**Documentation of Graphical User Interface**

**Notes:-**

**For making the front end of UI, we have used Tkinter framework along with many libraries of python. For the backend logic, we have used mysql.connector library especially**

**server side code runs in python**

**raspberry pi code runs in python3**

**In both server and raspberry pi paho mqtt library has been installed for a viable MQTT connection between client and server.**

**2 codes always run in server: - oximter.py and signup.py inside Documents/Neelam/mqtt**

**1 code always run in raspberry pi:- recvflag.py**

**Required Libraries: - paho.mqtt.client, os, hashlib, tkinter, matplotlib, PIL, random, string, time, sys, mysql.connector**

**RASPBERRY PI FILES:**

**Index.py:**

This python file is the starting user interface of Raspberry pi where two tabs are present of which one tab is for signup and another is signin.

Signup page has different fields to be filled either by a patient or by a paramedic. During this registration fingerprint registration is mandatory. The fingerprint of a particular user is saved in a text file locally named signup\_logs.txt against the hashkey of that person. Hashkey is generated by sha1 function of hashlib library in python. After filling all the fields, the user is asked to register his/her fingerprint which is saved in the text file (id-num, hashkey). The details of this functionality is inside fpsignup function. On clicking the submit button, a function savessignup is called whose main function is to extract different parameters from the field and then encode them all into a json format. After encoding the data are sent to the main server over MQTT publish method with a topic “signup”. During registration, validation of each and every field has also added.

Signin Page provides two different methods to login a user. One by validation of email id & password and another method is by fingerprint login. In the first method, after clicking the submit button, savessignin function is called which extract data from the fields, encodes the data into json format and then transfer those data to server over MQTT with a topic “signin” for checking of email and password.

In local server whose IP is 10.14.79.58, there is a file called signup.py inside Documents/Neelam/mqtt which should always run. The broker should remain on every time. The local broker HiveMQ is installed in server which is inside /opt folder. To run HiveMQ, go inside HiveMQ folder through terminal and then run the command “./bin/run.sh”. The function of signup.py file is to accept data by subscribing to 2 particular topics “signup” and “signin”. Whenever a user want to register or login from the Device, the code which is running on the device publishes those data to server with topic “signup” or “signin” depending on the action performed by the user.

If a user wishes to login via fingerprint, then after clicking the fingerprint option, we search for a validate fingerprint in our local file “signup\_logs.txt” and extract its hashkey saving it into “temphash.text” which is used later to send along with the sensor data.

After a successful registration or login, server publishes “1” with topic “signupresult” or “signinresult” to the HiveMQ broker which is in local server. Recvflag.py in raspberry pi always runs in the background whose function is to accept signupresult and signinresult and accordingly take action.

**Recvflag.py:**

The main functionality of this code is to accept result of a registration and login after a verification from the database. If the user is registering for the first time using the kit, then after a successful registration, signup.py file located in server sends a result code 1 with topic “signupresult” along with the hashkey of that particular user, then after\_login.py file is opened via os.system method. Similarly, after a successful login, signup.py file sends result code 1 with topic “signinresult” along with hashkey of the user, then the UI is directed to after\_login.py. Another functionality of this file is that it asks for the hashkey of the user from the server and saves it into a file such that whenever sensor data is being sent to the local server, this hashkey is appended with those data for identification of sensor data of individual patient. At one time, only one user can use the kit, keeping that in mind the hashkey is being override in the “temphash.text” file again and again after a login or registration.

**After\_login.py:**

The main functionality of this python code is to run two files simultaneously those are servertosend.py and icon.py. After login or registration, the UI of icon.py is displayed along with that sensor data are sent to the server in background.

**Icon.py:**

This python file contains 5 buttons which is displayed in UI in image format. Those 5 buttons are pulse, sop2, temperature, airflow and ECG. On clicking pulse button servertosend.py file is being killed by a shell file, current UI is destroyed and then barplot\_pbm.py file is opened. The servertosend.py file is killed because in barplot\_pbm.py file also we are trying to access the same port of oximeter which may later cause a multiple access port error. Similarly, on clicking spo2 button servertosend.py file is being killed by a shell file, current UI is destroyed and then barplot\_spo2.py file is opened. On clicking temp button servertosend.py file is being killed by a shell file, current UI is destroyed and then barplot\_temp.py file is opened. On clicking airflow button servertosend.py file is being killed by a shell file, current UI is destroyed and then barplot\_airflow.py file is opened.

A “LOGOUT” button is present below. On clicking this button, servertosend file is destroyed as there is no more requirement of transfer of data to server and then the user is redirected to index.py .

**Kill\_serverfile.sh:**

The function of the shell file is to search for the pid of servertosend.py and kill this python file.

**Servertosend.py:**

The main functionality of this python file is to serially read oximter data from eHealth sensor board and send them to the server along with the hashkey which is saved in temphash.text

**Temphash.text:**

This text file saves the hashkey of the current user who has been signed in. This file is overrided every time when an individual user access the device.

**Signup\_logs.txt:**

This file saves the fingerprint ID of an individual user along with his/her hashkey such that during the method of fingerprint lognin, the user is being logged in using his/her hashkey.

**Barplot\_pbm.py:**

The main work of this python file is to provide live plotting of pulse data which is being read from the pulse oximeter along with parallel sending/publishing of data to the server over MQTT with a topic “oximeter”. Another back button is present in this UI which on click directed to after\_login.py. On server side, oximeter.py file is running which accepts the data after subscribing to “oximeter” and then insert those data to the database along with the hashkey of the patient.

**Barplot\_spo2.py:**

The main work of this python file is to provide live plotting of spo2 data which is being read from the pulse oximeter along with parallel sending/publishing of data to the server over MQTT with a topic “oximeter”. Another back button is present in this UI which on click directed to after\_login.py. On server side, oximeter.py file is running which accepts the data after subscribing to “oximeter” and then insert those data to the database along with the hashkey of the patient.

**Barplot\_temp.py:**

The main work of this python file is to provide live plotting of temperature data which is being read from the pulse oximeter along with parallel sending/publishing of data to the server over MQTT with a topic “oximeter”. Another back button is present in this UI which on click directed to after\_login.py. On server side, oximeter.py file is running which accepts the data after subscribing to “oximeter” and then insert those data to the database along with the hashkey of the patient.

**Barplot\_airflow.py:**

The main work of this python file is to provide live plotting of airflow data which is being read from the pulse oximeter along with parallel sending/publishing of data to the server over MQTT with a topic “oximeter”. Another back button is present in this UI which on click directed to after\_login.py. On server side, oximeter.py file is running which accepts the data after subscribing to “oximeter” and then insert those data to the database along with the hashkey of the patient.

**SERVER FILES:**

1. **Oximeter.py:**

The function of this file is to connect to the broker, subscribe to the broker with topic “oximter”. Whenever any device provides/publishes it pbm,spo2, temperature and airflow data along with hashkey, it inserts those data to the EpiOne\_Data table using mysql.connector.

1. **Signup.py:**

It’s main functionality is to connect to the broker, subscribe to the broker with topic “signup ” and “signin”.

When user is registering using the device, then the signup data is being received by this file. Hashkey is generated from the name, email and contact of the user. Then these data are inserted into the Patient Table. During the insertion, keep in mind that all the columns are mentioned in the insert query and all the values are provided otherwise it may cause error. After a successful insertion to database, it sends/publishes “signupresult” as 1 to the broker Over MQTT along with the “hkey” as the hashkey of the particular user to the raspberry pi . Then, after\_login.py(in raspberry pi) is opened.

When user is logging in from the Display Raspberry pi using email id and password, then the email and password is verified from the content of the database. If email and password is correct, then the user has to be logged in successfully. For this reason, it sends/publishes “signinresult” as 1 to the broker Over MQTT along with the “hkey”. Hashkey is extracted from the database and send to raspberry pi. Then, after\_login.py (in raspberry pi) is opened.