Visvesvaraya National Institute of Technology (VNIT), Nagpur

Embedded Systems (ECL301)

Project Report

Submitted by :
P.Gowtham Kumar (BT21ECE077)
G.Guru Swami (BT21ECE078)
P.Adish (BT21ECE079)
Semester 4

Submitted to :
Dr. Ankit A. Bhurane
(Course Instructor)
Department of Electronics and Communication Engineering,
VNIT Nagpur

GUI CODE: 1

:

```
import g4p_controls.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;
import processing.net.*;
import controlP5.*;
import processing.serial.*;
import java.util.Arrays;
import java.io.DataOutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
boolean savingThreadActive = false;

String botToken = "6887716997:AAGXlPq0B3298Y54deznedwEkryPgVS0qbk";
  // Assuming this is the username


  // Construct the Telegram Bot API URL
  String apiUrl = "https://api.telegram.org/bot" + botToken + ...
     "/sendMessage";

  // Construct the POST data

String receivedMessage = "";
processing.net.Client client;
Serial port;
Chart myChart1;
Chart myChart2;
Chart myChart3;
ControlP5 cp5;
PFont font;
int nl = 10;
String received = null;
String[] readings;
int stop_flag = 0;
int start_flag = 0;
PrintWriter save_data;
int requestInterval = 300; // Set the interval for requesting ...
    values (in milliseconds)
int lastRequestTime = 0;
String esp32IP = "192.168.137.185";
int esp32Port = 80;

```

```
44 GTextField usernameField, passwordField, emailField, ...
       signInUsernameField, signInPasswordField;
45 GButton signupButton, AlreadyRegisteredButton, newuserButton, ...
       signInButton;
46 GLabel messageLabel, signInMessageLabel;
47 GLabel signupLabel, signInLabel;
48 String name="";
49 GLabel ...
       usernameLabel,passwordLabel,emailLabel,signInUsernameLabel,signInPasswordLabel;
50 int id;
51
52 Connection conn = null;
53 boolean signedUp = false;
54 boolean signin = false;
55
56 void setup() {
57   size(800, 450);
58   createGUI();
59   connectToDatabase();
60 }
61
62 void draw() {
63   background(220);
64   if(signin){
65       background(0, 0, 0);
66   fill(0, 255, 0);
67   textFont(font);
68   text("PULSE WAVEFORM", 60, 80);
69
70   // Check if it's time to send a new request
71   int currentTime = millis();
72
73   if (currentTime - lastRequestTime ≥ requestInterval) {
74     // Send a request
75     sendRequest();
76     // Update the last request time
77     lastRequestTime = currentTime;
78   }
79
80   // Handle the response if available
81   if (client != null && client.active() && client.available() > ...
         0) {
82     receivedMessage = client.readString();
83     received = receivedMessage;
84     if (received != null && start_flag == 1 && stop_flag == 0) {
85       readings = received.split(",", 4);
86       readings[1] = readings[1];
87       readings[2] = readings[2];
88       readings[3] = readings[3];
```

```
89
90        println(readings[1], readings[2], readings[3]);
91
92        String dataToAppend = hour() + "." + nf(minute(), 2) + "." ...
              + nf(second(), 2) + "." + nf(millis(), 3) + " : " +
93          readings[1] + "," + readings[2] + "," + readings[3];
94
95        String[] existingData = loadStrings("sensor_data.txt");
96        String[] newData = concat(existingData, new String[] { ...
              dataToAppend });
97        saveStrings("sensor_data.txt", newData);
98
99        save_data.println(hour() + "." + "0" + minute() + "." + ...
              second() + "." + millis() + " : " +
100         readings[1] + "," + readings[2] + "," + readings[3]);
101
102       myChart1.push("incoming", float(readings[1]));
103       myChart2.push("incoming", float(readings[2]));
104       myChart3.push("incoming", float(readings[3]));
105       if (name!=""){
106         int numid=getUserIdFromDatabase(name);
107         insertSensorData(numid,readings[1],readings[2],readings[3]) ...
              ;
108
109       }
110
111     }
112     // Uncomment the next line if you want to close the ...
          connection after receiving a response
113     client.stop();
114   }
115   }
116
117 }
118 void insertSensorData(int id, String value1, String value2, ...
      String value3) {
119   try {
120     // Prepare the INSERT SQL statement
121     String insertQuery = "INSERT INTO PULSE (id, VATA, PIIA, ...
          KAPHA, DA_TE) VALUES (?, ?, ?, ?, NOW())";
122     PreparedStatement pstmt = conn.prepareStatement(insertQuery);
123     pstmt.setInt(1, id);
124     pstmt.setFloat(2, float(value1));
125     pstmt.setFloat(3, float(value2));
126     pstmt.setFloat(4, float(value3));
127
128     // Execute the INSERT query
129     pstmt.executeUpdate();
130
```

```
131     // Close resources
132     pstmt.close();
133   } catch (SQLException e) {
134     e.printStackTrace();
135     println("Error inserting sensor data into the database");
136   }
137 }
138 
139 void pulsesetup(){
140   printArray(Serial.list());
141   cp5 = new ControlP5(this);
142   font = createFont("calibri light bold", 20);
143 
144   cp5.addButton("Start")
145     .setPosition(80, 120)
146     .setSize(120, 70)
147     .setFont(font);
148 
149   cp5.addButton("Stop")
150     .setPosition(80, 220)
151     .setSize(120, 70)
152     .setFont(font);
153 
154   cp5.addButton("Save")
155     .setPosition(80, 320)
156     .setSize(120, 70)
157     .setFont(font);
158 
159   myChart1 = cp5.addChart("pulse waveform 1")
160     .setPosition(260, 10)
161     .setSize(500, 125)
162     .setView(Chart.LINE)
163     .setRange(0, 150);
164 
165 
166   myChart1.addDataSet("incoming");
167   myChart1.setData("incoming", new float[100]);
168 
169   myChart2 = cp5.addChart("pulse waveform 2")
170     .setPosition(260, 155)
171     .setSize(500, 125)
172     .setView(Chart.LINE)
173     .setRange(0, 150);
174 
175   myChart2.addDataSet("incoming");
176   myChart2.setData("incoming", new float[100]);
177 
178   myChart3 = cp5.addChart("pulse waveform 3")
179     .setPosition(260, 300)
```

```
180      .setSize(500, 125)
181      .setView(Chart.LINE)
182      .setRange(0, 150);
183
184    myChart3.addDataSet("incoming");
185    myChart3.setData("incoming", new float[100]);
186
187    save_data = createWriter("sensordata.txt" + day() + "-" + ...
           month() + "-" + year() + ":" + hour() + "." + minute());
188  }
189  void createGUI() {
190    signupLabel = new GLabel(this, 200, 10, 100, 20);
191    signupLabel.setText("Sign Up");
192
193    usernameLabel = new GLabel(this, 50, 30, 100, 30);
194    usernameLabel.setText("Username:");
195
196    passwordLabel = new GLabel(this, 50, 80, 100, 30);
197    passwordLabel.setText("Password:");
198
199    emailLabel = new GLabel(this, 50, 130, 100, 30);
200    emailLabel.setText("TelegramChatID");
201
202    usernameField = new GTextField(this, 150, 30, 200, 30);
203    passwordField = new GTextField(this, 150, 80, 200, 30);
204    emailField = new GTextField(this, 150, 130, 200, 30);
205
206    signupButton = new GButton(this, 50, 180, 100, 30);
207    signupButton.setText("Sign Up");
208
209    AlreadyRegisteredButton = new GButton(this, 200, 180, 100, 30);
210    AlreadyRegisteredButton.setText(" Registered?");
211
212    messageLabel = new GLabel(this, 100, 220, 200, 30);
213    messageLabel.setTextAlign(GAlign.LEFT, GAlign.MIDDLE);
214  }
215
216  void connectToDatabase() {
217    String dbURL = "jdbc:mysql://localhost:3306/SIGNUP";
218    String dbUser = "root";
219    String dbPassword = "Guru@2002";
220
221    try {
222      conn = DriverManager.getConnection(dbURL, dbUser, dbPassword);
223    } catch (SQLException e) {
224      e.printStackTrace();
225      println("Database connection error");
226    }
227  }
```

```
228
229 void handleButtonEvents(GButton button, GEvent event) {
230   if (button == signupButton && event == GEvent.CLICKED) {
231     String username = usernameField.getText();
232     String password = passwordField.getText();
233     String email = emailField.getText();
234
235     if (signupSuccessful(username, password, email)) {
236       if (insertUser(username, password, email)) {
237         String welcomeMessage = "Welcome, " + username + "!" + " ...
                Kindly press registered and signin using user id " ;
238         messageLabel.setText(welcomeMessage);
239       } else {
240         messageLabel.setText("Signup failed. Please try again.");
241       }
242     } else {
243       messageLabel.setText("Invalid input. Please check your ...
            information.");
244     }
245   }
246
247   if (button == AlreadyRegisteredButton && event == ...
        GEvent.CLICKED) {
248     // Clear the existing GUI
249     clearGUI();
250
251     // Create the sign-in GUI
252     createSignInGUI();
253   }
254
255   if (button == newuserButton && event == GEvent.CLICKED) {
256     // Clear the existing GUI
257     clearSignInGUI();
258
259     // Create the sign-up GUI
260     createGUI();
261   }
262   if (button == signInButton && event == GEvent.CLICKED) {
263     String enteredUsername = signInUsernameField.getText();
264     String enteredPassword = signInPasswordField.getText();
265
266     if (checkUserCredentials(enteredUsername, enteredPassword)) {
267       clearSignInGUI();
268       pulsesetup();
269       signin = true;
270       id =  getUserIdFromDatabase(enteredUsername);
271     } else {
272       signInMessageLabel.setText("Invalid username or password. ...
            Please try again.");
```

```
273       }
274     }
275   }
276   int getUserIdFromDatabase(String username) {
277     int userId = -1; // Default value if no user is found
278
279     try {
280       // Prepare the SELECT SQL statement
281       String selectQuery = "SELECT id FROM SIGN WHERE username = ?";
282       PreparedStatement pstmt = conn.prepareStatement(selectQuery);
283       pstmt.setString(1, username);
284
285       // Execute the SELECT query
286       ResultSet resultSet = pstmt.executeQuery();
287
288       // Check if any row is returned
289       if (resultSet.next()) {
290         // Get the id from the result set
291         userId = resultSet.getInt("id");
292       }
293
294       // Close resources
295       resultSet.close();
296       pstmt.close();
297     } catch (SQLException e) {
298       e.printStackTrace();
299       println("Error retrieving user id from the database");
300     }
301
302     return userId;
303   }
304
305   void clearGUI() {
306     usernameField.setVisible(false);
307     emailLabel.setVisible(false);
308     usernameLabel.setVisible(false);
309     passwordLabel.setVisible(false);
310
311     passwordField.setVisible(false);
312     emailField.setVisible(false);
313     signupButton.setVisible(false);
314     AlreadyRegisteredButton.setVisible(false);
315     messageLabel.setVisible(false);
316     signupLabel.setVisible(false);
317   }
318
319   void createSignInGUI() {
320     signInLabel = new GLabel(this, 200, 10, 100, 20);
321     signInLabel.setText("Sign In");
```

```
322
323   signInUsernameLabel = new GLabel(this, 50, 30, 100, 30);
324   signInUsernameLabel.setText("Username:");
325
326   signInPasswordLabel = new GLabel(this, 50, 80, 100, 30);
327   signInPasswordLabel.setText("Password:");
328
329   signInUsernameField = new GTextField(this, 150, 30, 200, 30);
330   signInPasswordField = new GTextField(this, 150, 80, 200, 30);
331
332   signInButton = new GButton(this, 50, 130, 100, 30);
333   signInButton.setText("Sign In");
334
335   newuserButton = new GButton(this, 180, 130, 100, 30);
336   newuserButton.setText("New User?");
337
338   signInMessageLabel = new GLabel(this, 100, 220, 200, 30);
339   signInMessageLabel.setTextAlign(GAlign.LEFT, GAlign.MIDDLE);
340 }
341
342 void clearSignInGUI() {
343   signInUsernameField.setVisible(false);
344   signInPasswordField.setVisible(false);
345   signInButton.setVisible(false);
346   signInUsernameLabel.setVisible(false);
347   signInPasswordLabel.setVisible(false);
348   newuserButton.setVisible(false);
349   signInMessageLabel.setVisible(false);
350   signInLabel.setVisible(false);
351 }
352
353 boolean signupSuccessful(String username, String password, ...
      String email) {
354   return !username.isEmpty() && !password.isEmpty() && ...
        !email.isEmpty();
355 }
356
357 boolean insertUser(String username, String password, String ...
      email) {
358   try {
359     String insertQuery = "INSERT INTO SIGN (username, password1, ...
          email) VALUES (?, ?, ?)";
360     PreparedStatement pstmt = conn.prepareStatement(insertQuery);
361     pstmt.setString(1, username);
362     pstmt.setString(2, password);
363     pstmt.setString(3, email);
364     pstmt.executeUpdate();
365     pstmt.close();
366     return true;
```

```
367     } catch (SQLException e) {
368       e.printStackTrace();
369       println("user account is already there try to sign in ");
370       return false;
371     }
372   }
373   boolean checkUserCredentials(String enteredUsername, String ...
        enteredPassword) {
374     try {
375       // Query the database to check if the entered credentials exist
376       name= enteredUsername;
377       String query = "SELECT * FROM SIGN WHERE username = ? AND ...
            password1 = ?";
378       PreparedStatement pstmt = conn.prepareStatement(query);
379       pstmt.setString(1, enteredUsername);
380       pstmt.setString(2, enteredPassword);
381
382       // Execute the query
383       ResultSet resultSet = pstmt.executeQuery();
384
385       // Check if any row is returned, indicating a matching user
386       if (resultSet.next()) {
387         return true; // User with the provided credentials exists
388       } else {
389         return false; // No user found with the provided credentials
390       }
391     } catch (SQLException e) {
392       e.printStackTrace();
393       return false; // Error occurred during database query
394     }
395   }
396
397   void Start() {
398     start_flag = 1;
399     stop_flag = 0;
400   }
401
402   void Stop() {
403     stop_flag = 1;
404     start_flag = 0;
405
406   }
407   void Save() {
408     if (!savingThreadActive) {
409       // Start a new thread for saving data
410       thread("saveDataThread");
411     }
412   }
413
```

```
414 String getEmailForCurrentUser() {
415   // Retrieve the email for the current user from the database
416   String userEmail = "";
417 
418   try {
419     // Prepare the SELECT SQL statement to get the email
420     String emailQuery = "SELECT email FROM SIGN WHERE id = ?";
421     PreparedStatement pstmt = conn.prepareStatement(emailQuery);
422     pstmt.setInt(1, id);
423 
424     // Execute the SELECT query
425     ResultSet resultSet = pstmt.executeQuery();
426 
427     // Check if any row is returned
428     if (resultSet.next()) {
429       // Get the email from the result set
430       userEmail = resultSet.getString("email");
431     }
432 
433     // Close resources
434     resultSet.close();
435     pstmt.close();
436   } catch (SQLException e) {
437     e.printStackTrace();
438     println("Error retrieving email for the current user");
439   }
440 
441   return userEmail;
442 }
443 void saveDataThread() {
444   // Set the flag to indicate that the saving thread is active
445   savingThreadActive = true;
446 
447   // Perform the save data operations
448   save_data.flush();
449   save_data.close();
450   calculateAveragesForCurrentUser();
451   String message = calculateAveragesForCurrentUser();
452   String targetUserId =  getEmailForCurrentUser();
453   String postData = "chat_id=" + targetUserId + "&text=" + message;
454 
455   sendPostRequest(apiUrl, postData);
456 
457   // Create a new save_data writer
458   save_data = createWriter("sensordata.txt" + day() + "/" + ...
        month() + "/" + year() + ":" + hour() + "." + minute());
459 
460   // Reset the flag to indicate that the saving thread is no ...
        longer active
```

```
461   savingThreadActive = false;
462 }
463
464 void sendRequest() {
465   // Open a new client connection
466   client = new processing.net.Client(this, esp32IP, esp32Port);
467   if (client.active()) {
468     // Send a request
469     String request = "GET / HTTP/1.1\r\n" +
470                      "Host: " + esp32IP + "\r\n\r\n";
471     client.write(request);
472   }
473 }
474
475 String calculateAveragesForCurrentUser() {
476   // Calculate averages for the current user's ID and return a ...
        formatted string
477   StringBuilder result = new StringBuilder();
478
479   try {
480     // Prepare the SELECT SQL statement to get the averages
481     String avgQuery = "SELECT AVG(VATA) AS avgVATA, AVG(PIIA) AS ...
          avgPIIA, AVG(KAPHA) AS avgKAPHA FROM PULSE WHERE id = ?";
482     PreparedStatement pstmt = conn.prepareStatement(avgQuery);
483     pstmt.setInt(1, id);
484
485     // Execute the SELECT query
486     ResultSet resultSet = pstmt.executeQuery();
487
488     // Check if any row is returned
489     if (resultSet.next()) {
490       // Get the average values from the result set
491       float avgVATA = resultSet.getFloat("avgVATA");
492       float avgPIIA = resultSet.getFloat("avgPIIA");
493       float avgKAPHA = resultSet.getFloat("avgKAPHA");
494
495       // Append the average values to the result string
496       result.append("Average VATA: ").append(avgVATA).append("\n");
497       result.append("Average PIIA: ").append(avgPIIA).append("\n");
498       result.append("Average KAPHA: ...
            ").append(avgKAPHA).append("\n");
499     }
500
501     // Close resources
502     resultSet.close();
503     pstmt.close();
504   } catch (SQLException e) {
505     e.printStackTrace();
506     println("Error calculating averages for the current user");
```

```
507   }
508
509   return result.toString();
510 }void stop() {
511   // Close the client connection when the sketch is stopped
512   if (client != null && client.active()) {
513     client.stop();
514   }
515 }
516
517 void sendPostRequest(String url, String data) {
518   try {
519     // Create a URL object
520     URL apiUrl = new URL(url);
521
522     // Open a connection
523     HttpURLConnection connection = (HttpURLConnection) ...
          apiUrl.openConnection();
524
525     // Set the request method to POST
526     connection.setRequestMethod("POST");
527
528     // Enable input/output streams
529     connection.setDoOutput(true);
530
531     // Set the content type
532     connection.setRequestProperty("Content-Type", ...
          "application/x-www-form-urlencoded");
533
534     // Get the output stream
535     try (DataOutputStream outputStream = new ...
          DataOutputStream(connection.getOutputStream())) {
536       // Write the data
537       outputStream.writeBytes(data);
538       outputStream.flush();
539     }
540
541     // Get the response code (optional)
542     int responseCode = connection.getResponseCode();
543     println("Response Code: " + responseCode);
544
545     // Close the connection
546     connection.disconnect();
547   }
548   catch (Exception e) {
549     println("An error occurred: " + e.getMessage());
550     }
551 }
```

GUI CODE EXPLANATION:

void setup():

Sets up the initial size of the display window to 800x450 pixels. Calls the createGUI() function to set up the graphical user interface. Calls the connectToDatabase() function to establish a connection to the database. void draw():

Draws the graphical elements on the screen. If the signin flag is true, sets the background to black and displays "PULSE WAVEFORM" text at coordinates (60, 80). Checks if it's time to send a new request to an external device (ESP32) by comparing the current time with the last request time. If it's time, calls the sendRequest() function to send a request to the external device. Handles the response if the client is active and data is available. Parses the received data, updates charts, and saves sensor data to a file and the database.

void insertSensorData(int id, String value1, String value2, String value3):

Inserts sensor data (pulse readings) into the database. Constructs an SQL statement to insert data into the "PULSE" table with user ID (id), VATA, PIIA, KAPHA values, and the current date and time (NOW()). Prepares and executes the SQL statement, updating the database with the sensor data.

void pulsesetup():

printArray(Serial.list());: Lists the available serial ports. cp5 = new ControlP5(this);: Initializes a ControlP5 object to create graphical user interface controls. font = createFont("calibri light bold", 20);: Creates a font for the GUI elements. Button Creation: Creates three buttons (Start, Stop, Save) using ControlP5. Positions and sets the size of each button on the screen. Sets the font for the buttons. Chart Creation: Initializes three line charts (myChart1, myChart2, myChart3) using ControlP5. Positions and sets the size of each chart on the screen. Configures the view and range of each chart. Adds an "incoming" data set to each chart and initializes it with an array of floats. File Writer Initialization: Creates a file writer $(\text{save}_data) with a filename based on the current date and time to save sensor data.$

void createGUI():

Creates the graphical user interface elements for the signup form: Labels: Creates labels for "Sign Up," "Username," "Password," and "TelegramChatID." Text Fields: Generates text input fields for the username, password, and Telegram chat ID. Buttons: Creates "Sign Up" and "Registered?" buttons for user interaction. Message Label: Sets up a label for displaying messages or instructions.

void connectToDatabase():

Purpose: Establishes a connection to a MySQL database. Steps: Defines the database URL, username, and password. Attempts to connect to the database using DriverManager.getConnection(). If successful, conn (presumably a database connection object) is assigned the database connection; otherwise, an error is printed.

void handleButtonEvents(GButton button, GEvent event):

Purpose: Handles button events triggered in the graphical user interface (GUI). Steps: signupButton: If the "Sign Up" button is clicked: Retrieves text input for username, password, and email fields. Checks if the signup inputs are valid using signupSuccessful(). Inserts the user if signup is successful using insertUser(). Displays appropriate messages based on success or failure. AlreadyRegisteredButton: If the "Registered?" button is clicked: Clears the existing GUI and switches to the sign-in GUI using clearGUI() and createSignInGUI(). newuserButton: If the "New User?" button is clicked: Clears the existing sign-in GUI and switches back to the sign-up GUI using clearSignInGUI() and createGUI(). signInButton: If the "Sign In" button is clicked: Retrieves entered username and password. Validates credentials using checkUserCredentials(). Switches to the main functionality (pulses

int getUserIdFromDatabase(String username):

Purpose: Retrieves the user ID from the database based on the provided username. Steps: Initializes userId to -1 as a default value if no user is found. Attempts to execute a SQL query to select the id from the SIGN table where the username matches the input parameter. Executes the query using conn (presumably a database connection) and retrieves the result set. If a row is returned (indicating a match), extracts the id value and assigns it to userId. Closes the result set and the prepared statement. If an SQL exception occurs, it prints the stack trace and an error message. Returns the retrieved userId, which defaults to -1 if no matching user is found.

void clearGUI():

Purpose: Hides various GUI components (fields, buttons, labels) associated with the sign-up interface. Steps: Sets visibility to false for: usernameField: Input field for the username. emailLabel: Label for the email input. usernameLabel: Label for the username input. passwordLabel: Label for the password input. passwordField: Input field for the password. emailField: Input field for the email. signupButton: Button triggering the sign-up process. AlreadyRegisteredButton: Button for users who are already registered. messageLabel: Label displaying messages on the GUI. signupLabel: Label indicating the sign-up section.

createSignInGUI():

Purpose: This function creates graphical user interface (GUI) components for the

sign-in section. Steps: Creates GUI elements such as labels, text fields, and buttons for signing in. signInLabel: A label indicating the "Sign In" section. signInUsernameLabel: Label for the username input field. signInPasswordLabel: Label for the password input field. signInUsernameField: Text field for entering the username. signInPasswordField: Text field for entering the password. signInButton: Button to initiate the sign-in process. newuserButton: Button for users who are new and want to sign up. signInMessageLabel: A message label for displaying sign-in related messages. These GUI elements are positioned at specific coordinates and given sizes to create a coherent interface for users to sign in.

clearSignInGUI():

Purpose: This function hides or sets the visibility of GUI components associated with the sign-in section. Steps: Sets the visibility to false for the GUI elements created in createSignInGUI(). This effectively hides these elements from the user interface. signInUsernameField: Hides the username text field. signInPasswordField: Hides the password text field. signInButton: Hides the sign-in button. signInUsernameLabel: Hides the username label. signInPasswordLabel: Hides the password label. newuserButton: Hides the "New User?" button. signInMessageLabel: Hides the sign-in message label. signInLabel: Hides the "Sign In" label.

signupSuccessful(String username, String password, String email):

Purpose: This function checks whether the input fields for username, password, and email are not empty. Explanation: It takes three strings as arguments: username, password, and email. Checks if all three input strings are not empty using the isEmpty() method. Returns true if none of the input strings are empty; otherwise, returns false. This function is used to verify that the essential fields for signing up are filled before attempting to create a new user account.

insertUser(String username, String password, String email):

Purpose: This function inserts a new user into the database with provided username, password, and email. Explanation: Takes three strings as arguments: username, password, and email. Constructs an SQL query to insert these values into the SIGN table of the database. Attempts to execute the SQL query using a PreparedStatement. If successful, it returns true, indicating the user insertion was successful. If an exception occurs (usually due to a duplicate entry violating a unique constraint), it catches the SQLException, prints the stack trace, logs an error message, and returns false.

checkUserCredentials(String enteredUsername, String enteredPassword):

Purpose: This function checks whether the entered username and password match a

record in the database. Explanation: Takes two strings as arguments: enteredUsername and enteredPassword. Constructs an SQL query to check for a matching record in the SIGN table based on the provided username and password. Executes the SQL query using a PreparedStatement. Checks if the result set contains any rows; if it does, it means a user with matching credentials was found, and it returns true. If no matching record is found or an exception occurs during the query, it returns false.

AURDINO CODE:

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <Wire.h>
#include "MAX30100_PulseOximeter.h"
#define REPORTING_PERIOD_MS     100

uint32_t tsLastReport = 0;
#define TCA_ADDRESS 0x70 // Define the I2C address of the ...
    TCA9548A multiplexer

const char *ssid = "GK";
const char *password = "87654321";
PulseOximeter vataSensor;  // MAX30100 sensor named "Vata"
PulseOximeter pittaSensor;
PulseOximeter kaphaSensor;



void onBeatDetected()
{
//Serial.println("Beat!");
}
AsyncWebServer server(80);
float sensorValue1 = 0;
float sensorValue2 = 0;
float sensorValue3 = 0;
unsigned long lastUpdateTime = 0;
const unsigned long updateInterval = 300;
void setup()
{

  Serial.begin(115200);
  Serial.println("Initializing...");
   WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Print the IP address to Serial Monitor
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());


```

```
45   Wire.begin();
46
47   // Select the TCA9548A multiplexer channel for "Vata"
48   Wire.beginTransmission(TCA_ADDRESS);
49   Wire.write(1 << 2); // Enable channel 0 for "Vata"
50   Wire.endTransmission();
51
52   // Initialize Vata sensor
53   if (!vataSensor.begin())
54    {
55         Serial.println("FAILED");
56         for(;;);
57     } else {
58         Serial.println("SUCCESS");
59     }
60  vataSensor.setOnBeatDetectedCallback(onBeatDetected);
61
62
63
64
65   // Select the TCA9548A multiplexer channel for "Vata"
66   Wire.beginTransmission(TCA_ADDRESS);
67   Wire.write(1 << 1); // Enable channel 0 for "Vata"
68   Wire.endTransmission();
69
70   // Initialize Vata sensor
71   if (!pittaSensor.begin())
72    {
73       Serial.println("FAILED");
74         for(;;);
75     } else {
76       Serial.println("SUCCESS");
77     }
78  pittaSensor.setOnBeatDetectedCallback(onBeatDetected);
79
80
81   // Select the TCA9548A multiplexer channel for "Vata"
82   Wire.beginTransmission(TCA_ADDRESS);
83   Wire.write(1 << 1); // Enable channel 0 for "Vata"
84   Wire.endTransmission();
85
86   // Initialize Vata sensor
87   if (!kaphaSensor.begin())
88    {
89         Serial.println("FAILED");
90         for(;;);
91     } else {
92         Serial.println("SUCCESS");
93     }
```

```
94    kaphaSensor.setOnBeatDetectedCallback(onBeatDetected);
95
96      server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
97      // Increment the counter for each request
98      String json = String(0) + "," + String(sensorValue1) + "," + ...
           String(sensorValue2) + "," + String(sensorValue3);
99      AsyncWebServerResponse *response = ...
           request->beginResponse(200, "application/json", json);
100
101     // Add the Connection: keep-alive header
102     response->addHeader("Connection", "keep-alive");
103
104     // Send the response
105     request->send(response);
106 }); // <- Add this closing parenthesis
107 server.begin();
108   // Select the TCA9548A multiplexer channel for "Pitta"
109 }
110 void loop()
111 {
112     // Make sure to call update as fast as possible
113    vataSensor.update();
114     pittaSensor.update();
115     kaphaSensor.update();
116     if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
117         ///Serial.print("Heart rate vata:");
118           sensorValue1 = vataSensor.getHeartRate();
119           sensorValue2 = pittaSensor.getHeartRate();
120           sensorValue3 = kaphaSensor.getHeartRate();
121
122             Serial.print(sensorValue1);
123             Serial.print(",");
124             Serial.print(sensorValue2);
125             Serial.print(",");
126             Serial.print(sensorValue3);
127             Serial.println("");
128
129         //Serial.print("Heart rate pitta:");
130         //Serial.println(pittaSensor.getHeartRate());
131         //Serial.print("Heart rate kapha:");
132         //Serial.println(kaphaSensor.getHeartRate());
133         tsLastReport = millis();
134     }
135 }
```

AURDINO EXPLANATION:

The variation in the magnitudes of frequency response is successfully analyzed. This code configures an ESP32 microcontroller to connect to WiFi, initialize three MAX30100 Pulse Oximeter sensors ('Vata', 'Pitta', 'Kapha'), and sets up an AsyncWebServer. The setup includes multiplexer channel selections for the sensors, sensor initialization, and callbacks for heartbeat detection. The main loop continuously updates sensor data and reports the heart rates of the sensors over the Serial Monitor at a reporting period of 100 milliseconds. Additionally, it responds to HTTP GET requests with a JSON containing sensor data. However, there are potential issues with multiplexer channel selection duplication and an empty callback function for heartbeat detection.

Database Code

```
CREATE DATABASE SIGNUP ;
USE SIGNUP;
CREATE TABLE SIGN (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL CHECK (CHAR_LENGTH(email) = 10),
    password1 VARCHAR(255) NOT NULL CHECK ...
        (CHAR_LENGTH(password1) ≥ 8)
);


SELECT * FROM SIGN ;
delete table PULSE;
DROP TABLE SIGN;
DROP TABLE PULSE;
CREATE TABLE PULSE (
    id INT,
    VATA FLOAT,
    PIIA FLOAT,
    KAPHA FLOAT,
    FOREIGN KEY (id) REFERENCES SIGN(id)
);
ALTER TABLE PULSE
ADD COLUMN DA_TE DATETIME;

-- Create a trigger to set the DA_TE column to the current date ...
    and time on INSERT
DELIMITER //
CREATE TRIGGER before_pulse_insert
BEFORE INSERT ON PULSE
FOR EACH ROW
SET NEW.DA_TE = NOW();
//
DELIMITER ;

SELECT * FROM PULSE ;
INSERT INTO PULSE (VATA, PIIA, KAPHA)
VALUES (7.8, 9.0, 1.2);
```

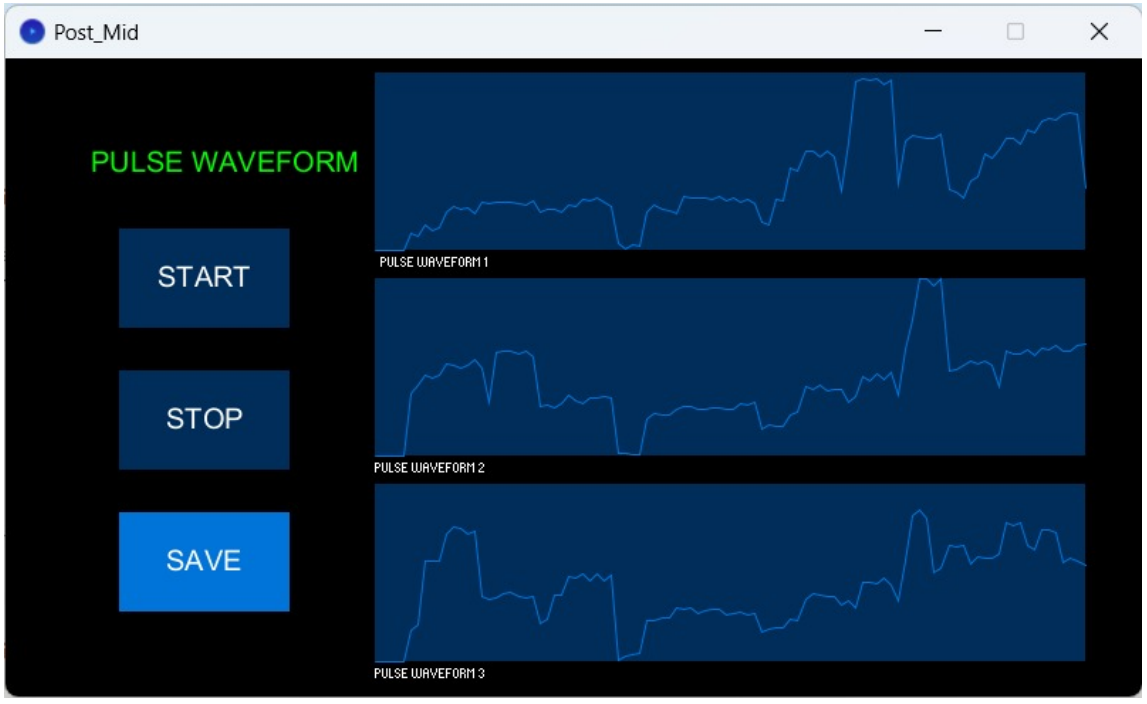
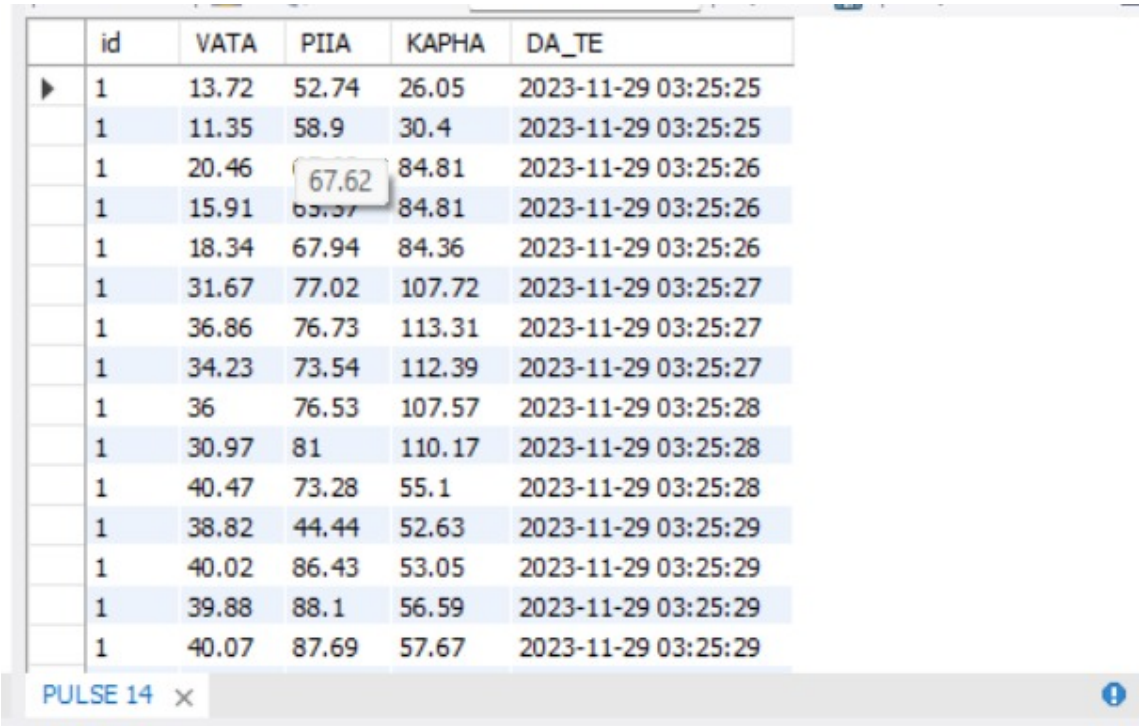


Figure 1: Pulse Waveforms

SQL Database Code Explaination:

The code creates a database called 'SIGNUP', adds tables 'SIGN' and 'PULSE', linking 'PULSE' to 'SIGN' via a foreign key. 'SIGN' stores user information, ensuring the email length is 10 characters and the password length is at least 8 characters. The 'PULSE' table holds values related to VATA, PIIA, and KAPHA, along with a 'DATE' column updated by a trigger to capture insertion date/time. Lastly, it attempts to insert values into the 'PULSE' table, but there might be an error due to missing 'id' value for the foreign key reference to 'SIGN'.



id	VATA	PIIA	KAPHA	DA_TE
1	13.72	52.74	26.05	2023-11-29 03:25:25
1	11.35	58.9	30.4	2023-11-29 03:25:25
1	20.46	67.62	84.81	2023-11-29 03:25:26
1	15.91	[illegible]	84.81	2023-11-29 03:25:26
1	18.34	67.94	84.36	2023-11-29 03:25:26
1	31.67	77.02	107.72	2023-11-29 03:25:27
1	36.86	76.73	113.31	2023-11-29 03:25:27
1	34.23	73.54	112.39	2023-11-29 03:25:27
1	36	76.53	107.57	2023-11-29 03:25:28
1	30.97	81	110.17	2023-11-29 03:25:28
1	40.47	73.28	55.1	2023-11-29 03:25:28
1	38.82	44.44	52.63	2023-11-29 03:25:29
1	40.02	86.43	53.05	2023-11-29 03:25:29
1	39.88	88.1	56.59	2023-11-29 03:25:29
1	40.07	87.69	57.67	2023-11-29 03:25:29

PULSE 14

Figure 2: Database