integer not null,
    sesion integer not null
);
create table Informes (
    fecha integer not null PRIMARY KEY,
    datos text not null,
    sesion integer not null
);
```

Para no tener que crear la base de datos manualmente, se ha creado un script que hace la conexión con SQLite desde Python. Este script se encarga de ejecutar las consultas almacenadas en este archivo SQL, creando la base de datos si no existe, y creando las tablas para esta base de datos.

El script es el siguiente:

```
import sqlite3

################ Configuracion ####################
# Nombre de la base de datos
DB_NAME = "datosPECL"

# Archivo SQL con la definicion de las tablas
SQL_File_Name = "CrearTablas.sql"
###################################################

# Se carga el archivo SQL a una variable y se eliminan los saltos de linea
TableSchema=""
with open(SQL_File_Name, 'r') as SchemaFile:
    TableSchema=SchemaFile.read().replace('\n', '')

# Se crea la nueva base de datos
conn = sqlite3.connect(DB_NAME)
curs = conn.cursor()

# Se lanza la consulta de creacion de tablas
sqlite3.complete_statement(TableSchema)
curs.executescript(TableSchema)

# Se cierra la conexion con la base de datos
curs.close()
conn.close()
```

Debido a las primeras dos sentencias SQL del archivo de creación de tablas, si se decide ejecutar este script cuando ya existen las tablas, éstas serán borradas y creadas de nuevo. De esta forma, se puede usar el script para hacer un reset total de los datos de la base de datos.

CONSULTAS A LA BDD

Se ha creado una clase en Python para la gestión de las consultas a la base de datos.

Esta clase se encarga de conectarse a la base de datos SQLite, y realizar todas las consultas. Hace de interfaz de conexión entre el resto de la aplicación y la base de datos.

La clase tiene un único atributo privado, que es el objeto que representa la conexión con la base de datos. Este objeto se instancia desde el constructor y no se vuelve a modificar:

```
def __init__(self):
        self.__conexion = sqlite3.connect("datosPECL")
```

Una vez tenemos la conexión abierta, podemos empezar a trabajar con la base de datos.

Tenemos dos métodos para insertar cada uno de los valores a la base de datos, uno para informes, y otro para las lecturas de los sensores:

```
def insertarLectura(self, presion1, presion2, presion3, pulsaciones, sesion):
    """
        Inserta una nueva lectura en la base de datos con los datos
        de presiones y pulsaciones que se le pasan como parametro
    """
    try:
        cursor = self.__conexion.cursor()
        cursor.execute("INSERT INTO LecturasSensores VALUES(?, ?, ?, ?, ?, ?)
", [int(time.time()), presion1, presion2, presion3, pulsaciones, sesion])
        self.__conexion.commit()
        cursor.close()
    except sqlite3.Error as error:
        print("Error al insertar: ", error)

def insertarInforme(self, texto, sesion):
    """
        Inserta un informe nuevo con el texto que se le pasa
        como parametro
    """
    try:
        cursor = self.__conexion.cursor()
        cursor.execute("INSERT INTO Informes VALUES(?, ?, ?)", [int(time.time
()), texto, sesion])
        self.__conexion.commit()
        cursor.close()
    except sqlite3.Error as error:
        print("Error al insertar: ", error)
```

Estos métodos están pensados para pasarle los datos que recojan los sensores como parámetro para que sean insertados en la base de datos. El timestamp lo realiza Python a través la función time() de la librería time.

El único problema es que, al realizar esta consulta al tiempo de Unix, se consigue un valor para la franja horaria GMT+0, por lo que, al recuperar los datos en consultas posteriores, se deberá sumar 1 hora a cada uno de los datos para que se encuentren en nuestra franja horaria.

En cuanto a las consultas para recuperar datos, se tienen 4 consultas preparadas con las que se cubren todas las necesidades para la aplicación.

En primer lugar, para recuperar los datos de los sensores y poder trabajar con ellos, el método de consulta recibe como parámetro un número de horas como integer. Este número de horas representa la duración de la sesión de sueño. Con este número, se multiplica por 3600 para obtener el número de segundos y se recuperan todos los datos cuyo timestamp sea mayor que la fecha resultante de restarle ese número de segundos al tiempo actual.

```
def consultarLecturas(self, horas):
    """
        Recupera las lecturas de todos los sensores guardadas en las ultimas
horas
        Las horas se le pasan como parametro
    """
    try:
        cursor = self.__conexion.cursor()
        # Se le suma 1 hora para que este en nuestra franja horaria
        cursor.execute("SELECT datetime(fecha + 3600, 'unixepoch'), presion1,
 presion2, presion3, pulsaciones, sesion FROM LecturasSensores WHERE fecha >=
 " + str(int(time.time() - 3600 * horas)))
        self.__conexion.commit()
        datos = cursor.fetchall()
        cursor.close()

        return datos

    except sqlite3.Error as error:
        print("Error al consultar la base de datos: ", error)
```

Tenemos también dos métodos encargados de recuperar informes de la base de datos. Contamos con un método que devuelve únicamente el último informe que haya sido insertado, y otro que nos devuelve todos los informes ordenados de más reciente a más antiguo. Puesto que ambos métodos son iguales salvo por la cláusula LIMIT 1, solo se presenta uno de ellos en este documento.

```
def consultarUltimoInforme(self):
        """
            Devuelve el ultimo informe generado
        """
        try:
            cursor = self.__conexion.cursor()
            # Se le suma 1 hora para que este en nuestra franja horaria
            cursor.execute("SELECT datetime(fecha + 3600, 'unixepoch'), datos
, sesion FROM Informes ORDER BY fecha DESC LIMIT 1")
            self.__conexion.commit()
            datos = cursor.fetchall()
            cursor.close()

            return datos

        except sqlite3.Error as error:
            print("Error al consultar la base de datos: ", error)
```

Por último, de cara al establecimiento de sesiones de sueño, se cuenta con un método que consulta la tabla LecturasSensores y encuentra el valor máximo para la sesión. Este valor se incrementará en uno para hacer inserciones para sesiones nuevas.

El método es el siguiente:

```
def getUltimaSesion(self):
    """
    Devuelve el id de la última sesión registrada.
    """
    try:
        cursor = self.__conexion.cursor()
        cursor.execute("SELECT MAX(sesion) FROM LecturasSensores")
        self.__conexion.commit()
        datos = cursor.fetchall()
        cursor.close()
        id_sesion = datos[0][0]
        if(id_sesion is None):
            id_sesion=0
        return id_sesion
    except sqlite3.Error as error:
        print("Error al consultar la base de datos: ", error)
```

FUNCIONAMIENTO

Para probar el correcto funcionamiento de la base de datos, se ha desarrollado el siguiente script que prueba todas las consultas de la base de datos:

```
# Tests
if __name__ == "__main__":
    b = BaseDatos()
    b.insertarLectura(300, 450, 7500, 65,1)
    sleep(1)
```

```
b.insertarLectura(600, 480, 17500, 80,2)
sleep(1)
b.insertarLectura(0, 7500, 9500, 59,3)
b.insertarInforme("Muy bueno",1)
sleep(1)
b.insertarInforme("No tan bueno",2)
sleep(1)
b.insertarInforme("Muy bueno",3)
for i in b.consultarLecturas(2):
    print(i)
for i in b.consultarInformes():
    print(i)
print(b.getUltimaSesion())
```

El resultado de la ejecución de este script con una base de datos vacía es el siguiente:

```
pi@raspberrypi:~/Desktop/PECL Ubicua/Scripts $ 
('2020-01-08 14:04:06', 300, 450, 7500, 65, 1)
('2020-01-08 14:04:07', 600, 480, 17500, 80, 2)
('2020-01-08 14:04:08', 0, 7500, 9500, 59, 3)
('2020-01-08 14:04:10', 'Muy bueno', 3)
('2020-01-08 14:04:09', 'No tan bueno', 2)
('2020-01-08 14:04:08', 'Muy bueno', 1)
3
```

Como podemos ver, los timestamp se han transformado a un formato legible y el resultado de la consulta se hace en forma de tuplas, lo que nos permite coger los datos que necesitemos en cada momento.

TRATAMIENTO DE DATOS

Una vez obtenidos los datos estos se hacen pasar a través de una serie de algoritmos diseñados para puntuar cualitativamente el sueño del usuario.

DATOS DE ENTRADA

Los datos que hemos tomado en consideración a la hora de generar el informe del sueño son fundamentalmente 3: la frecuencia cardíaca, los cambios de postura del usuario durante el sueño y el tiempo de sueño.

Puede encontrar la implementación de todos estos algoritmos en informe.py.

FRECUENCIA CARDÍACA

Es un hecho que la FC representa en buena medida la actividad del organismo, en tal efecto hemos empleado esta medida para ver si el usuario está realmente durmiendo o simplemente está quieto sobre los sensores de presión.

Una frecuencia cardíaca demasiado alta se puede deber tanto a la falta de forma física como a la falta de sueño, así se lo hará saber al usuario en caso de detectarla.

También se ha usado la frecuencia cardíaca como medio para calcular si un usuario está durmiendo.

CAMBIO DE POSTURA

Los sensores de presión no devuelven si se ha producido un cambio en la distribución de presión, en consecuencia, hemos implementado un algoritmo que detecta si un usuario ha cambiado de postura durante el sueño. Su pseudocódigo es:

```
bool ha_cambiado_de_postura(int lectura_anterior[3], int lectura_actual[3]){
    bool cambio = false;
    for(int i = 0; i < 3; i++)
        razon = lectura_actual/lectura_anterior;
        if(razon > 1.2 || razon < 0.8) cambio = true;

    return cambio;
}
```

Básicamente considera que el usuario ha cambiado de postura si se produce un cambio de al menos el 20% en la lectura de cualquiera de los sensores.

Si se ha movido demasiadas veces durante la noche se puede concluir que ha tenido un sueño agitado y no ha reposado adecuadamente, lo que se verá reflejado en el informe.

MINUTOS DE SUEÑO

De acuerdo con la Sociedad Española del Sueño la cantidad de sueño efectivo repercute en gran medida en la salud general del organismo.

La SES recomienda dormir como mínimo 6h al día y 8h como cantidad recomendable de sueño.

Las condiciones que se han empleado a la hora de medir el sueño son:

- Que la frecuencia cardíaca descienda por debajo de 70 bpm
- Que alguno de los sensores de presión esté activado

Si alguno de estos no está activado se considera que el usuario no está durmiendo.

El tiempo de sueño se va aumentando cada vez que se recibe una medida que cumple las condiciones anteriormente descritas.

GRÁFICOS

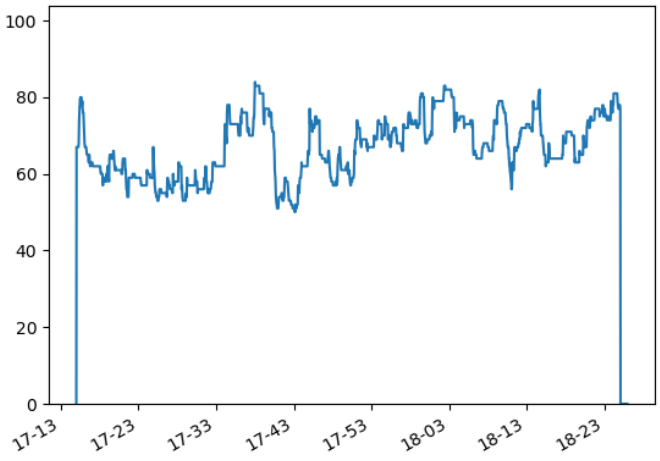
Finalmente, se emplea la librería **MatPlotLib** para dibujar los distintos gráficos que representan movimientos, pulsaciones, etc:

Un ejemplo podría ser el gráfico de las pulsaciones por minuto a lo largo de la noche:

```
def generarGraficoBpm(idSesion):
    bbdd = BaseDatos()
    datos = bbdd.getPulsacionesSesion(idSesion)
    fechas = []
    valores = []

    for dato in datos:
        fecha = dato[0]
        fecha_dt = dt.datetime.strptime(fecha, "%Y-%m-%d %H:%M:%S")
        fechas.append(fecha_dt)
        valores.append(dato[1])

    now = fechas[-1]
    then = fechas[0]
    rango = mdates.drange(then, now, dt.timedelta(seconds=1))
    y = rellenar_huecos(valores, rango)

    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%H-
%M'))
    plt.gca().xaxis.set_major_locator(mdates.MinuteLocator(interval
=10))
    plt.gca().set_ylim([0, max(y)+20])
    plt.plot(rango, y)
    #plt.show()
    plt.gcf().autofmt_xdate()

    fecha_sesion = fechas[int(len(valores)/2)]
```

```
    fecha_sesion = fecha_sesion.timetuple()
    ano = str(fecha_sesion.tm_year)
    mes = fecha_sesion.tm_mon
    if(mes < 10):
        mes=str(0)+str(mes)
    else:
        mes=str(mes)
    dia = fecha_sesion.tm_mday
    if(dia < 10):
        dia=str(0)+str(dia)
    else:
        dia=str(dia)

    fecha = (ano+"-"+mes+"-"+dia+"")

    plt.savefig("static/images/Sesion_"+str(idSesion)+"_"+fecha+"_"
+"bpm")
    plt.cla()
    return fecha
```

Generando el siguiente gráfico (en función de los datos):



CONCLUSIÓN

La aplicación valorará tres aspectos sobre el sueño del usuario en base a las características anteriores.

Ej:

```
Has dormido demasiado poco. (tiempo)
Tu sueño ha sido agitado, te has movido mucho. (cambios de postura)
```

```
No has entrado en sueño profundo, deberías hacer más ejercicio también. (bpm)
```

Además, se aprovecharán los datos recogidos de estos algoritmos para representarlos gráficamente y poder observar una evolución del sueño.

SERVIDOR WEB

Se ha utilizado el framework Flask de Python3 para el desarrollo del servidor web.

Esto nos ha posibilitado un desarrollo rápido y sencillo de la parte front-end de la aplicación, gracias al lenguaje de plantillas Jinja usado por Flask. Además, al haber escrito el servidor en Python, la totalidad de nuestra aplicación está codificada en Python, haciendo que el mantenimiento y expansión posterior sea mucho más fácil.

Nuestra aplicación web tiene dos partes:

- **Página índice:** desde la cual se nos presenta una tabla que recoge todos los informes generados por la aplicación. Si hacemos click en alguno de estos informes, podremos acceder a ellos para revisarlos.
  Esta página es generada dinámicamente en base a la respuesta que reciba el script de Python de la base de datos.
- **Páginas de los informes:** en las cuales se nos presenta la información asociada a cada sesión de sueño. Estas páginas HTML son generadas dinámicamente desde Python, pero no por Flask. Flask simplemente se encarga de servir estas páginas, no de generarlas.

El código del servidor es el siguiente:

```
from flask import Flask, render_template, url_for
from Consultas import BaseDatos
import sqlite3

app = Flask(__name__)

@app.route('/')
def index():
    bbdd = BaseDatos()
    informes = bbdd.consultarInformes()
    return render_template('index.html', tasks=informes)

@app.route('/<string:page_name>/')
def render_static(page_name):
    return render_template(page_name + ".html")

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0')
```

El primer método es el utilizado al acceder al índice de la aplicación, como podemos ver se hace una consulta a la base de datos y se pasa la información de los informes al método de renderización.

El segundo método es el utilizado cuando la página web intenta acceder a elementos estáticos (páginas de los informes).

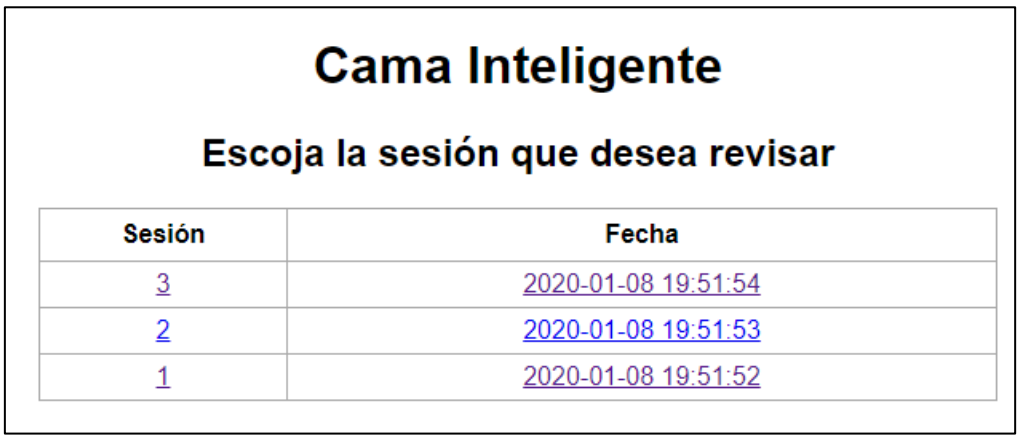
PÁGINA ÍNDICE

La página índice tiene el siguiente código HTML:

```
<!doctype html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Cama Inteligente</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css'
) }}">
    <link rel="icon" href="data:;base64,iVBORw0KGgo=">

</head>
<body>
    <div class="content">
        <h1>Cama Inteligente</h1>
        <h2>Escoja la sesión que desea revisar</h2>
        <div>
            <table>
                <tr>
                    <th>Sesión</th>
                    <th>Fecha</th>
                </tr>
                {% for task in tasks %}
                <tr>
                    {% set informe = "Sesión_" + task[2]|string + "_" + task[
0].split(" ")[0] %}
                    <td><a href={{informe}}>{{task[2]}}</a></td>
                    <td><a href={{informe}}>{{task[0]}}</a></td>
                </tr>
                {% endfor %}
        </table>
        </div>
    </div>
</body>
</html>
```

Como podemos apreciar, hay parte de la página que usan el lenguaje de plantillas Jinja. Estas se han usado para la generación dinámica de código HTML en base al resultado de la consulta a la base de datos realizada en el servidor.

Un posible resultado de esta página web es:



Simplemente tendríamos que hacer click en alguno de estos elementos para ir al informe correspondiente.

PÁGINAS INFORME

Estas páginas son generadas por la clase Python 3 "PaginaHTML" que se encuentra dentro del archivo "GenerarHTML.py".

Esta clase se instancia cuando se quiera construir un nuevo informe en formato HTML. Simplemente tenemos que llamar al método "crearFichero" pasándole los parámetros necesarios para construir el informe.

La página web generada se crea dentro del directorio "templates", lo cual es necesario para que Flask pueda servirla.

El código Python de esta web es el siguiente:

```
import textwrap
import os
from Funcionalidad import *
from Consultas import *

def getFechaHoy():
    fecha = datetime.datetime.now()
    ano = fecha.year
    mes = fecha.month
    dia = fecha.day

class PaginaHTML:
    def __init__(self, sesion, fecha):
        self.__nombre = "Sesion_" + str(sesion) + "_" + fecha
        self.__titulo = "Sesion " + str(sesion) + " - " + fecha
        self.__ruta = "./templates/" + self.__nombre + ".html"

    def crearFichero(self, texto, mediaBPM, cambiosPostura, tiempoSuenno):
        inicioHTML = textwrap.dedent("""\
            <!doctype html>
```

```
            <html lang=\"es\">
            <head>
                <meta charset=\"UTF-8\">
                <title>Informe - """ + self.__titulo + """</title>
                <link rel="icon" href="data:;base64,iVBORw0KGgo=">
            </head>
            <body>
                <div style=\"width:800px; margin: 0 auto;\">
                    <h1>Informe - """ + self.__titulo + """</h1>
            """)
        finalHTML = textwrap.dedent("""\
                </div>
            </body>
            </html>
            """)

        # Abrimos el archivo y escribimos el html inicial
        f = open(self.__ruta, "w+", encoding="utf8")
        f.write(inicioHTML)

        # Construimos la parte dinámica de la página HTML
        contenidoHTML = textwrap.dedent("""\
                <div>
                    <hr>
                    <h2>Resumen de tu sueño para esta sesión</h2>
                    <p>""" + texto + """</p>
                    <ul>
                        <li>Media de BPM: <strong>"""+str(int(mediaBPM))+"""<
/strong></li>
                        <li>Número de cambios de postura: <strong>"""+str(cam
biosPostura)+"""</strong></li>
                        <li>Tiempo de sueño: <strong>"""+str(tiempoSuenno)+""
" minutos</strong></li>
                    </ul>
                    <hr>
                    <h2>Gráfico de BPM a lo largo de la noche</h2>
                    <img src=\"/static/images/"""+self.__nombre+"""_bpm.png\"
 height=\"600\" width=\"800\">
                    <hr>
                    <h2>Gráfico de movimientos a lo largo de la noche</h2>
                    <img src=\"/static/images/"""+self.__nombre+"""_movimient
os.png\" height=\"600\" width=\"800\">
                    <hr>
                    <h2>Gráfico de presión sobre los sensores a lo largo de l
a noche</h2>
                    <img src=\"/static/images/"""+self.__nombre+"""_presiones
.png\" height=\"600\" width=\"800\">
                </div>
            """)
```

```
        f.write(contenidoHTML)

        # Cerramos el archivo html y cerramos el fichero
        f.write(finalHTML)
        f.close()
```

El código HTML resultante de la ejecución de este script para una sesión en concreto tiene la siguiente forma:

```
<!doctype html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Informe - Sesion 1 - 2020-01-08</title>
    <link rel="icon" href="data:;base64,iVBORw0KGgo=">
</head>
<body>
    <div style="width:800px; margin: 0 auto;">
        <h1>Informe - Sesion 1 - 2020-01-08</h1>
                <div>
                    <hr>
                    <h2>Resumen de tu sueño para esta sesión</h2>
                    <p>Has dormido demasiado poco.
No te has movido mucho al dormir.
No has entrado en sueño profundo</p>
                    <ul>
                        <li>Media de BPM: <strong>67</strong></li>
                        <li>Número de cambios de postura: <strong>3</strong><
/li>
                        <li>Tiempo de sueño: <strong>70 minutos</strong></li>
                    </ul>
                    <hr>
                    <h2>Gráfico de BPM a lo largo de la noche</h2>
                    <img src="/static/images/Sesion_1_2020-01-
08_bpm.png" height="600" width="800">
                    <hr>
                    <h2>Gráfico de movimientos a lo largo de la noche</h2>
                    <img src="/static/images/Sesion_1_2020-01-
08_movimientos.png" height="600" width="800">
                    <hr>
                    <h2>Gráfico de presión sobre los sensores a lo largo de l
a noche</h2>
                    <img src="/static/images/Sesion_1_2020-01-
08_presiones.png" height="600" width="800">
                </div>
    </div>
</body>
</html>
```

Y la página resultante para el archivo HTML anterior, es la siguiente:

Informe - Sesion 1 - 2020-01-08

Resumen de tu sueño para esta sesión

Has dormido demasiado poco. No te has movido mucho al dormir. No has entrado en sueño profundo

- Media de BPM: **67**
- Número de cambios de postura: **3**
- Tiempo de sueño: **70 minutos**

Gráfico de BPM a lo largo de la noche

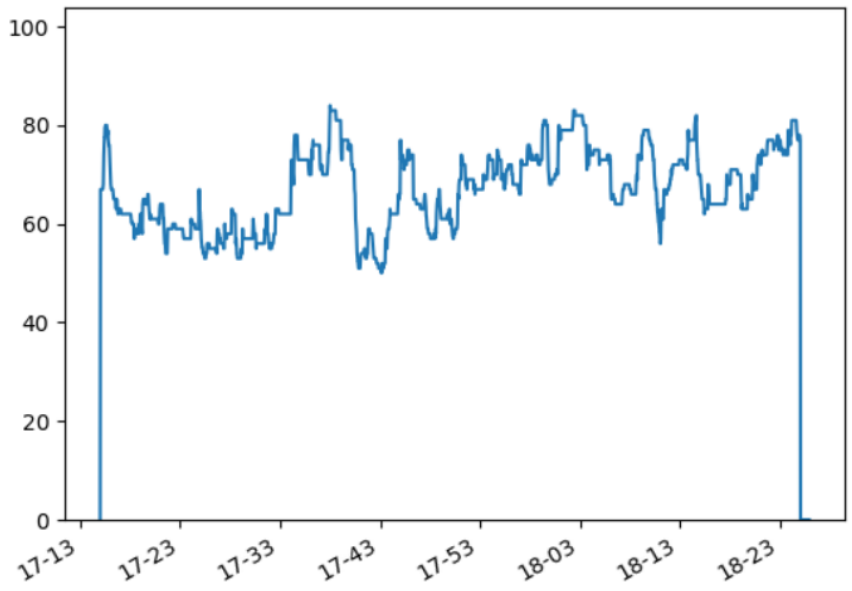


Gráfico de movimientos a lo largo de la noche

La captura de la página web está limitada por el tamaño del monitor.

FUENTES / BIBLIOGRAFÍA

- Infraestructura de sensores https://acaird.github.io/computers/2015/01/07/raspberry-pi-fsr
- Lectura de sensores con adafruit-circuitpython https://acaird.github.io/computers/2015/01/07/raspberry-pi-fsr
- Lectura de pulsaciones con la mi band 3 https://github.com/yogeshojha/MiBand3
- Datos médicos http://ses.org.es/docs/rev-neurologia2016.pdf