REAL TIME MONITORING SYSTEM

A MINI-PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of

BACHELOR ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2024

Submitted by

NAME	REGISTER.NO
KANNAN R	230701519
SANJEEV ADITYA P	230701292

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2024

BACHELOR ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project "REAL TIME MONITORING SYSTEM" is the

Bonafide Work Of "**KANNAN R**, **SANJEEV ADITYA P**" Who Carried Out The Project

Work Under My Supervision.

SIGNATURE	**SIGNATURE**
MR.G.SARAVANA GOKUL	**MRS V JANANEE**
ASSISTANT PROFESSOR	**ASSISTANT PROFESSOR**

This mini project report is submitted for the viva voce examination to be held on _____

Table of contents

PG.NO

Abstract:

This program is a graphical user interface (GUI) application developed in Java using Swing that interacts with a MySQL database to perform various operations on temperature and humidity data, which are collected by an Arduino Uno. The Arduino Uno is connected to temperature and humidity sensors (such as DHT11 or DHT22), and it continuously measures and sends the data to a MySQL database via a communication interface (e.g., serial communication or Wi-Fi module).

The Java application allows users to retrieve the maximum, minimum, and average values of the recorded temperature and humidity data from specified columns and tables in the database. It also enables users to view all records from a given table. The program uses JDBC to establish a connection to the MySQL database, execute SQL queries, and display results in a JTextArea within the GUI.

The GUI consists of text fields for user input (to specify the column and table names), buttons for executing queries, and a result display area. Additionally, the application includes a background image and handles basic error scenarios such as empty inputs or database connection issues. The program supports operations like MAX, MIN, and AVG on selected columns, and can display the entire content of a table.

This application demonstrates the integration of Arduino-based sensors with a Java Swing GUI and a MySQL database, providing real-time monitoring and querying capabilities for temperature and humidity data. Future improvements could include enhanced error handling, protection against SQL injection, and optimizations for handling large data sets.

Brief Introduction of the Overall Process:

The project involves the integration of an Arduino Uno microcontroller with a Java-based graphical user interface (GUI) application to monitor and interact with temperature and humidity data stored in a MySQL database. The system is designed to collect real-time environmental data (temperature and humidity) using sensors connected to the Arduino Uno, and then store this data in a MySQL database for further analysis.

Process Overview:

1. **Arduino Data Collection**:

   - The Arduino Uno is connected to temperature and humidity sensors (such as DHT11 or DHT22). These sensors continuously measure the environmental conditions and send the data to the Arduino.
   - The Arduino Uno is programmed to send the collected data to a MySQL database via a communication module (e.g., Wi-Fi or serial communication).

2. **Database Storage**:
    - The temperature and humidity data received from the Arduino is inserted into a MySQL database. The database stores the readings along with their corresponding timestamps for historical data analysis.
3. **Java GUI Interface**:
    - A Java Swing-based GUI application is developed, which provides a user-friendly interface to interact with the database. The application includes features to retrieve the maximum, minimum, and average values of the temperature and humidity data, as well as view all records stored in the database.
    - Users can input the column (e.g., temperature or humidity) and table name through the GUI, and perform database queries to retrieve the requested information.
4. **SQL Query Execution**:
    - The Java application communicates with the MySQL database using JDBC (Java Database Connectivity). Upon user interaction, it constructs SQL queries (such as SELECT MAX, SELECT MIN, SELECT AVG) to fetch the required data from the database.
    - The results of the queries are displayed in the GUI, where users can view the computed values or the entire dataset.

5. **Real-Time Data Monitoring**:
    - The system allows users to monitor temperature and humidity data in real-time. It supports various SQL operations to extract meaningful insights from the stored data and helps track environmental changes over time.

Overall, this system provides an end-to-end solution for collecting, storing, querying, and visualizing temperature and humidity data using Arduino, MySQL, and Java Swing, offering users a simple and efficient way to interact with environmental data.

AURDINO UNO R3:

Arduino Uno R3 Overview

The **Arduino Uno R3** is one of the most popular microcontroller boards in the Arduino family, widely used for educational purposes, prototyping, and DIY electronics projects. The Uno R3 is an open-source hardware platform based on the **ATmega328P** microcontroller and provides an easy-to-use development environment for creating and testing a wide range of electronic circuits and applications.

Key Features of Arduino Uno R3:

1. **Microcontroller**:
    - The Arduino Uno R3 is powered by the **ATmega328P** microcontroller, which is an 8-bit microcontroller from Atmel (now part of Microchip). This microcontroller has a clock speed of **16 MHz**, allowing it to handle basic processing tasks.

2. **Input/Output Pins**:
    - The board includes **14 digital input/output pins**, which can be configured as either inputs or outputs. These pins can be used to interface with various sensors, actuators, LEDs, buttons, and other electronic components.
    - It also has **6 analog input pins** for reading analog signals from sensors, such as temperature sensors, light sensors, etc.

USB Connectivity:

- The Uno R3 has a **USB Type-B** port that is used to connect the board to a computer for programming. This connection also supplies power to the board while it's being programmed or operated.
- The USB interface is handled by an **ATmega16U2** microcontroller, which acts as a bridge between the computer and the ATmega328P.

3. **Power Supply**:
    - The Arduino Uno R3 can be powered through the **USB port** or an external power supply (7V to 12V). If an external supply is used, it is regulated by the onboard voltage regulator, which outputs **5V** to power the board and connected components.

4. **Onboard LEDs**:
    - The board has a built-in **LED** connected to digital pin 13. This LED can be controlled programmatically to test simple outputs or for debugging purposes.

5. **Reset Button**:
    - The board features a **reset button**, which restarts the program running on the microcontroller. Pressing the reset button reboots the board and restarts the program from the beginning.

6. **Programming Interface**:
    - The Arduino Uno R3 is programmed using the **Arduino IDE (Integrated Development Environment)**, which provides a simple and easy-to-use interface for writing and uploading code to the board. The IDE uses a language based on C/C++ with pre-built functions to simplify coding.

7. **Storage (Flash Memory)**:
    - The Uno R3 has **32 KB of flash memory**, which is used to store the program code. Of this, 0.5 KB is used by the bootloader, and the rest is available for user programs.
    - It also has **2 KB of SRAM** (static RAM) for variables during program execution and **1 KB of EEPROM** (Electrically Erasable Programmable Read-Only Memory) for storing data that persists even when the board is powered off.

8. **Clock Speed**:
    - The board operates at a **16 MHz clock speed**, which determines how fast the microcontroller can execute instructions.

Communication:

The Arduino Uno R3 supports different types of communication:

- **Serial Communication**: The board can communicate with other devices through its **UART (Universal Asynchronous Receiver/Transmitter)** via digital pins 0 (RX) and 1 (TX).
- **I2C**: The Uno has **SCL** (clock) and **SDA** (data) pins (A5 and A4, respectively), which are used for I2C communication with compatible devices.
- **SPI**: The Uno R3 also supports **SPI (Serial Peripheral Interface)** using the pins 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK) for fast communication with peripheral devices like sensors and SD cards.

Programming the Arduino Uno R3:

To program the Arduino Uno R3:

1. Install the **Arduino IDE** on your computer.
2. Connect the Uno R3 to your computer via the USB port.
3. Write your program (called a sketch) in the Arduino IDE, using simple functions like digitalWrite(), digitalRead(), analogWrite(), and analogRead().
4. Upload the program to the Uno using the **Upload button** in the Arduino IDE. The Arduino IDE will compile the code and transfer it to the board.

5. Once the code is uploaded, the program will start running on the board, and you can monitor its behavior using the **Serial Monitor**.

Applications of Arduino Uno R3:

The Arduino Uno R3 is widely used in various fields and applications:

1. **Robotics**: Arduino Uno is commonly used as the controller for robotic projects, controlling motors, sensors, and actuators.
2. **Home Automation**: It is used for smart home projects like controlling lights, fans, or temperature regulation via sensors.
3. **IoT Projects**: The Uno can be interfaced with Wi-Fi modules (like the **ESP8266**) or Bluetooth modules for creating Internet of Things (IoT) projects.
4. **Data Logging**: It can be used to collect and store data from sensors (e.g., temperature, humidity, pressure) and store it on an SD card or send it to a cloud server.
5. **Educational Projects**: Due to its simplicity and affordability, it is widely used for teaching the basics of programming and electronics.

Advantages of Arduino Uno R3:

- **Easy to Use**: The Arduino IDE and its simplified C/C++ language make it accessible for beginners.
- **Large Community and Resources**: There is a large online community, providing extensive libraries, tutorials, and support.

- **Flexible**: It can be used in a wide variety of projects, from simple LED blinkers to complex robots and home automation systems.
- **Open Source**: Both the hardware and software of the Arduino Uno are open-source, allowing users to modify and share their designs freely.

DHT11 Sensor Overview:

The **DHT11** is a low-cost, digital temperature and humidity sensor commonly used in electronics projects, especially with microcontrollers like **Arduino**. It is used to measure the temperature (in °C) and humidity (in %RH) of the surrounding environment.

Key Features:

1. **Temperature Range**: 0 to 50°C (±2°C accuracy).
2. **Humidity Range**: 20% to 80% RH (±5% accuracy).
3. **Output**: The DHT11 provides a **digital signal** to the microcontroller, which simplifies the wiring and communication.
4. **Operating Voltage**: 3.3V to 5.5V, making it compatible with most common microcontrollers.
5. **Sampling Rate**: Data is updated once every **1 second**.

Working Principle:

- The DHT11 sensor contains a **capacitive humidity sensor** and a **thermistor** for measuring temperature.
- It uses **single-wire communication**, sending digital data to a microcontroller like Arduino.

- The Arduino reads the temperature and humidity values from the sensor and processes them for display or control in various applications.

Applications:

- Weather stations.
- Environmental monitoring (e.g., greenhouses).
- Home automation systems (e.g., controlling HVAC based on temperature/humidity).

How DHT11 Sensor Interacts with Arduino Uno R3:

The **DHT11** sensor is a simple and inexpensive temperature and humidity sensor commonly used with the **Arduino Uno R3** to measure environmental conditions. The interaction between the DHT11 sensor and the Arduino Uno involves reading temperature and humidity data from the sensor and processing it on the Arduino to perform actions, display data, or control other components.

Components Needed:

1. **DHT11 Sensor**: For measuring temperature and humidity.
2. **Arduino Uno R3**: Microcontroller to interface with the DHT11 and process the data.
3. **Jumper Wires**: For connecting the DHT11 to the Arduino.
4. **Resistor**: A 10kΩ pull-up resistor is typically used between the **VCC** and **DATA** pins of the DHT11 to ensure proper communication.

Pinout of the DHT11:

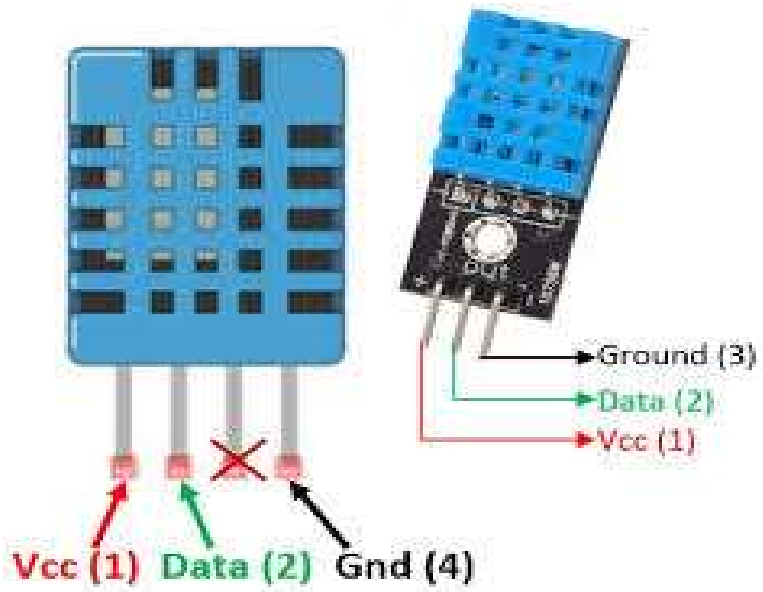
1. **VCC**: Provides power to the sensor (typically 5V).

2. **GND**: Ground pin to complete the circuit.
3. **DATA**: This is the data pin that sends temperature and humidity readings to the Arduino.

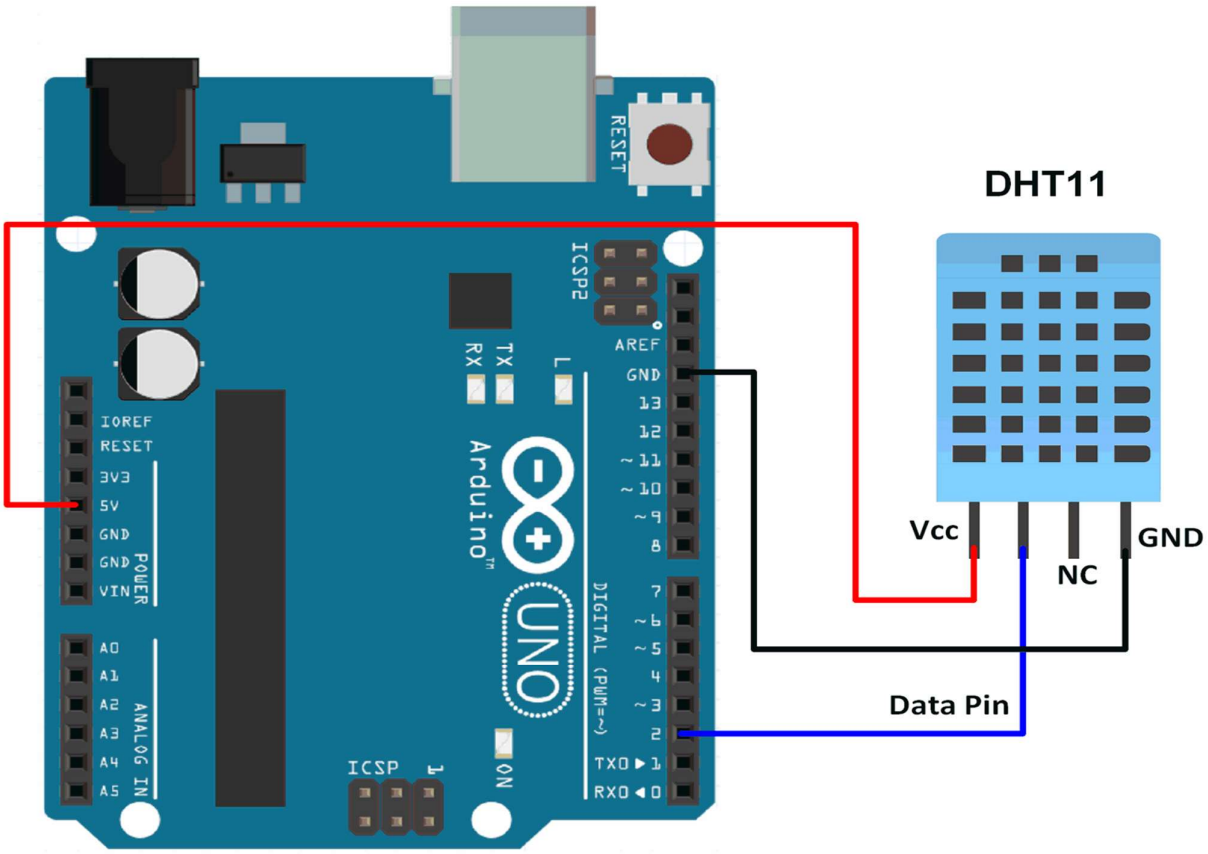
Wiring the DHT11 to the Arduino Uno R3:

- **VCC** pin of the DHT11 is connected to the **5V** pin of the Arduino.
- **GND** pin of the DHT11 is connected to the **GND** pin of the Arduino.
- **DATA** pin of the DHT11 is connected to a digital I/O pin (e.g., **Pin 2**) of the Arduino.

A **10kΩ resistor** is placed between the **VCC** and **DATA** pins as a pull-up resistor to ensure reliable data transmission.



Process of Interaction Between DHT11 and Arduino Uno R3:



1. **Powering Up the System**:
    - When the Arduino is powered on, the **VCC** and **GND** pins supply power to the DHT11 sensor, enabling it to operate.
2. **Reading the Data from DHT11**:
    - The **DATA** pin of the DHT11 sends the digital temperature and humidity data to the Arduino. The data is sent in a specific format, starting with an 8-bit humidity integer, followed by an 8-bit humidity decimal part, an 8-bit temperature integer, and an 8-bit temperature decimal part.

- The Arduino reads this data using a specific communication protocol. Arduino libraries like the **DHT** library simplify this process. The library provides functions to read the temperature and humidity from the sensor, handling the timing and protocols.

3. **Processing the Data on Arduino**:
   - Once the data is read, the Arduino processes it. The **temperature** and **humidity** are extracted from the received byte data and converted into readable values (e.g., Celsius for temperature and percentage for humidity).
4. **Displaying or Using the Data**:After processing the data, the Arduino can perform several tasks. For example, it can print the readings to the **Serial Monitor**, display the data on an LCD, or use the readings to control other components, like turning on a fan if the humidity exceeds a certain threshold.

Example Code for Interfacing DHT11 with Arduino Uno R3:

```
#include <DHT.h>
#define DHTPIN 2          // Pin where the DHT11 sensor is connected
#define DHTTYPE DHT11     // Define the type of sensor (DHT11 in this case)
DHT dht(DHTPIN, DHTTYPE); // Initialize the DHT sensor
void setup() {
  Serial.begin(9600);     // Start serial communication at 9600 baud rate
  dht.begin();            // Initialize the DHT11 sensor
}
void loop() {
  delay(2000);  // Wait for 2 seconds between readings
 // Read temperature in Celsius
 float temperature = dht.readTemperature();
```

```
 // Read humidity
  float humidity = dht.readHumidity();
 // Check if reading failed
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
// Print the temperature and humidity values to the Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C ");
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
}
```

Key Steps in the Code:

1. **Including the DHT Library**: The **DHT.h** library is included to provide functions for reading from the DHT11 sensor.
2. **Defining the Pin**: The DHTPIN constant is defined to indicate which Arduino pin the **DATA** pin of the DHT11 is connected to.
3. **Initializing the Sensor**: The dht.begin() function initializes the DHT11 sensor.
4. **Reading Temperature and Humidity**: The functions dht.readTemperature() and dht.readHumidity() read the temperature and humidity values from the sensor.

5. **Checking for Errors**: The code checks whether the readings are valid. If there's an issue, it will print an error message.
6. **Displaying the Data**: The temperature and humidity values are printed to the Serial Monitor.

Timing:

- The **DHT11** sensor has a sampling rate of **1Hz**, meaning the sensor sends data approximately every **1 second**. Therefore, the Arduino waits 2 seconds between readings using delay(2000) to ensure the sensor data is refreshed.

```
void loop()
{ delay(2000); // Wait for 2 seconds (DHT11 requires time between readings)
}
```

MAIN CODE:

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class DatabaseOperationGUI extends JFrame {
    private static final String URL = "jdbc:mysql://localhost:3306/db";
    private static final String USER = "root";
    private static final String PASSWORD = "09032006KDob";

    private JTextArea resultArea;
    private JTextField columnField;
    private JTextField tableField;
    private Image backgroundImage;

    public DatabaseOperationGUI() {
        setTitle("RTMS");
        setSize(800, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        try {
            backgroundImage = ImageIO.read(new File("C:\\Users\\kannan\\Downloads\\gradient-blue-pink-abstract-art-wallpaper-preview"));
        } catch (IOException e) {
            System.out.println("Error loading background image: " + e.getMessage());
```

```
}

columnField = new JTextField(20);
tableField = new JTextField(20);
JButton maxButton = new JButton("Get Max");
JButton minButton = new JButton("Get Min");
JButton avgButton = new JButton("Get Avg");
JButton selectAllButton = new JButton("History");
resultArea = new JTextArea();
resultArea.setEditable(false);
resultArea.setForeground(Color.pink);
resultArea.setBackground(Color.BLACK);
JScrollPane scrollPane = new JScrollPane(resultArea);

JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());

JPanel inputPanel = new JPanel();
inputPanel.setLayout(new GridLayout(2, 2, 10, 10));
inputPanel.add(new JLabel("TEMPERATURE or HUMIDITY:"));
inputPanel.add(columnField);
inputPanel.add(new JLabel("PLACE:"));
inputPanel.add(tableField);
panel.add(inputPanel, BorderLayout.NORTH);

JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new FlowLayout());
buttonPanel.add(maxButton);
buttonPanel.add(minButton);
```

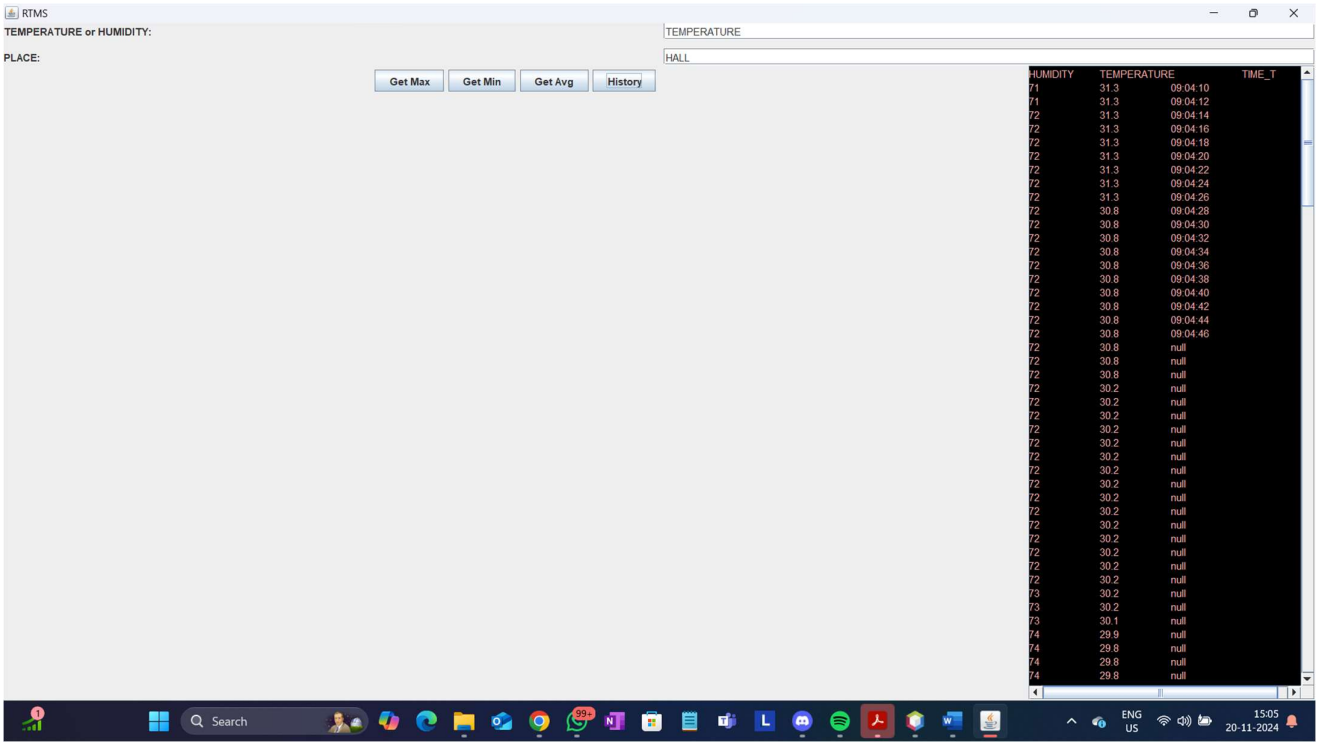
```
    buttonPanel.add(avgButton);
    buttonPanel.add(selectAllButton);
    panel.add(buttonPanel, BorderLayout.CENTER);

    panel.add(scrollPane, BorderLayout.EAST);

    maxButton.addActionListener(e -> executeQuery("MAX"));
    minButton.addActionListener(e -> executeQuery("MIN"));
    avgButton.addActionListener(e -> executeQuery("AVG"));
    selectAllButton.addActionListener(e -> selectAllRecords());

    add(panel);
}

@Override
public void paint(Graphics g) {
    super.paint(g);
    if (backgroundImage != null) {
        g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);
    }
}

private void executeQuery(String operation) {
    String column = columnField.getText().trim();
    String table = tableField.getText().trim();
    resultArea.setText("");

    if (column.isEmpty() || table.isEmpty()) {
        resultArea.setText("Please provide both a column and a table name.");
```

```
            return;
        }

        String query = String.format("SELECT %s(%s) FROM %s", operation, column, table);

        try (Connection con = DriverManager.getConnection(URL, USER, PASSWORD);
             Statement st = con.createStatement();
             ResultSet rs = st.executeQuery(query)) {

            if (rs.next()) {
                resultArea.setText(operation + " of " + column + ": " + rs.getString(1));
            } else {
                resultArea.setText("No results found.");
            }

        } catch (SQLException e) {
            resultArea.setText("Error connecting to database. Please check your connection settings.");
        }
    }

    private void selectAllRecords() {
        String table = tableField.getText().trim();
        resultArea.setText("");

        if (table.isEmpty()) {
            resultArea.setText("Please provide a table name.");
            return;
        }
```

```
String query = String.format("SELECT * FROM %s", table);

try (Connection con = DriverManager.getConnection(URL, USER, PASSWORD);
     Statement st = con.createStatement();
     ResultSet rs = st.executeQuery(query)) {

    StringBuilder results = new StringBuilder();
    ResultSetMetaData metaData = rs.getMetaData();
    int columnCount = metaData.getColumnCount();

    for (int i = 1; i <= columnCount; i++) {
        results.append(metaData.getColumnName(i)).append("\t");
    }
    results.append("\n");

    while (rs.next()) {
        for (int i = 1; i <= columnCount; i++) {
            results.append(rs.getString(i)).append("\t");
        }
        results.append("\n");
    }

    if (results.length() == 0) {
        resultArea.setText("No records found.");
    } else {
        resultArea.setText(results.toString());
    }
```

```
        } catch (SQLException e) {
            resultArea.setText("SQL Exception: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            DatabaseOperationGUI gui = new DatabaseOperationGUI();
            gui.setVisible(true);
        });
    }
}
```

OUTPUT:



Overview of the Code:

The code you've shared is for a **Java Swing application** that connects to a **MySQL database** and performs basic **database operations** like retrieving the maximum, minimum, and average values of a given column, or displaying the full records from a specified table. The user interacts with the application via a graphical user interface (GUI), and the results of database queries are displayed in a text area.

Key Components of the Application:

1. **JDBC Connection**: The code uses JDBC (Java Database Connectivity) to connect to a MySQL database. The URL, USER, and PASSWORD constants are used to establish the connection to the MySQL server.
2. **GUI Components**:
    - **JTextField**: Two text fields are used for user input: one for specifying the column name (e.g., "temperature" or "humidity") and another for specifying the table name.
    - **JButton**: Buttons like "Get Max", "Get Min", "Get Avg", and "History" allow the user to interact with the application and perform the corresponding database operations.
    - **JTextArea**: This component is used to display the results of the database queries. The text area is set to have a black background and pink text to improve visibility.
    - **JScrollPane**: It ensures that the results in the text area are scrollable if the data is too large to fit within the visible area.
3. **Database Operations**:
    - The executeQuery method is used to perform the "MAX", "MIN", and "AVG" operations on a specified column of a given table. The SQL query is dynamically constructed using the input column and table values.
    - The selectAllRecords method retrieves all records from the specified table and displays them in the text area, including column names and their respective data.
4. **Background Image**: A background image is loaded and displayed using paint(Graphics g) method. The image is loaded from the file system and

drawn onto the frame using g.drawImage(). This makes the UI more visually appealing.

How It Works:

1. **User Input**:
    - The user enters the **column name** (e.g., "temperature" or "humidity") in the columnField.
    - The user enters the **table name** (e.g., "sensor_data") in the tableField.
2. **Button Click**:
    - When the user clicks one of the operation buttons ("Get Max", "Get Min", "Get Avg"), an ActionListener triggers the execution of the corresponding SQL query. The executeQuery method runs the appropriate query (e.g., SELECT MAX(temperature) FROM sensor_data).
3. **SQL Query Execution**:
    - The query is sent to the MySQL database using JDBC.
    - The result is returned and displayed in the resultArea text area.
4. **Displaying Results**:
    - If the query retrieves data, the result (e.g., maximum temperature or humidity) is displayed.
    - If no results are found, an appropriate message ("No results found") is shown.
    - If an error occurs while connecting to the database or executing the query, an error message is displayed.

Example of Interaction:

1. **Max Temperature**:
    - The user enters temperature in the column field and sensor_data in the table field.
    - The user clicks "Get Max".
    - The query SELECT MAX(temperature) FROM sensor_data is executed, and the result is displayed in the result area.

2. **View All Records**:
    - The user enters a table name and clicks the "History" button.
    - The selectAllRecords method executes a SELECT * FROM <table> query and displays all records from the specified table in the text area.

Summary of Key Methods:

1. **executeQuery(String operation)**:
    - Executes SQL queries for the operations (MAX, MIN, AVG).
    - Constructs the query dynamically using the input column and table.
    - Executes the query and processes the result.
2. **selectAllRecords()**:
    - Retrieves all records from the specified table and displays them in the result area.
    - It includes the column names as headers and the data for each row.

Error Handling:

- If the input fields are empty, or if an exception occurs during database operations, the program handles errors by displaying a message in the text area. This ensures the user is informed of any issues during the query execution.

Enhancements:

- **Validation**: The program checks whether the column and table names are provided before executing a query.
- **Error Messages**: Proper error messages are displayed when there are database connection issues or query failures.