**BLDEA’s Dr. P.G.Halakatti College of Engineering & Technology, Vijayapur-586103.**

**PROJECT TITLE: IOT BASED HEALTH MONITORING SYSTEM**

**PROJECT LEADER: SHAMBHAVI TOLNUR(2BL15CS092)**

**Department : Computer Science & Engineering**

**Installation and Setup**

#1 Raspberry pi setup

Step 1:Insert the memory card in the Raspberry pi board.

Step 2:Load the Rasbian os into the memory card.

Step 3:Login to Raspbian os through Wifi using ssh pi@IPaddress cmd.

Step 4:Stop.

#2 DBMS installation

Step 1: install mysql.

Step 2:Draw the ER diagram of patient and health info.

Step 3:create two tables namely patient\_info and health\_info.

Step 4: normalize the table to 3NF.

Step 5: stop.

#3 Client server interaction

Step 1:Login to server IPaddress through WiFi module.

Step 2: Register the patient details into server’s patient\_info table.

Step 3: Enroll.py

import MySQLdb

import os

import time

#connect to database

try:

db=MySQLdb.connect(“192.168.43.113”, “root”, “password”, “healthCare”)

except Exception as e:

print(‘error opening database!’)

print(‘exception message;’ +str(e))

exit(1)

##gets some sensor information

while True:

print(‘enter patient information\n’)

name=raw\_input(‘enter name of the patient:’)

age=raw\_input(‘enter age of the patient:’)

gender=raw\_input(‘enter name of the patient:’)

gender=raw\_input(‘enter name of the patient:’)

address=raw\_input(‘enter address of the patient:’)

city=raw\_input(‘enter city of the patient:’)

phone=raw\_input(‘enter name of the patient:’)

#add to db

cur=db.cursor()

try:

stmt = ‘insert into patient(name,age,gender,address,city,phone,date,time) values(“%s”, %d, “%s”, “%s”,“%s”,“%s”,CURDATE(),CURTIME())’%(name,age,gender,address,city,phone)

cur.execute(stmt)

db.commit()

except:

db.rollback()

conti = raw\_input(‘do you wish to add one more entry ?(yes/no) :’)

if conti!=’yes’:

break

Step 4: Stop

.

#4 Sensors setup

Step 1:Temperature sensor(LM35) has three pins namely Vcc,ground and

Signal.

Step 2: Connect the pins of LM35 to the Raspberry pi. Connect Vcc-pin 1,

ground-pin 6 and signal-pin 7 through jumper wires.

Step 3: stop.

#5 Reading data from sensors

Step 1:Reading LM35.py

# Import Libraries

import os

import glob

import time

# Initialize the GPIO Pins

os.system('modprobe w1-gpio') # Turns on the GPIO module

os.system('modprobe w1-therm') # Turns on the Temperature module

# Finds the correct device file that holds the temperature data

base\_dir = '/sys/bus/w1/devices/'

device\_folder = glob.glob(base\_dir + '28\*')[0]

device\_file = device\_folder + '/w1\_slave'

# A function that reads the sensors data

def read\_temp\_raw():

f = open(device\_file, 'r') # Opens the temperature device file

lines = f.readlines() # Returns the text

f.close()

return lines

# Convert the value of the sensor into a temperature

def read\_temp():

lines = read\_temp\_raw() # Read the temperature 'device file'

# While the first line does not contain 'YES', wait for 0.2s

# and then read the device file again.

while lines[0].strip()[-3:] != 'YES':

time.sleep(0.2)

lines = read\_temp\_raw()

# Look for the position of the '=' in the second line of the

# device file.

equals\_pos = lines[1].find('t=')

# If the '=' is found, convert the rest of the line after the

# '=' into degrees Celsius, then degrees Fahrenheit

if equals\_pos != -1:

temp\_string = lines[1][equals\_pos+2:]

temp\_c = float(temp\_string) / 1000.0

return temp\_c

# Print out the temperature until the program is stopped.

while True:

temp\_c=read\_temp()

temp\_f = temp\_c \* 9.0 / 5.0 + 32.0

print(temp\_c, temp\_f)

time.sleep(1)

and

#!/usr/bin/python

from \_\_future\_\_ import print\_function

import cgi

import cgitb

import glob

import time

import os

cgitb.enable()

print("Content-type: text/html\n\n")

print("<h1>Temperature</h1>")

# Initialize the GPIO Pins

os.system('modprobe w1-gpio') # Turns on the GPIO module

os.system('modprobe w1-therm') # Turns on the Temperature module

# Finds the correct device file that holds the temperature data

base\_dir = '/sys/bus/w1/devices/'

device\_folder = glob.glob(base\_dir + '28\*')[0]

device\_file = device\_folder + '/w1\_slave'

# A function that reads the sensors data

def read\_temp\_raw():

f = open(device\_file, 'r') # Opens the temperature device file

lines = f.readlines() # Returns the text

f.close()

return lines

# Convert the value of the sensor into a temperature

def read\_temp():

lines = read\_temp\_raw() # Read the temperature 'device file'

# While the first line does not contain 'YES', wait for 0.2s

# and then read the device file again.

while lines[0].strip()[-3:] != 'YES':

time.sleep(0.2)

lines = read\_temp\_raw()

# Look for the position of the '=' in the second line of the

# device file.

equals\_pos = lines[1].find('t=')

# If the '=' is found, convert the rest of the line after the

# '=' into degrees Celsius, then degrees Fahrenheit

if equals\_pos != -1:

temp\_string = lines[1][equals\_pos+2:]

temp\_c = float(temp\_string) / 1000.0

return temp\_c

# Print out the temperature until the program is stopped.

f = open("temp.dat","w")

temp\_c=read\_temp()

temp\_f = temp\_c \* 9.0 / 5.0 + 32.0

#time.sleep(60)

print(temp\_f,file=f)

f.close()

cmd = '/usr/bin/sudo /usr/bin/scp /usr/lib/cgi-bin/temp.dat sana@192.168.43.113:/home/sana'

ret = os.system(cmd)

if ret == 0:

print("<h2>Temperature in Degree Fahrenheit: %f</h2><br>"%temp\_f)

#print("data transfered to server<br>")

else:

print("error transferring data..try again<br>")

Step 2: Temperature of patient is displayed in degree ℃ and ℉ on LCD display.

Step 3: The displayed value is stored in health\_info table of server.

Step 4:Reading AD8232(ECG Sensor)

#!/usr/bin/python

from \_\_future\_\_ import print\_function

import cgi

import cgitb

from MCP3008 import MCP3008

import RPi.GPIO as GPIO

import time

import os

cgitb.enable()

print("Content-type: text/html\n\n")

print("<h1>ECG SCAN</h1>")

lop = 19

lom = 26

GPIO.setmode(GPIO.BCM)

GPIO.setup(lop, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

GPIO.setup(lom, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

lop\_val = GPIO.input(lop)

lom\_val = GPIO.input(lom)

def isConnected(lopv,lomv):

if lopv == 0 and lomv == 0:

return 1 #ok

else:

return 0 #not ok

#print(lop\_val, lom\_val)

if isConnected(lop\_val, lom\_val)==1:

print("<h2>Electrodes connected <br>reading data....</h2><br>")

adc = MCP3008(0, 0)

f = open("ecg.dat","w")

n = 1500

i = 0

while i<n:

print( "%d" % adc.read(0), file=f )

time.sleep(0.005)

i+=1

print("<h2>reading done</h2><br>")

f.close()

cmd = '/usr/bin/sudo /usr/bin/scp /usr/lib/cgi-bin/ecg.dat sana@192.168.43.113:/home/sana'

ret = os.system(cmd)

if ret == 0:

print("file transfered<br>")

else:

print("error in transferring file..try again<br>")

else:

print("<h2>Electrodes not connected</h2>")

Step 5: Reading MC3008(Converter)

from spidev import SpiDev

import time

class MCP3008:

def \_\_init\_\_(self, bus = 0, device = 0):

self.bus, self.device = bus, device

self.spi = SpiDev()

self.open()

def open(self):

self.spi.open(self.bus, self.device)

def read(self, channel = 0):

self.spi.max\_speed\_hz = 1350000

adc = self.spi.xfer2([1, (8 + channel) << 4, 0])

#print( "raw data", adc)

data = ((adc[1] & 3) << 8) + adc[2]

return data

def close(self):

self.spi.close()

and

from spidev import SpiDev

import time

class mcpp:

def \_\_init\_\_(self, bus = 0, device = 0):

self.bus, self.device = bus, device

self.spi = SpiDev()

self.open()

def open(self):

self.spi.open(self.bus, self.device)

def read(self, channel = 1):

self.spi.max\_speed\_hz = 1350000

adc = self.spi.xfer2([1, (8 + channel) << 4, 0])

#print( "raw data", adc)

data = ((adc[1] & 3) << 8) + adc[2]

return data

def close(self):

self.spi.close()

and

from \_\_future\_\_ import print\_function

from MCP3008 import MCP3008

import RPi.GPIO as GPIO

import time

lop = 19

lom = 26

f = open("b.b","w")

GPIO.setmode(GPIO.BCM)

GPIO.setup(lop, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

GPIO.setup(lom, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

adc = MCP3008(0, 0)

n = 20

i = 0

print( GPIO.input(lop), GPIO.input(lom) )

print("A B")

while i<n:

print("%d %d" % (i,adc.read(0)), file=f)

time.sleep(0.01)

i+=1

Step 5: Reading TCRT1000(Pulse sensor)

#!/usr/bin/python

from \_\_future\_\_ import print\_function

import cgi

import cgitb

from mcpp import mcpp

import RPi.GPIO as GPIO

import time

import os

cgitb.enable()

print("Content-type: text/html\n\n")

print("<h1>PULSE RATE MEASUREMENT</h1>")

"""lop = 19

lom = 26

GPIO.setmode(GPIO.BCM)

GPIO.setup(lop, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

GPIO.setup(lom, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

lop\_val = GPIO.input(lop)

lom\_val = GPIO.input(lom)"""

adc = mcpp(0, 0)

f = open("pulse.dat","w")

n = 10

i = 0

pulse=0

wrong=0

while i<n:

#print( "%d" % adc.read(1) )

cur\_pulse = adc.read(1)

if cur\_pulse < 600 and cur\_pulse > 400:

pulse = pulse+cur\_pulse

i+=1

#print(cur\_pulse)

else:

#print('Connect Pulse Rate Sensor')

f.close()

time.sleep(1)

f = open("pulse.dat","w")

print("<h2>Place Finger on pulse sensor <br>Try Again....</h2><br>")

wrong=1

break;

time.sleep(0.5)

if wrong == 0:

pulse = pulse/(70)

print(pulse, file=f)

print("<h2>Pulse Rate = %d</h2>" %pulse);

#print("<h2>reading done</h2><br>")

f.close()

cmd = '/usr/bin/sudo /usr/bin/scp /usr/lib/cgi-bin/pulse.dat sana@192.168.43.113:/home/sana'

ret = os.system(cmd)

if ret == 0:

#print("file transfered<br>")

print(".")

else:

print("error transferring Data to the server..try again<br>")

and

import time

import threading

from MCP3008 import MCP3008

class Pulsesensor:

def \_\_init\_\_(self, channel = 0, bus = 0, device = 0):

self.channel = channel

self.BPM = 0

self.adc = MCP3008(bus, device)

def getBPMLoop(self):

# init variables

rate = [0] \* 10 # array to hold last 10 IBI values

sampleCounter = 0 # used to determine pulse timing

lastBeatTime = 0 # used to find IBI

P = 512 # used to find peak in pulse wave, seeded

T = 512 # used to find trough in pulse wave, seeded

thresh = 525 # used to find instant moment of heart beat, seeded

amp = 100 # used to hold amplitude of pulse waveform, seeded

firstBeat = True # used to seed rate array so we startup with reasonable BPM

secondBeat = False # used to seed rate array so we startup with reasonable BPM

IBI = 600 # int that holds the time interval between beats! Must be seeded!

Pulse = False # "True" when User's live heartbeat is detected. "False" when not a "live beat".

lastTime = int(time.time()\*1000)

while not self.thread.stopped:

Signal = self.adc.read(self.channel)

currentTime = int(time.time()\*1000)

sampleCounter += currentTime - lastTime

lastTime = currentTime

N = sampleCounter - lastBeatTime

# find the peak and trough of the pulse wave

if Signal < thresh and N > (IBI/5.0)\*3: # avoid dichrotic noise by waiting 3/5 of last IBI

if Signal < T: # T is the trough

T = Signal # keep track of lowest point in pulse wave

if Signal > thresh and Signal > P:

P = Signal

# signal surges up in value every time there is a pulse

if N > 250: # avoid high frequency noise

if Signal > thresh and Pulse == False and N > (IBI/5.0)\*3:

Pulse = True # set the Pulse flag when we think there is a pulse

IBI = sampleCounter - lastBeatTime # measure time between beats in mS

lastBeatTime = sampleCounter # keep track of time for next pulse

if secondBeat: # if this is the second beat, if secondBeat == TRUE

secondBeat = False; # clear secondBeat flag

for i in range(len(rate)): # seed the running total to get a realisitic BPM at startup

rate[i] = IBI

if firstBeat: # if it's the first time we found a beat, if firstBeat == TRUE

firstBeat = False; # clear firstBeat flag

secondBeat = True; # set the second beat flag

continue

# keep a running total of the last 10 IBI values

rate[:-1] = rate[1:] # shift data in the rate array

rate[-1] = IBI # add the latest IBI to the rate array

runningTotal = sum(rate) # add upp oldest IBI values

runningTotal /= len(rate) # average the IBI values

self.BPM = 60000/runningTotal # how many beats can fit into a minute? that's BPM!

if Signal < thresh and Pulse == True: # when the values are going down, the beat is over

Pulse = False # reset the Pulse flag so we can do it again

amp = P - T # get amplitude of the pulse wave

thresh = amp/2 + T # set thresh at 50% of the amplitude

P = thresh # reset these for next time

T = thresh

if N > 2500: # if 2.5 seconds go by without a beat

thresh = 512 # set thresh default

P = 512 # set P default

T = 512 # set T default

lastBeatTime = sampleCounter # bring the lastBeatTime up to date

firstBeat = True # set these to avoid noise

secondBeat = False # when we get the heartbeat back

self.BPM = 0

time.sleep(0.005)

# Start getBPMLoop routine which saves the BPM in its variable

def startAsyncBPM(self):

self.thread = threading.Thread(target=self.getBPMLoop)

self.thread.stopped = False

self.thread.start()

return

# Stop the routine

def stopAsyncBPM(self):

self.thread.stopped = True

self.BPM = 0

return

#6 Testing

Step 1: All the sensors are connected to the patient.

Step 2:Different sensor parameters( Temperature, heart rate,ECG and blood Pressure) are

recoreded and stored into the health\_info table.

Step 3:The database contents are analysed by the doctor for patient disease diagnosis.