



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY
TECHNISCHE UNIVERSITÄT MÜNCHEN

INGENIEURPRAXISBERICHT

Implementierung eines Sensornetzwerks für optische Labore

Mingi Kang

1. Januar 2025

Inhaltsverzeichnis

1	Einleitung	2
1.1	Motivation und Ziel	2
2	Technischer Hintergrund	3
2.1	Hardware	3
2.2	Datenbank	8
2.2.1	Datenbankeinrichtung	8
2.2.2	Datenbank-Schema	9
2.2.3	Datenverwaltung	9
2.2.4	Backup älterer Daten	9
2.2.5	Zusammenfassung	10
3	Implementierung	11
3.1	Python-Skript für Datenverarbeitung	11
3.1.1	Webschnittstelle	14
4	Ergebnisse	18
4.1	Serverbetrieb und Funktionsweise	18
4.2	Visualisierung der Ergebnisse	20
5	Diskussion und Ausblick	23
5.1	Erreichte Ziele	23
5.2	Zukünftige Arbeiten	25

Kapitel 1

Einleitung

1.1 Motivation und Ziel

In hochsensiblen Forschungsumgebungen wie einem experimentellen Laser- und Quantentechnologie-Labor ist die präzise Überwachung von Temperatur und Feuchtigkeit von entscheidender Bedeutung. Schwankungen in den Umgebungsbedingungen können die Ergebnisse von Experimenten erheblich beeinflussen und zu ungenauen oder gar fehlerhaften Messungen führen. Eine stabile Umgebung ist daher essenziell, um reproduzierbare Ergebnisse zu gewährleisten und die empfindliche experimentelle Ausrüstung zu schützen.

Das Ziel dieses Projekts ist es, ein Sensorsystem zu entwickeln, das die Umgebungsbedingungen eines Labors in Echtzeit überwacht und speichert. Dieses System soll auf einem **SHTC3-Temperatur- und Feuchtigkeitssensor**, einem **Arduino Nano**, sowie einem **Raspberry Pi** basieren. Der Sensor erfasst die aktuellen Werte für Temperatur und Luftfeuchtigkeit, die anschließend über den Arduino an den Raspberry Pi übertragen werden. Der Raspberry Pi speichert die gesammelten Daten in einer lokalen MariaDB-Datenbank und stellt diese über eine benutzerfreundliche Webschnittstelle dar.

Die Implementierung dieses Systems umfasst folgende zentrale Schritte:

- Die Entwicklung eines zuverlässigen Codes für die Integration des SHTC3-Sensors mit dem Arduino Nano.
- Die Einrichtung eines Systems zur seriellen Datenübertragung zwischen dem Arduino und dem Raspberry Pi.
- Die Konfiguration des Raspberry Pi zur Datenverarbeitung, Speicherung und Visualisierung der erfassten Daten.

Langfristig dient dieses Projekt als Grundlage für die Integration in ein automatisiertes Steuerungssystem, das in der Lage ist, Temperatur und Feuchtigkeit aktiv zu regeln. Insbesondere soll das System mit einer Klimaanlage gekoppelt werden, um die Laborbedingungen optimal zu halten. Darüber hinaus wird angestrebt, die gesammelten Daten nicht nur lokal verfügbar zu machen, sondern sie auch über eine Internetverbindung zugänglich zu gestalten. Dies ermöglicht es den Forschenden, die Labordaten auch außerhalb der Einrichtung in Echtzeit einzusehen und darauf zu reagieren.

Dieses Projekt liefert somit nicht nur eine Lösung für die aktuelle Herausforderung der Echtzeit-Überwachung, sondern schafft auch die Basis für zukünftige Erweiterungen und Anwendungen in der Klimatisierung und Datenanalyse.

Kapitel 2

Technischer Hintergrund

2.1 Hardware

Arduino Nano und Sensor

In diesem Projekt wurde der **SHTC3-Temperatur- und Feuchtigkeitssensor** verwendet, der über den EasyC-Adapter mit dem **Arduino Nano** verbunden wurde. Der EasyC-Adapter ermöglicht eine einfache Verbindung zwischen Sensor und Mikrocontroller, wobei die I2C-Schnittstelle genutzt wird.

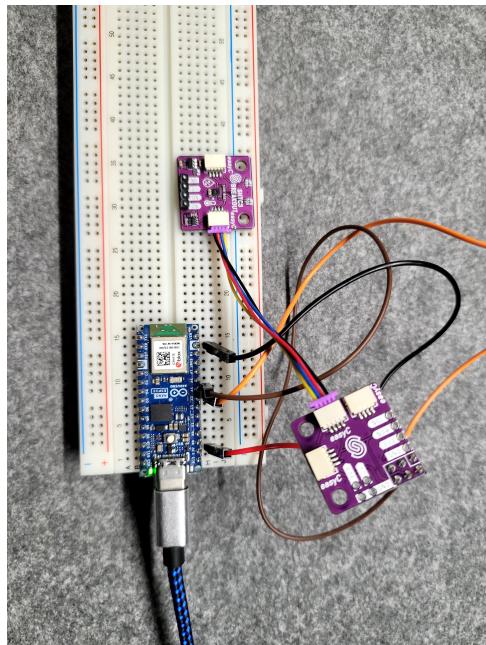


Abbildung 2.1: Verkabelung des SHTC3-Sensors mit dem Arduino Nano über den EasyC-Adapter.

Verkabelung Die Verbindung zwischen dem SHTC3-Sensor und dem Arduino Nano wurde wie folgt hergestellt:

- Der **SDA-Pin** des Sensors wurde mit dem **A4-Pin (SDA)** des Arduino Nano verbunden.

- Der **SCL-Pin** des Sensors wurde mit dem **A5-Pin (SCL)** des Arduino Nano verbunden.
- Die Stromversorgung des Sensors erfolgte über den **3.3V-Pin** und den **GND-Pin** des Arduino Nano.

Da der EasyC-Adapter die Sensoren bereits physisch verbindet, war keine manuelle Lötung notwendig.

I2C-Kommunikation Die I2C-Adresse des Sensors wurde standardmäßig auf 0x70 eingestellt. Um sicherzustellen, dass der Sensor ordnungsgemäß erkannt wird, wurde ein Test mit folgendem Arduino-Code durchgeführt:

Listing 2.1: Arduino-Code zur Überprüfung der I2C-Verbindung

```

1 #include <Wire.h>
2
3 void setup() {
4     Wire.begin();
5     Serial.begin(9600);
6     Serial.println("Scanning I2C devices...");
7     for (byte address = 1; address < 127; address++) {
8         Wire.beginTransmission(address);
9         if (Wire.endTransmission() == 0) {
10             Serial.print("Found I2C device at address: 0x");
11             Serial.println(address, HEX); // I2C-Adresse wird
12             angezeigt
13         }
14     }
15
16 void loop() {
17     // Keine weiteren Funktionen im Loop notwendig
18 }
```

Die Ausgabe dieses Codes im Arduino-Serial-Monitor zeigte die Adresse 0x70, was die erfolgreiche Verbindung des Sensors bestätigte.

Hauptprogramm für den Sensorbetrieb Nach der erfolgreichen Verbindung und Überprüfung wurde der Hauptcode implementiert, um die Temperatur- und Feuchtigkeitswerte vom Sensor zu lesen. Der Code verwendet die Adafruit-Bibliothek Adafruit_SHTC3.h, die speziell für diesen Sensor entwickelt wurde.

Listing 2.2: Arduino-Hauptprogramm für SHTC3-Sensorbetrieb

```

1 #include <Adafruit\_SHTC3.h> // Bibliothek fuer SHTC3-Sensor
2 Adafruit_SHTC3 shtc3;
3
4 void setup() {
5     Serial.begin(9600); // Serielle Verbindung initialisieren
6     if (!shtc3.begin()) {
7         Serial.println("Couldn't find SHTC3"); // Fehlerausgabe, wenn
8         Sensor nicht erkannt wird
9         while (1);
10    }
11    Serial.println("SHTC3 found and initialized.");
```

```

12
13 void loop() {
14     sensors_event_t temp, humidity;
15     if (shtc3.getEvent(&temp, &humidity)) {
16         Serial.print("Temperature: ");
17         Serial.print(temp.temperature); // Temperatur wird ausgegeben
18         Serial.print(" °C , Humidity: ");
19         Serial.println(humidity.relative_humidity); // Feuchtigkeit
20             wird ausgegeben
21     } else {
22         Serial.println("Failed to read from sensor."); // Fehler beim
23             Lesen des Sensors
24     }
25     delay(1000); // 1-Sekunden-Intervall fuer die Messung
}

```

Erklärung des Codes mit Kommentaren

- `#include <Adafruit_SHTC3.h>`¹.
- `Serial.begin(9600)`².
- `shtc3.begin()`³.
- `shtc3.getEvent()`⁴.
- `delay(1000)`⁵.

Zusammenfassung Die erfolgreiche Integration des Sensors mit dem Arduino Nano und die Kommunikation über I2C ermöglichen eine präzise Messung der Umgebungsbedingungen. Der Sensor sendet die Daten in Echtzeit über die serielle Schnittstelle an den angeschlossenen Raspberry Pi, der für die weitere Verarbeitung und Speicherung der Daten zuständig ist.

¹Die Bibliothek stellt Funktionen zur Verfügung, um den Sensor anzusteuern. Sie wird über den Arduino Library Manager installiert.

²Initialisiert die serielle Kommunikation mit einer Baudrate von 9600, um Daten im Serial Monitor anzuzeigen.

³Initialisiert die Verbindung mit dem Sensor. Falls der Sensor nicht gefunden wird, bleibt das Programm in einer Schleife stehen.

⁴Liest die aktuellen Temperatur- und Feuchtigkeitswerte aus dem Sensor.

⁵Wartet 1 Sekunde, bevor die nächste Messung erfolgt. Dies ist wichtig, um die Auslastung des Sensors zu reduzieren.

```

TempAndHumidity.ino
1 Adafruit_SHTC3 shtc3;
2
3 void setup() {
4     Serial.begin(115200); // Serial communication
5     Wire.begin(); // I2C
6
7     if (!shtc3.begin()) {
8         Serial.println("Couldn't find SHTC3 sensor!");
9     }
10    while (1) delay(1);
11
12    Serial.println("SHTC3 sensor initialized.");
13}
14
15 void loop() {
16     sensors_event_t humidity, temp;
17
18     if (shtc3.getEvent(&humidity, &temp)) {
19         // JSON
20         Serial.print("{\"temperature\":");
21         Serial.print(temp.temperature);
22         Serial.print(",");
23         Serial.print("humidity\"");
24         Serial.print(humidity.relative_humidity);
25         Serial.println("}");
26     } else {
27         Serial.println("Failed to read from sensor.");
28     }
29
30     delay(1000); // 1 sec
31 }
32

```

Output: Serial Monitor X

Message (Enter to send message to 'Arduino Nano ESP32' on /dev/cu.usbmodem744DBD7708482)

{"temperature":13.41, "humidity":60.34}

{"temperature":13.46, "humidity":60.37}

New Line 9600 baud

Abbildung 2.2: Arduino Nano ESP32 mit Temperatur- und Feuchtigkeitssensor im Serial Monitor

Raspberry Pi

Das **Raspberry Pi 5** wurde in diesem Projekt als zentrale Recheneinheit verwendet. Es dient zur Verarbeitung, Speicherung und Visualisierung der von den Sensoren gesammelten Daten. Zusätzlich ermöglicht es die Kommunikation mit der Arduino-Plattform und fungiert als lokaler Server für die Webanwendung.



Abbildung 2.3: Verbindung zwischen dem Arduino Nano und dem Raspberry Pi mit einer externen Festplatte.

Hardware-Konfiguration Das Raspberry Pi wurde mit den folgenden Komponenten eingerichtet:

- **Netzteil:** Ein 5V/3A-Netzteil zur Stromversorgung des Raspberry Pi.

- **MicroSD-Karte:** Eine 256GB MicroSD-Karte mit Raspberry Pi OS wurde für das Betriebssystem verwendet.
- **USB-Verbindung:** Der Arduino Nano wurde über ein USB-Kabel mit dem Raspberry Pi verbunden.
- **Externe Festplatte:** Eine externe Festplatte wurde über den USB-3.0-Anschluss des Raspberry Pi angeschlossen, um ältere Daten langfristig zu speichern.
- **WLAN-Adapter:** Der Netgear AC1200 WLAN-Adapter wurde für die Erstellung eines lokalen Hotspots konfiguriert.

Softwareinstallation Nach der Hardwarekonfiguration wurden die erforderlichen Softwarepakete und Bibliotheken installiert. Dies erfolgte über den Terminal des Raspberry Pi:

Listing 2.3: Installation von Softwarepaketen auf dem Raspberry Pi

```
1 sudo apt update
2 sudo apt upgrade
3 sudo apt install python3 python3-pip mariadb-server
4 pip3 install flask pymysql adafruit-circuitpython-sh1c3
```

I2C-Schnittstelle aktivieren Um die Kommunikation mit dem Sensor über den EasyC-Adapter zu ermöglichen, musste die I2C-Schnittstelle auf dem Raspberry Pi aktiviert werden:

- Öffnen des Konfigurationsmenüs:

```
1 sudo raspi-config
```

- Navigieren zu: Interfacing Options > I2C > Enable.
- Überprüfen, ob die I2C-Geräte erkannt wurden:

Listing 2.4: Prüfung der I2C-Geräte mit i2cdetect

```
1 sudo i2cdetect -y 1
```

Die Ausgabe sollte den Sensor unter der Adresse 0x70 anzeigen.

Dateisystem und externe Festplatte einrichten Die externe Festplatte wurde als Speicherort für ältere Daten eingerichtet:

- Festplatte anschließen und Gerätedatei ermitteln:

```
1 lsblk
```

- Mounten der Festplatte:

```
1 sudo mount /dev/sda1 /media/pi/MyDrive
```

- Automatisches Mounten beim Systemstart konfigurieren:

```
1 sudo nano /etc/fstab
2      # Zeile hinzufügen:
3      /dev/sda1 /media/pi/MyDrive exfat defaults 0 0
```

Verbindung mit dem Arduino Nano Der Arduino Nano wurde über ein USB-Kabel mit dem Raspberry Pi verbunden. Der Raspberry Pi empfängt die über die serielle Schnittstelle gesendeten Temperatur- und Feuchtigkeitsdaten. Dies wurde mit dem folgenden Python-Skript sichergestellt:

Listing 2.5: Python-Code für die serielle Kommunikation

```

1 import serial
2
3 SERIAL_PORT = '/dev/ttyACM0'
4 BAUD_RATE = 9600
5 ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
6
7 while True:
8     if ser.in_waiting > 0:
9         line = ser.readline().decode('utf-8').strip()
10        print(f"Received data: {line}")

```

Zusammenfassung Das Raspberry Pi ist als zentraler Bestandteil des Systems für die Datenspeicherung, Verarbeitung und Visualisierung verantwortlich. Durch die erfolgreiche Einrichtung der I2C-Schnittstelle und die Konfiguration der externen Festplatte konnte die Infrastruktur für ein stabiles und erweiterbares Datenerfassungssystem geschaffen werden.

2.2 Datenbank

Die Speicherung und Verwaltung der gesammelten Temperatur- und Feuchtigkeitsdaten wurde mithilfe von MariaDB, einer relationalen Datenbank, realisiert. Die Datenbank wurde auf dem Raspberry Pi lokal eingerichtet, um die erfassten Daten effizient zu speichern und zu verarbeiten. Zusätzlich wurde ein Mechanismus zur regelmäßigen Sicherung älterer Daten auf eine externe Festplatte implementiert.

2.2.1 Datenbankeinrichtung

Die Installation und Konfiguration von MariaDB erfolgte über den Terminal des Raspberry Pi:

Listing 2.6: Installation von MariaDB

```

1 sudo apt update
2 sudo apt install mariadb-server
3 sudo mysql_secure_installation

```

Nach der erfolgreichen Installation wurde die Datenbank für die Speicherung der Sensorsdaten erstellt:

Listing 2.7: Erstellung der Datenbank und Tabelle

```

1 CREATE DATABASE sensor_data;
2
3 USE sensor_data;
4
5 CREATE TABLE sensor_data (
6     id INT AUTO_INCREMENT PRIMARY KEY,

```

```

7   timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP ,
8   temperature FLOAT NOT NULL ,
9   humidity FLOAT NOT NULL
10 );

```

2.2.2 Datenbank-Schema

Das Datenbank-Schema besteht aus einer Tabelle `sensor_data`, die die folgenden Spalten enthält:

- **id**: Eine eindeutige ID für jeden Eintrag.
- **timestamp**: Der Zeitstempel, der die Uhrzeit der Messung angibt.
- **temperature**: Die gemessene Temperatur in Grad Celsius.
- **humidity**: Die gemessene relative Luftfeuchtigkeit in Prozent.

2.2.3 Datenverwaltung

Die Sensordaten wurden über Python-Skripte direkt in die Datenbank geschrieben. Der folgende Python-Code zeigt den Mechanismus zum Speichern der Daten in der Datenbank:

Listing 2.8: Python-Code für die Speicherung der Sensordaten in MariaDB

```

1 import pymysql
2
3 def save_to_database(temp, hum):
4     conn = pymysql.connect(
5         host="localhost",
6         user="admin1",
7         password="admin1",
8         database="sensor_data"
9     )
10    cursor = conn.cursor()
11    cursor.execute(
12        "INSERT INTO sensor_data (temperature, humidity) VALUES (%s, %s"
13        ),
14        (temp, hum)
15    )
16    conn.commit()
17    conn.close()

```

Dieser Code verbindet sich mit der Datenbank `sensor_data`, führt einen SQL-Befehl zum Einfügen der Daten aus und schließt die Verbindung.

2.2.4 Backup älterer Daten

Ein Mechanismus zur regelmäßigen Sicherung von älteren Daten wurde implementiert, um Speicherplatz auf dem Raspberry Pi freizugeben und die Daten langfristig zu sichern. Daten, die älter als 24 Stunden sind, werden in einer CSV-Datei auf einer externen Festplatte gespeichert. Der folgende Python-Code zeigt die Implementierung des Backups:

Listing 2.9: Backup älterer Daten auf externe Festplatte

```
1 import csv
2 import pymysql
3
4 def save_old_data_to_external_drive():
5     conn = pymysql.connect(
6         host="localhost",
7         user="admin1",
8         password="admin1",
9         database="sensor_data"
10    )
11    cursor = conn.cursor()
12    cursor.execute("SELECT * FROM sensor_data WHERE timestamp < NOW() - "
13                   "INTERVAL 1 DAY")
14    rows = cursor.fetchall()
15
16    if rows:
17        backup_file = "/media/pi/MyDrive/backup.csv"
18        with open(backup_file, "a", newline="") as f:
19            writer = csv.writer(f)
20            writer.writerow(["ID", "Timestamp", "Temperature", "Humidity"])
21            writer.writerows(rows)
22
23        cursor.execute("DELETE FROM sensor_data WHERE timestamp < NOW() - "
24                       "INTERVAL 1 DAY")
25        conn.commit()
26        conn.close()
```

2.2.5 Zusammenfassung

Die Datenbank bildet die zentrale Komponente für die Speicherung und Verwaltung der Sensordaten. Durch die Verwendung von MariaDB konnten die Daten effizient verarbeitet und in einer strukturierten Form gespeichert werden. Der implementierte Backup-Mechanismus gewährleistet eine langfristige Sicherung der Daten und optimiert gleichzeitig die Speicherplatznutzung auf dem Raspberry Pi.

Kapitel 3

Implementierung

3.1 Python-Skript für Datenverarbeitung

Das Python-Skript bildet das Herzstück des Systems und ermöglicht die Kommunikation zwischen der Hardware (Arduino Nano und Sensor) und der Software (Datenbank und Webanwendung). Es übernimmt die Verarbeitung der Sensordaten, deren Speicherung und die Visualisierung in einer Webanwendung. Dieses Kapitel beschreibt die Funktionsweise des Skripts detailliert.

Grundlegende Architektur des Skripts Das Skript besteht aus mehreren Modulen, die spezifische Aufgaben übernehmen:

- **Datenempfang:** Daten werden über die serielle Schnittstelle vom Arduino Nano empfangen.
- **Datenverarbeitung:** Empfangenes Rohdatenformat wird geprüft, verarbeitet und für die Speicherung vorbereitet.
- **Speicherung:** Validierte Daten werden in einer MariaDB-Datenbank gespeichert.
- **Webserver:** Die Flask-Webanwendung stellt eine Benutzeroberfläche bereit, auf der die Daten in Echtzeit visualisiert werden.
- **Backup-System:** Ältere Daten werden regelmäßig auf eine externe Festplatte gesichert, um den Speicherbedarf zu optimieren.

Datenempfang und Verarbeitung Die Kommunikation zwischen dem Arduino Nano und dem Raspberry Pi erfolgt über die serielle Schnittstelle. Der Arduino sendet die Temperatur- und Feuchtigkeitswerte im JSON-Format. Das Skript empfängt diese Daten, prüft deren Format und extrahiert die benötigten Informationen. Der folgende Code zeigt die Implementierung des Datenempfangs:

Listing 3.1: Datenempfang über die serielle Schnittstelle

```
1 import serial
2 import json
3
4 SERIAL_PORT = '/dev/ttyACM0'
```

```

5 BAUD_RATE = 9600
6 ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
7
8 def process_serial_data():
9     while True:
10         if ser.in_waiting > 0:
11             line = ser.readline().decode('utf-8').strip()
12             print(f"Rohdaten vom Arduino: {line}")
13             try:
14                 data = json.loads(line)
15                 temperature = data["temperature"]
16                 humidity = data["humidity"]
17                 print(f"Verarbeitete Daten: Temperatur={temperature},"
18                       "Feuchtigkeit={humidity}")
19                 return temperature, humidity
20             except (KeyError, ValueError, TypeError):
21                 print("Ungültiges Datenformat empfangen.")
22             return None, None

```

Die Methode `process_serial_data` sorgt dafür, dass nur gültige Daten verarbeitet werden. Fehlerhafte Daten werden erkannt und ignoriert.

Speicherung der Daten in der Datenbank Nach der Verarbeitung werden die Daten in der Datenbank gespeichert. Der folgende Code zeigt die Implementierung des Speichervorgangs:

Listing 3.2: Speicherung der Daten in MariaDB

```

1 import pymysql
2
3 def save_to_database(temp, hum):
4     conn = pymysql.connect(
5         host="localhost",
6         user="admin1",
7         password="admin1",
8         database="sensor_data"
9     )
10    cursor = conn.cursor()
11    cursor.execute(
12        "INSERT INTO sensor_data (temperature, humidity) VALUES (%s, %s"
13        ),
14        (temp, hum)
15    )
16    conn.commit()
17    conn.close()

```

Dieser Code öffnet eine Verbindung zur Datenbank `sensor_data`, fügt die neuen Daten ein und schließt anschließend die Verbindung.

Webserver und Benutzeroberfläche Das Skript nutzt das Flask-Framework, um eine Webanwendung bereitzustellen, die die Daten in Echtzeit visualisiert. Es wurden mehrere Endpunkte implementiert, darunter:

- `/dashboard`: Zeigt eine grafische Darstellung der Temperatur- und Feuchtigkeitswerte.
- `/data`: Liefert die letzten 50 gespeicherten Datenpunkte im JSON-Format.

- /view-data: Bietet eine Tabelle mit historischen Daten an.

Beispiel für den Endpunkt /data:

Listing 3.3: Bereitstellung von Daten über den /data-Endpunkt

```

1 from flask import Flask, jsonify
2
3 app = Flask(__name__)
4
5 @app.route('/data', methods=['GET'])
6 def get_data():
7     global data
8     if not data:
9         return jsonify({"error": "Keine Daten verfügbar"}), 404
10    return jsonify(data[-50:])

```

Die Daten werden in der Benutzeroberfläche mit dem JavaScript-Framework Chart.js visualisiert. Der folgende HTML-Code zeigt die Integration der Charts:

Listing 3.4: HTML für die Visualisierung der Daten

```

1 <canvas id="tempChart" width="400" height="200"></canvas>
2 <canvas id="humChart" width="400" height="200"></canvas>
3 <script>
4     const tempChart = new Chart(document.getElementById('tempChart')).getContext('2d'), {
5         type: 'line',
6         data: { /* Datenstruktur */ },
7         options: { /* Konfigurationsoptionen */ }
8     );
9 </script>

```

Backup-System Ein automatisiertes Backup-System wurde implementiert, um ältere Daten auf eine externe Festplatte zu speichern. Der folgende Code zeigt den Mechanismus:

Listing 3.5: Backup älterer Daten auf externe Festplatte

```

1 import csv
2
3 def save_old_data_to_external_drive():
4     conn = pymysql.connect(
5         host="localhost",
6         user="admin1",
7         password="admin1",
8         database="sensor_data"
9     )
10    cursor = conn.cursor()
11    cursor.execute("SELECT * FROM sensor_data WHERE timestamp < NOW() - "
12                  "INTERVAL 1 DAY")
13    rows = cursor.fetchall()
14
15    if rows:
16        backup_file = "/media/pi/MyDrive/backup.csv"
17        with open(backup_file, "a", newline="") as f:
18            writer = csv.writer(f)
19            writer.writerow(["ID", "Timestamp", "Temperature", "Humidity"])
20            writer.writerows(rows)

```

```

20
21     cursor.execute("DELETE FROM sensor_data WHERE timestamp < NOW()
22         - INTERVAL 1 DAY")
23     conn.commit()
24     conn.close()

```

Zusammenfassung des Serverbetriebs Das Python-Skript arbeitet kontinuierlich, um die Sensordaten in Echtzeit zu verarbeiten und bereitzustellen:

1. Es empfängt Daten über die serielle Schnittstelle und verarbeitet diese.
2. Die Daten werden in der Datenbank gespeichert und über Flask-Endpunkte bereitgestellt.
3. Die Daten können über die Benutzeroberfläche in Form von Graphen visualisiert werden.
4. Ältere Daten werden automatisch gesichert, um Speicherplatz freizugeben.

Durch diese Architektur konnte ein robustes und skalierbares System für die Datenerfassung und -verarbeitung geschaffen werden, das die Anforderungen des Projekts erfüllt.

3.1.1 Webschnittstelle

Die Webschnittstelle wurde entwickelt, um die Sensordaten benutzerfreundlich darzustellen. Sie ermöglicht es, die Temperatur- und Feuchtigkeitswerte in Echtzeit zu visualisieren, historische Daten einzusehen und Benutzeraktionen wie Login und Logout durchzuführen. Die Webschnittstelle basiert auf dem Flask-Framework und besteht aus mehreren HTML-Seiten, die über Flask-Endpunkte bereitgestellt werden.

Hauptfunktionen der Webschnittstelle Die Webschnittstelle bietet folgende Hauptfunktionen:

- **Dashboard:** Echtzeitvisualisierung der Temperatur- und Feuchtigkeitsdaten.
- **Datenanzeige:** Tabellenansicht der gespeicherten historischen Daten.
- **Benutzeroberflächen:** Login- und Logout-Funktionalitäten, um den Zugriff auf geschützte Seiten zu kontrollieren.

Implementierung der HTML-Seiten Die Webschnittstelle besteht aus drei Hauptseiten: `index.html`, `login.html` und `view_data.html`. Nachfolgend werden diese Seiten und ihre Funktionalitäten im Detail beschrieben.

index.html - Dashboard

Die Datei `index.html` stellt die zentrale Seite der Webschnittstelle dar und visualisiert die Daten in Form von Graphen. Hier werden die Temperatur- und Feuchtigkeitsdaten mithilfe der JavaScript-Bibliothek `Chart.js` dargestellt. Der Code sieht wie folgt aus:

Listing 3.6: HTML-Code für das Dashboard (index.html)

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale
6         =1.0">
7     <title>Temperature and Humidity Dashboard</title>
8     <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
9     <style>
10        body {
11            background-color: black;
12            color: white;
13            font-family: Arial, sans-serif;
14        }
15        .buttons {
16            position: absolute;
17            top: 10px;
18            left: 20px;
19        }
20        #info {
21            position: absolute;
22            top: 10px;
23            right: 20px;
24        }
25        canvas {
26            display: block;
27            margin: 0 auto;
28            max-width: 90%;
29        }
30     </style>
31 </head>
32 <body>
33     <div class="buttons">
34         <button onclick="logout()">Logout</button>
35         <button onclick="viewData()">View Data</button>
36     </div>
37     <div id="info">
38         <p>Temperature: <span id="tempInfo">--</span> C</p>
39         <p>Humidity: <span id="humInfo">--</span> %</p>
40     </div>
41     <canvas id="tempChart" width="400" height="200"></canvas>
42     <canvas id="humChart" width="400" height="200"></canvas>
43     <script>
44         const tempCtx = document.getElementById('tempChart').getContext
45             ('2d');
46         const humCtx = document.getElementById('humChart').getContext
47             ('2d');
48         const tempChart = new Chart(tempCtx, { /* Konfiguration fuer
49             Temperatur */ });
50         const humChart = new Chart(humCtx, { /* Konfiguration fuer
51             Feuchtigkeit */ });
52         async function fetchData() {
53             const response = await fetch('/data');
54             const json = await response.json();
55             // Daten verarbeiten und anzeigen
56         }
57         setInterval(fetchData, 20000);
```

```

53     function logout() {
54         window.location.href = "/logout";
55     }
56     function viewData() {
57         window.location.href = "/view-data";
58     }
59 </script>
60 </body>
61 </html>

```

login.html - Login-Seite

Listing 3.7: HTML-Code für die Login-Seite (login.html)

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale
6          =1.0">
7      <title>Login</title>
8      <style>
9          body {
10              font-family: Arial, sans-serif;
11              background-color: black;
12              color: white;
13              display: flex;
14              justify-content: center;
15              align-items: center;
16              height: 100vh;
17          }
18          form {
19              background: #333;
20              padding: 20px;
21              border-radius: 10px;
22              box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
23          }
24          input {
25              margin-bottom: 10px;
26              padding: 10px;
27              width: 100%;
28          }
29      </style>
30  </head>
31  <body>
32      <form method="POST" action="/login">
33          <h1>Login</h1>
34          <label>Username:</label>
35          <input type="text" name="username" required>
36          <label>Password:</label>
37          <input type="password" name="password" required>
38          <button type="submit">Login</button>
39      </form>
40  </body>
</html>

```

view_data.html - Datenanzeige

Die Datei `view_data.html` zeigt die gespeicherten historischen Daten in Tabellenform an. Der Code sieht wie folgt aus:

Listing 3.8: HTML-Code für die Datenanzeige (`view_data.html`)

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale
6         =1.0">
7     <title>View Data</title>
8     <style>
9         body {
10             font-family: Arial, sans-serif;
11             background-color: black;
12             color: white;
13         }
14         table {
15             width: 100%;
16             border-collapse: collapse;
17         }
18         th, td {
19             padding: 10px;
20             text-align: left;
21             border-bottom: 1px solid #ddd;
22         }
23     </style>
24 </head>
25 <body>
26     <h1>View Data</h1>
27     <table>
28         <tr>
29             <th>ID</th>
30             <th>Timestamp</th>
31             <th>Temperature (C)</th>
32             <th>Humidity (%)</th>
33         </tr>
34         {% for row in rows %}
35         <tr>
36             <td>{{ row[0] }}</td>
37             <td>{{ row[1] }}</td>
38             <td>{{ row[2] }}</td>
39             <td>{{ row[3] }}</td>
40         </tr>
41         {% endfor %}
42     </table>
43 </body>
</html>
```

Zusammenfassung Die Webschnittstelle ermöglicht eine effiziente und benutzerfreundliche Visualisierung der Sensordaten. Die Kombination aus HTML, JavaScript und Flask-Framework bildet die Grundlage für die Interaktivität und Skalierbarkeit des Systems.

Kapitel 4

Ergebnisse

Das entwickelte System ermöglicht eine Echtzeitüberwachung von Temperatur- und Feuchtigkeitsdaten mithilfe einer Webschnittstelle. Dieser Abschnitt beschreibt, wie der Server arbeitet, welche Schritte durchlaufen werden und wie die Daten letztendlich verarbeitet und visualisiert werden.

4.1 Serverbetrieb und Funktionsweise

Aktivieren der virtuellen Umgebung Vor dem Start des Servers muss die virtuelle Python-Umgebung mit dem folgenden Befehl aktiviert werden:

```
1 source venv/bin/activate
```

Die Aktivierung der virtuellen Umgebung ist notwendig, da alle Abhängigkeiten des Projekts, wie die Python-Bibliotheken `Flask`, `pymysql`, und `bcrypt`, in dieser Umgebung isoliert installiert sind. Dies hat folgende Vorteile:

- **Unabhängigkeit:** Die virtuelle Umgebung trennt die Projektabhängigkeiten von den globalen Python-Bibliotheken des Systems, um Versionskonflikte zu vermeiden.
- **Konsistenz:** Alle Benutzer des Projekts können sicherstellen, dass sie mit den exakt gleichen Bibliotheksversionen arbeiten.
- **Sicherheit:** Änderungen an Abhängigkeiten beeinflussen nicht das gesamte System, sondern nur das spezifische Projekt.

Nach der Aktivierung wird die Kommandozeilenanzeige um den Namen der virtuellen Umgebung ergänzt, beispielsweise:

```
1 (venv) pi@raspberrypi:~/myproject $
```

Dies zeigt an, dass die virtuelle Umgebung erfolgreich aktiviert wurde. Anschließend kann der Server gestartet werden:

```
1 python3 server.py
```

Nach dem Start des Servers durch den Befehl:

```
1 pi@raspberrypi:~/myproject $ source venv/bin/activate
2 (venv) pi@raspberrypi:~/myproject $ python3 server.py
```

beginnt das Python-Skript, die Kommunikation mit der Hardware (Arduino Nano und Sensoren) aufzunehmen und eine Webschnittstelle bereitzustellen.

1. Datenempfang und Verarbeitung Der Server empfängt die Sensordaten über die serielle Schnittstelle, die mit dem Arduino Nano verbunden ist. Diese Daten werden im JSON-Format gesendet und enthalten Temperatur- und Feuchtigkeitswerte. Der Server überprüft die Integrität der empfangenen Daten und verarbeitet diese für die Speicherung und Visualisierung.

Während des Betriebs protokolliert der Server folgende Daten im Terminal:

Listing 4.1: Beispiel für Server-Logs während des Betriebs

```

1 Raw data from Arduino: {"temperature":13.42, "humidity":59.22}
2 Parsed data: Temp=13.42, Hum=59.22
3 Raw data from Arduino: {"temperature":13.44, "humidity":59.20}
4 Parsed data: Temp=13.44, Hum=59.2
5 192.168.67.240 - - [21/Dec/2024 17:10:33] "GET /dashboard HTTP/1.1" 200
-
6 192.168.67.240 - - [21/Dec/2024 17:10:33] "GET /data HTTP/1.1" 200 -
7 Raw data from Arduino: {"temperature":13.38, "humidity":59.21}
8 Parsed data: Temp=13.38, Hum=59.21
9 192.168.67.240 - - [21/Dec/2024 17:10:34] "GET /data HTTP/1.1" 200 -

```

2. Speicherung der Daten Die validierten Daten werden in einer MariaDB-Datenbank gespeichert. Jede Datenzeile enthält:

- einen eindeutigen *ID*-Wert,
- einen Zeitstempel (*timestamp*),
- die gemessene Temperatur (*temperature*),
- die gemessene Luftfeuchtigkeit (*humidity*).

Ein Beispiel für gespeicherte Daten zeigt Tabelle 4.1.

ID	Zeitstempel	Temperatur (°C)	Luftfeuchtigkeit (%)
1562	2024-12-21 17:10:17	13.39	59.18
1563	2024-12-21 17:10:18	13.40	59.19
1564	2024-12-21 17:10:19	13.42	59.20

Tabelle 4.1: Beispiel für gespeicherte Sensordaten in der Datenbank

3. Bereitstellung von Daten Der Server stellt zwei Hauptendpunkte bereit:

- */data*: Dieser Endpunkt liefert die letzten 50 Datenpunkte im JSON-Format, die von der Webschnittstelle abgerufen werden.
- */view-data*: Hier können Benutzer historische Daten in Tabellenform einsehen.

4. Datenvisualisierung Die Temperatur- und Feuchtigkeitsdaten werden in Echtzeit auf der Webschnittstelle visualisiert. Mithilfe der *Chart.js*-Bibliothek werden zwei separate Graphen für Temperatur und Feuchtigkeit erstellt. Die Daten werden alle 20 Sekunden aktualisiert und die Graphen kontinuierlich angepasst.

5. Benutzeroberfläche Die Webschnittstelle ist durch ein Login-System geschützt. Nach erfolgreicher Anmeldung können Benutzer auf das Dashboard und die Datenansicht zugreifen. Die Sicherheitsmechanismen stellen sicher, dass unbefugte Zugriffe verhindert werden.

6. Datensicherung Ein automatisches Backup-System speichert Daten, die älter als 24 Stunden sind, im CSV-Format auf eine angeschlossene externe Festplatte. Dies gewährleistet eine effiziente Speicherverwaltung auf dem Raspberry Pi.

4.2 Visualisierung der Ergebnisse

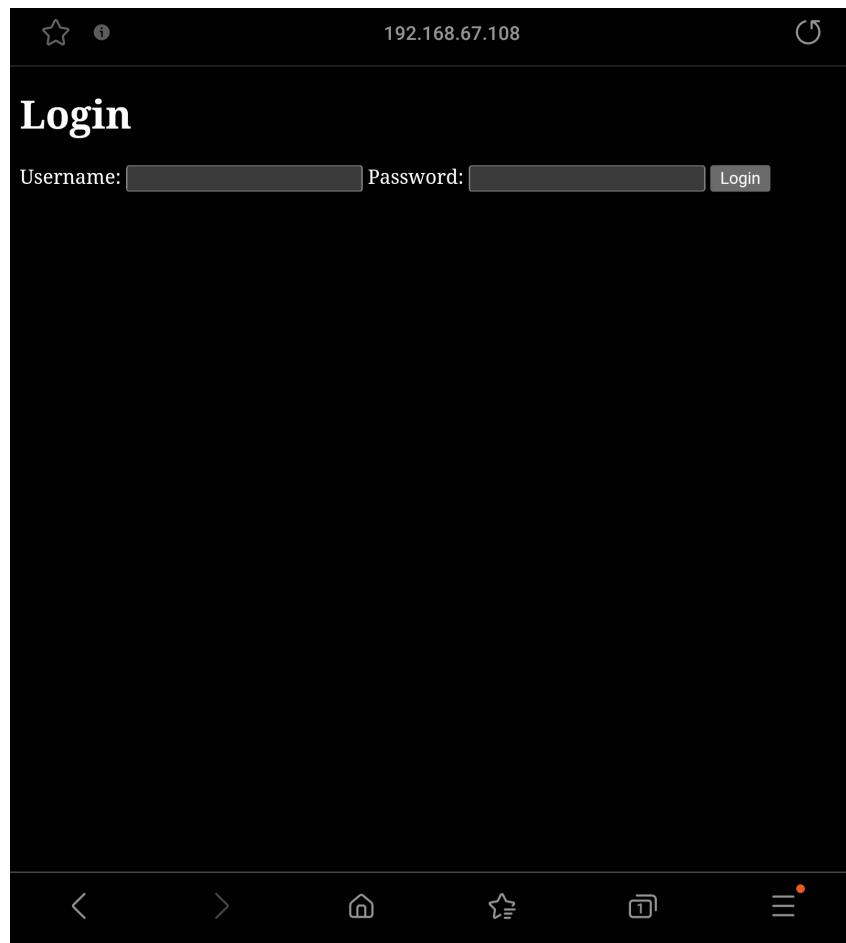


Abbildung 4.1: Login-Seite der Webschnittstelle

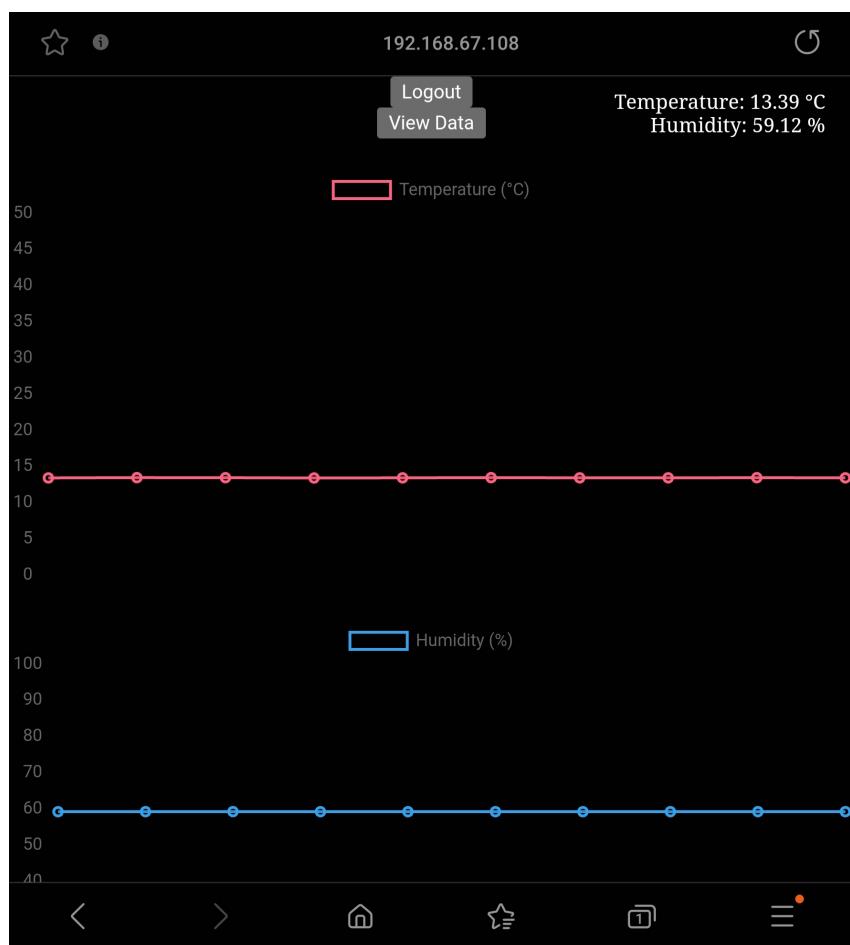


Abbildung 4.2: Dashboard mit Echtzeitvisualisierung von Temperatur und Feuchtigkeit

ID	Timestamp	Temperature (°C)	Humidity (%)
1562	2024-12-21 17:10:17	13.39	59.18
1563	2024-12-21 17:10:18	13.4	59.19
1564	2024-12-21 17:10:19	13.42	59.2
1565	2024-12-21 17:10:20	13.37	59.19
1566	2024-12-21 17:10:21	13.39	59.19
1567	2024-12-21 17:10:22	13.43	59.21
1568	2024-12-21 17:10:23	13.42	59.19
1569	2024-12-21 17:10:24	13.41	59.19
1570	2024-12-21 17:10:25	13.4	59.2
1571	2024-12-21 17:10:26	13.42	59.19

Abbildung 4.3: Ansicht der gespeicherten Sensordaten

Kapitel 5

Diskussion und Ausblick

5.1 Erreichte Ziele

Im Verlauf dieses Projekts wurden mehrere wichtige Meilensteine erreicht, die ein robustes und funktionsfähiges System zur Überwachung von Temperatur und Feuchtigkeit in einem Laborumfeld ermöglicht haben. Nachfolgend werden die erreichten Ziele detailliert beschrieben.

1. Einrichtung der Hardware

- **Arduino und Sensorintegration:** Der SHTC3-Temperatur- und Feuchtigkeitssensor wurde erfolgreich über den EasyC-Adapter mit dem Arduino Nano verbunden. Die Verbindungen für I2C (SDA, SCL), Stromversorgung und Masse wurden korrekt hergestellt.
- **Raspberry Pi Integration:** Der Raspberry Pi wurde mit einem Arduino Nano über eine USB-Schnittstelle verbunden, um Daten direkt zu empfangen und zu verarbeiten.
- **Externe Speicherintegration:** Eine externe Festplatte wurde am Raspberry Pi eingerichtet, um historische Daten im CSV-Format zu sichern. Dies ermöglicht eine effiziente Datenverwaltung und langfristige Speicherung.

2. Softwareentwicklung

- **Datenverarbeitung mit Python:** Ein Python-Skript wurde entwickelt, um die Daten vom Arduino Nano zu empfangen, zu verarbeiten und in einer lokalen MariaDB-Datenbank zu speichern. Dies umfasst die Implementierung der Funktionen zur Sicherung alter Daten und zur kontinuierlichen Datenaufnahme.
- **Webschnittstelle:** Eine intuitive und benutzerfreundliche Webschnittstelle wurde mithilfe von Flask und HTML entwickelt. Diese Schnittstelle umfasst:
 - Ein Dashboard zur Echtzeitvisualisierung der Temperatur- und Feuchtigkeitswerte.
 - Ein Login-System, das nur autorisierten Benutzern Zugriff auf das Dashboard und die Datenansicht gewährt.

- Eine Funktion zur Anzeige gespeicherter Daten in Tabellenform.
- **Datensicherung:** Daten, die älter als 24 Stunden sind, werden automatisch auf der externen Festplatte im CSV-Format gesichert. Diese Funktion minimiert den Speicherverbrauch des Raspberry Pi und erleichtert die Analyse der Daten.

3. Systemstabilität und Funktionalität

- **Echtzeitbetrieb:** Die Daten werden kontinuierlich vom Arduino empfangen und im Webinterface in Echtzeit visualisiert. Dies ermöglicht eine direkte Überwachung der Laborbedingungen.
- **Zuverlässigkeit:** Das System arbeitet stabil, wobei keine Verbindungsabbrüche zwischen Arduino und Raspberry Pi festgestellt wurden.
- **Skalierbarkeit:** Die bestehende Architektur erlaubt es, zusätzliche Sensoren hinzuzufügen und weitere Funktionen wie Klimasteuerung oder Cloud-Integration zu implementieren.

4. Benutzerauthentifizierung und Datensicherheit

- **Login-Mechanismus:** Ein sicherer Login-Mechanismus wurde implementiert, um unbefugten Zugriff zu verhindern. Benutzer müssen sich authentifizieren, bevor sie auf das Dashboard oder die gespeicherten Daten zugreifen können.
- **Sichere Speicherung:** Die gespeicherten Daten in der MariaDB-Datenbank sind vor unerlaubtem Zugriff geschützt.

5. Visualisierung und Benutzererfahrung

- **Echtzeitvisualisierung:** Temperatur- und Feuchtigkeitswerte werden in separaten Graphen dargestellt, die sich kontinuierlich aktualisieren. Diese Funktion wurde mit der Chart.js-Bibliothek implementiert.
- **Responsives Design:** Die Webschnittstelle ist so gestaltet, dass sie auf verschiedenen Geräten wie Desktops, Tablets und Smartphones gleichermaßen gut funktioniert.

Zusammenfassung Das entwickelte System erfüllt die grundlegenden Anforderungen des Projekts:

1. **Überwachung:** Es ermöglicht die Echtzeitüberwachung der Temperatur- und Feuchtigkeitswerte im Laborumfeld.
2. **Datenmanagement:** Historische Daten werden effizient gespeichert und gesichert.
3. **Erweiterbarkeit:** Die bestehende Infrastruktur ist flexibel und kann leicht um neue Funktionen erweitert werden.

Die erreichten Ziele bilden eine solide Grundlage für zukünftige Entwicklungen, wie z.B. die Integration von Klimasteuerungen oder die Überwachung über ein Cloud-System. Das System ist sowohl robust als auch benutzerfreundlich, was es ideal für den Einsatz in experimentellen Umgebungen macht.

5.2 Zukünftige Arbeiten

Obwohl das entwickelte System bereits eine zuverlässige Überwachung der Temperatur- und Feuchtigkeitswerte ermöglicht, gibt es noch mehrere Bereiche, die weiterentwickelt werden können, um die Funktionalität und Benutzerfreundlichkeit zu verbessern. Nachfolgend werden die wichtigsten zukünftigen Arbeiten beschrieben.

1. Integration eines Klimasteuerungssystems Ein entscheidender nächster Schritt ist die Integration eines Klimasteuerungssystems, das auf Basis der erfassten Temperatur- und Feuchtigkeitsdaten arbeitet. Dazu gehört:

- Die Verbindung des Raspberry Pi mit einem Klimagerät oder einem ähnlichen Steuerungssystem über geeignete Schnittstellen wie GPIO oder Relais.
- Die Entwicklung eines Algorithmus, der Temperatur- und Feuchtigkeitsschwellen definiert und das Klimagerät automatisch steuert, um die Bedingungen im Labor optimal zu halten.
- Die Erweiterung der Webschnittstelle, um Benutzern die manuelle Steuerung und Überwachung der Klimasteuerung zu ermöglichen.

2. Zugriff über das Internet Derzeit ist die Webschnittstelle nur über ein lokales Netzwerk zugänglich. Um die Reichweite und Benutzerfreundlichkeit zu erweitern, sollte eine Internetverbindung integriert werden:

- Einrichtung eines sicheren Zugangs über das Internet, z.B. durch Nutzung eines VPN oder Port-Forwarding.
- Implementierung zusätzlicher Sicherheitsmaßnahmen wie HTTPS und Zwei-Faktor-Authentifizierung, um den Zugriff zu schützen.
- Anpassung der Webschnittstelle, um eine bessere Unterstützung für verschiedene Geräte und Netzwerke zu gewährleisten.

3. Erweiterung der Datenspeicherung und Analyse Um die Analyse von langfristigen Trends zu ermöglichen, sollten weitere Funktionen zur Datenverarbeitung und -speicherung implementiert werden:

- Integration einer Cloud-Speicherlösung, um historische Daten zu sichern und von überall darauf zugreifen zu können.
- Implementierung von Analysetools zur Visualisierung von Langzeittrends und zur Unterstützung bei der Entscheidungsfindung.
- Entwicklung eines Exporttools, mit dem Benutzer Daten in verschiedenen Formaten (z.B. CSV, Excel) herunterladen können.

4. Verbesserung der Hardwareintegration Die derzeitige Hardwarekonfiguration kann weiter optimiert werden, um eine bessere Effizienz und Zuverlässigkeit zu gewährleisten:

- Untersuchung der Möglichkeit, mehrere Sensoren gleichzeitig zu integrieren und zu überwachen, um verschiedene Bereiche des Labors abzudecken.
- Test und Implementierung von alternativen Verbindungsmethoden wie drahtlosen Protokollen (z.B. ZigBee oder Bluetooth), um die Flexibilität des Systems zu erhöhen.
- Optimierung der Stromversorgung, um den Raspberry Pi und andere Komponenten über eine unterbrechungsfreie Stromversorgung (USV) zu betreiben.

5. Benutzerfeedback und Systemoptimierung Die Benutzerfreundlichkeit und Effizienz des Systems sollten durch regelmäßiges Feedback verbessert werden:

- Durchführung von Benutzerumfragen, um Schwachstellen und Verbesserungspotenziale der Webschnittstelle und des Systems zu identifizieren.
- Iterative Updates basierend auf Benutzerfeedback, um die Anforderungen des Labors besser zu erfüllen.
- Erweiterung der Dokumentation, um die Einrichtung und den Betrieb des Systems für zukünftige Nutzer einfacher zu gestalten.

Zusammenfassung Die genannten zukünftigen Arbeiten zielen darauf ab, das bestehende System zu einem umfassenden und flexiblen Werkzeug für die Überwachung und Steuerung der Laborumgebung weiterzuentwickeln. Die Integration von Klimasteuerung, Internetzugang und erweiterten Analysetools sowie die Verbesserung der Hardware stellen wichtige Schritte dar, um den Nutzen und die Effizienz des Systems weiter zu steigern.