# AirMetrics

## Praktische Ausarbeitung im Fach Internet Of Things

des Studienganges Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von


| | |
|---|---|
| Ahmetovic, Samir | 6948836 |
| Bai, Julia | 2090617 |
| Esenwein Benjamin | 5655208 |
| Kaczynski, Lucas | 9411266 |
| König, Marcel | 7203551 |
| Ernst, Lukas | 2094490 |


| | |
|---|---|
| Bearbeitungszeitraum | 10.03.2023 bis 27.04.2023 |
| Abgabe | 27.04.2023 |

# Inhaltsverzeichnis

# Abbildungsverzeichnis

## Eidesstattliche Erklärung

Hiermit bestätigen alle auf dem Deckblatt angegebenen Autoren, dass sie die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet haben.

Stuttgart, 24.03.23

S. Ahmetovic

Samir Ahmetovic

J. Bai

Julia Bai

B. Esenwein

Benjamin Esenwein

L. Kaczynski

Lucas Kaczynski

M. König

Marcel König

Lukas Ernst

Lukas Ernst

# 1 Allgemeine Beschreibung

In diesem Kapitel wird die IoT-Lab-Aufgabenstellung vom Dozenten beschrieben sowie die definierten Randbedingungen und den Aufbau der Arbeit vorgestellt.

## 1.1 Aufgabenstellung

Im IoT-Lab soll eine IoT-Implementierung während der Vorlesung designet, implementiert, dokumentiert und in einer kurzen Präsentation vorgestellt werden. Die Abgabe der Ausarbeitung findet am 27.04.2023 bis 20 Uhr statt. Das Referat wird am 28.04.2023 gehalten.

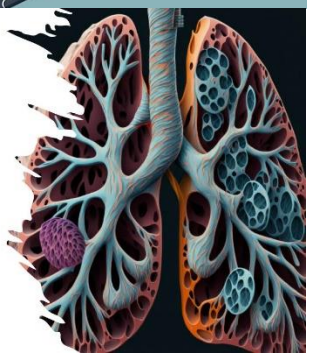
## 1.2 Aufbau der Arbeit

Nach der Vorstellung der allgemeinen Aufgabenbeschreibung des IoT-Labs folgt eine Auflistung der Präsentationsfolien. Anschließend wird der Anwendungsfall der IoT-Lösung beschrieben sowie näher auf die IoT-Lösung anhand des IoT Values Stacks eingegangen. Zum Abschluss wird in einer Schlussbetrachtung die Ausarbeitung kurz zusammengefasst.

# 2 Präsentationsfolien


## Anwendungsfall

- Überwachung und Analyse der Luftverschmutzung in Echtzeit
- Erfassung von Schadstoffen: Feinstaub, Stickoxide, Ozon und flüchtige organische Verbindungen
- Umfassendes Bild der aktuellen Luftqualität
- Frühzeitige Maßnahmen zur Minimierung gesundheitlicher Auswirkungen
- Erkennung von Trends und Mustern in der Luftverschmutzung, um gezielt Strategien zur Verbesserung der Luftqualität zu entwickeln



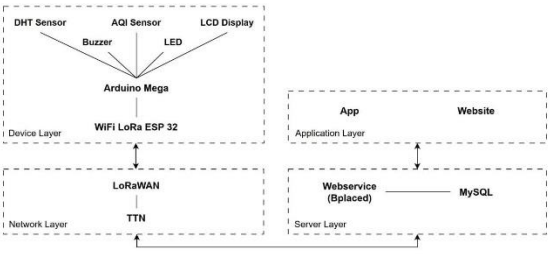
2

## IoT-Lösung

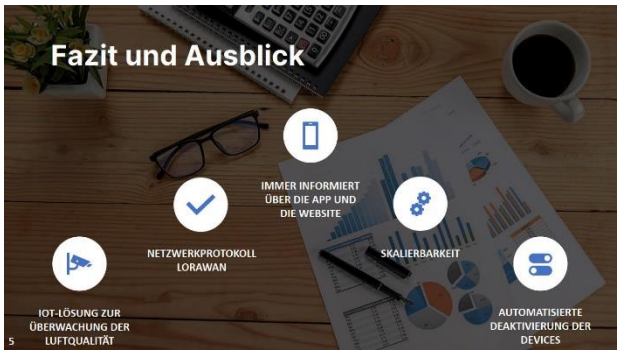
- Datenerfassung mithilfe von Sensoren
- Übertragung mit LoRaWAN
- Speicherung der Daten auf einem Server
- Der Endanwender soll am Ende über einen Buzzer, LEDs und einem LCD-Display die Sensordaten einsehen können
- Zusätzlicher Zugriff auf die Daten über eine Website und eine App

3

## Architekturdiagramm




4




5

# 3 AirMetrics

Im Folgenden werden die IoT-Lösung und ihre Umsetzung genauer vorgestellt.

In einer Zeit, in der Umweltverschmutzung und Klimawandel zunehmend globale Herausforderungen darstellen, ist es wichtiger denn je, über effektive und zuverlässige Systeme zur Überwachung der Luftqualität zu verfügen. Wir präsentieren eine IoT-Lösung, die speziell dafür entwickelt wurde, um die Luftqualität in Echtzeit präzise zu messen und zu überwachen. Diese Lösung nutzt Sensortechnologien, um in einem Raum die Temperatur, die Luftfeuchtigkeit sowie eine Vielzahl von Schadstoffen wie Feinstaub, Stickoxide, Ozon und flüchtige organische Verbindungen als Air Quality Index zu erfassen.

Das IoT-System ist darauf ausgelegt, eine kontinuierliche Datenerfassung zu ermöglichen, um ein detailliertes Verständnis der Umweltbedingungen zu gewährleisten. Durch die Integration der LoRaWAN-Kommunikationstechnologie ermöglicht das System eine effiziente und energiearme Datenübertragung über große Entfernungen, was den Einsatz in städtischen und ländlichen Gebieten erleichtert.

## 3.1 Anwendungsfall der IoT Lösung

Die IoT-Lösung zur Messung der Luftqualität kann verschiedene Geschäftsprobleme lösen. Sie nutzt den Air Quality Index (AQI), um die Luftverschmutzung und ihre Auswirkungen besser zu bewerten. Der Air Quality Index (AQI) ist ein standardisiertes Maß für die Konzentration von Schadstoffen in der Luft, welches ein besseres Verständnis der Luftqualität ermöglicht. Im Folgenden sind einige der geschäftlichen Probleme aufgeführt, die durch die IoT-Lösung gelöst werden können:

1. Kostenreduktion: Die IoT-Lösung ermöglicht eine automatisierte und effiziente Datenerfassung, wodurch die Kosten für manuelle Messungen und Wartung gesenkt werden können.
2. Gesundheitsrisiken minimieren: Durch die kontinuierliche Überwachung der Luftqualität und den Einsatz des AQI können Unternehmen und Organisationen

mögliche Gesundheitsrisiken für ihre Mitarbeiter und Kunden frühzeitig erkennen und entsprechende Maßnahmen ergreifen.

3. Umweltverträgliche Entscheidungen: Die IoT-Lösung hilft Unternehmen, datenbasierte Entscheidungen in Bezug auf Umweltschutz und Nachhaltigkeit zu treffen, z. B. bei der Auswahl von Standorten, Umweltmaßnahmen oder Verkehrsmanagement.
4. Rechtliche Compliance: Die IoT-Lösung ermöglicht es Unternehmen, die Einhaltung von Umweltvorschriften und -gesetzen sicherzustellen, indem sie die Luftqualität überwachen und potenzielle Probleme frühzeitig identifizieren.
5. Informierte Stadtplanung und Infrastrukturinvestitionen: Die IoT-Lösung kann dazu beitragen, den Bedarf an Investitionen in umweltfreundliche Infrastrukturen und Verkehrssysteme zu identifizieren, um die Lebensqualität und Nachhaltigkeit in städtischen Gebieten zu verbessern.

# 4 IoT Lösungsbeschreibung anhand des IoT Values Stacks

Der beschriebene IoT-Anwendungsfall folgt dem IoT Value Stack. Dieser besteht aus mehreren Schichten, die im Folgenden näher erläutert werden.
Die erste Schicht ist das Device Layer, auf diesem findet die Datenerfassung statt, um die Luftqualität einschätzen zu können. Dafür werden die Sensoren DHT11 und MQ135 eingesetzt. Als Mikrocontroller kommt ein Arduino zum Einsatz.
Die zweite Schicht ist das Network Layer, auf diesem erfolgt die Übertragung von Daten an die Cloud oder das Gateway. An dieser Stelle kommt LoRaWAN zum Einsatz, um eine energiesparende und effiziente Datenübertragung über große Entfernungen zu ermöglichen zu können.
Auf dem Server Layer werden die gesammelten Daten verarbeitet, analysiert und gespeichert. Dafür wird ein Webserver, gehostet bei bplaced, genutzt. Die Daten werden in einer MySQL Datenbank abgelegt.
Auf der vierten Schicht, dem Application Layer, werden die gewonnen Erkenntnisse dem Endanwender präsentiert. Dafür wird eine Website und eine App entwickelt.

## 4.1 Praktische Umsetzung einer Datenerfassung von Sensordaten mit Arduino

**DHT-Sensor**

Der Codeausschnitt in Abbildung 1 beinhaltet nur den für den DHT11 Sensor relevanten Code. Anfangs wird der Daten-Pin (Digital Pin 12) und der Sensortyp definiert und eine neue Instanz der Klasse DHT aus der DHT-Sensor Library von Adafruit erstellt. Im Setup wird der Sensor initialisiert und die Baudrate auf 115200 gesetzt, damit das WiFi LoRa ESP 32 V2 Board die Daten empfangen kann. In der Loop wird die Temperatur und die Feuchtigkeit des DHT11 Sensors ausgelesen. Nur bei Rückgabe von numerischen Werten erfolgt eine Ausgabe über den seriellen Monitor. Die ausgegebenen Werte werden über den TX-Pin des Arduino zum RX-Pin des WiFi LoRa ESP 32 V2 Boards übertragen.

```ino
#include <DHT.h>

// DHT11 definitions
#define DHTPIN 12 // digital data pin
#define DHTTYPE DHT11 // dht sensor type
// create instance of dht
DHT dht(DHTPIN,DHTTYPE);
// define and initialize values to default
float humidity = 0;
float temperature = 0;

void setup() {
  dht.begin();
  Serial.begin(115200);
}

void loop() {
  // read DHT sensor values
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

  // checks if the sensor has a number-like-value
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Fehler beim Auslesen der DHT11 Sensor Daten");
    return;
  }
  // prints to serial output & sends data to esp via TX / RX
  Serial.print(temperature);
  Serial.print(",");
  Serial.println(humidity);
}
```

Abbildung 1: DHT11-Sensor Code

**MQ135-Sensor**

Der Codeausschnitt in Abbildung 2 beinhaltet nur den für den MQ135 Sensor relevanten Code. Der Sensor wird über den Analog Pin 0 (A0) ausgelesen. Im Setup wird der Pin-Mode so gesetzt, dass Daten am Analog Pin A0 als Input gelesen werden. Im Loop-Teil des Programms wird der Sensor mithilfe Library-Funktion „analogRead‘ ausgelesen. Anschließend erfolgt die Datenvorverarbeitung, bei der ebenso die LEDs sowie der Buzzer angesprochen werden.

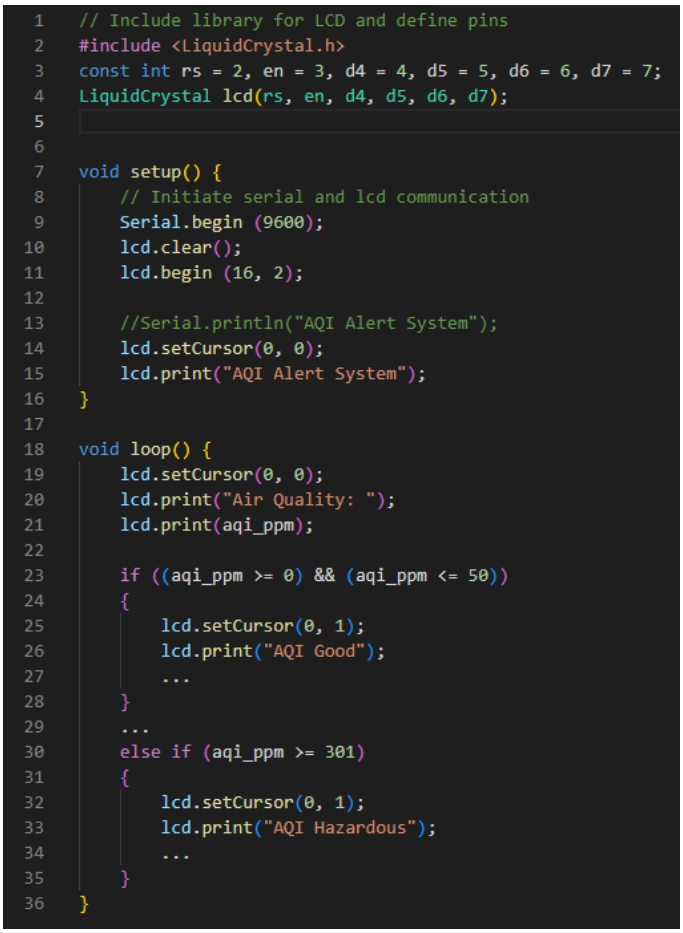
```
// Define pins and variable for input sensor and output led and buzzer
const int mq135_aqi_sensor = A0;
// Set threshold for AQI
int aqi_ppm = 0;

void setup() {
  // Set direction of input-output pins
  pinMode (mq135_aqi_sensor, INPUT);
}

void loop() {
    aqi_ppm = analogRead(mq135_aqi_sensor);

    // checks if the sensor has a number-like-value
    if (isnan(aqi_ppm)) {
        Serial.println("Fehler beim Auslesen der AQI Sensor Daten");
        return;
    }

    if ((aqi_ppm >= 0) && (aqi_ppm <= 50))
    {
        lcd.setCursor(0, 1);
        lcd.print("AQI Good");
    }
    else if ((aqi_ppm >= 51) && (aqi_ppm <= 100))
    {
        lcd.setCursor(0, 1);
        lcd.print("AQI Moderate");
    }
    else if ((aqi_ppm >= 101) && (aqi_ppm <= 200))
    {
        lcd.setCursor(0, 1);
        lcd.print("AQI Unhealthy");
    }
    else if ((aqi_ppm >= 201) && (aqi_ppm <= 300))
    {
        lcd.setCursor(0, 1);
        lcd.print("AQI V. Unhealthy");
    }
    else if (aqi_ppm >= 301)
    {
        lcd.setCursor(0, 1);
        lcd.print("AQI Hazardous");
    }
}
```

Abbildung 2: MQ135-Sensor Code

## 4.2 Steuerung von Geräten in Abhängigkeit der erfassten Daten

### LCD-Display

Der Codeausschnitt in Abbildung 3Abbildung 2 beinhaltet nur den für den LCD-Display relevanten Code. Das LCD-Display dient zur Anzeige der Sensorwerte nach der Vorverarbeitung der Daten. Vor dem Setup wird eine neue Instanz der Klasse LyquidCrystal aus der Arduino Library erstellt und die Pins Variablen zugewiesen. Im Setup wird die Displayausgabe gecleared und für die Verwendung vorbereitet. Der Cursor wird auf die Startposition (0, 0) verschoben. In der ersten Zeile des zweizeiligen LCD-Displays wird der aktuelle Air Quality Index ausgegeben. In der zweiten Zeile wird der aktuelle Zustand der Luftqualität (Schlecht, Medium, Gut) ausgegeben.



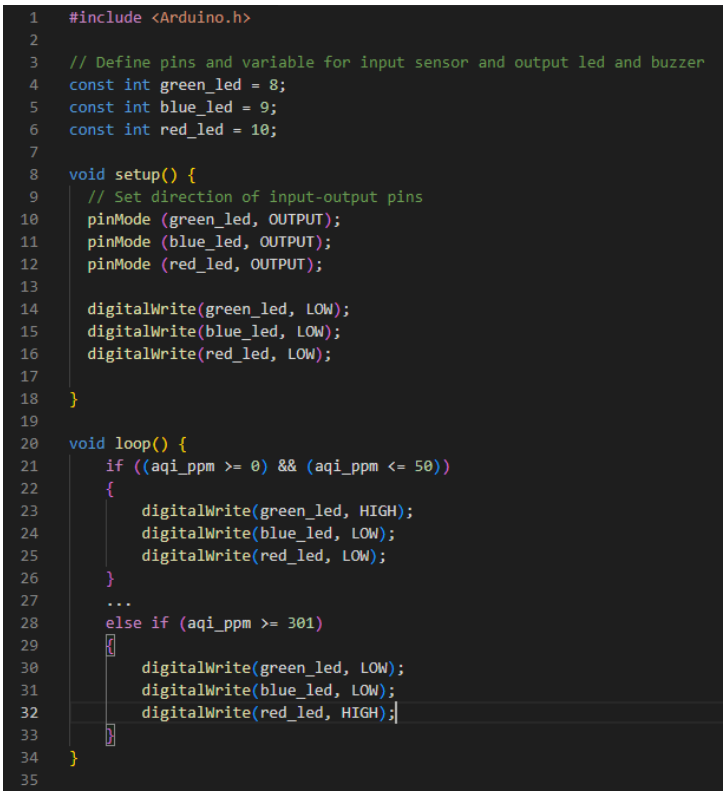
```
// Include library for LCD and define pins
#include <LiquidCrystal.h>
const int rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);


void setup() {
    // Initiate serial and lcd communication
    Serial.begin (9600);
    lcd.clear();
    lcd.begin (16, 2);

    //Serial.println("AQI Alert System");
    lcd.setCursor(0, 0);
    lcd.print("AQI Alert System");
}

void loop() {
    lcd.setCursor(0, 0);
    lcd.print("Air Quality: ");
    lcd.print(aqi_ppm);

    if ((aqi_ppm >= 0) && (aqi_ppm <= 50))
    {
        lcd.setCursor(0, 1);
        lcd.print("AQI Good");
        ...
    }
    ...
    else if (aqi_ppm >= 301)
    {
        lcd.setCursor(0, 1);
        lcd.print("AQI Hazardous");
        ...
    }
}
```

Abbildung 3: Arduino LCD-Display

**Ampel-System**

Da in diesem Aufbau nicht nur über das LCD-Display eine Ausgabe für den Nutzer erfolgen soll, wurde eine Ampelsystem implementiert, das in drei Abstufungen ebenfalls die Luftqualität anzeigt. Der Codeausschnitt in Abbildung 4 beinhaltet nur den für das LED-Ampelsystem relevanten Code.

Die LED-Pins werden als Output definiert und im Setup auf das LOW-Signal gesetzt. Je nach Luftqualität werden die einzelnen LEDs angesteuert und gegebenenfalls auf HIGH gesetzt. Die visuelle Anzeige in Form der LEDs sorgt dafür, dass der Nutzer eine vereinfachte Übersicht über die Luftqualität im Raum hat. Ein kurzer Blick auf die LEDs genügt, um zu wissen, ob der Raum gelüftet werden sollte.

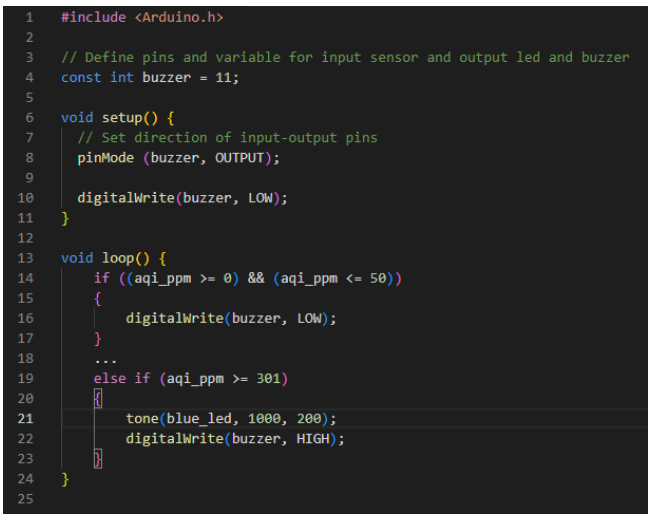


```
#include <Arduino.h>

// Define pins and variable for input sensor and output led and buzzer
const int green_led = 8;
const int blue_led = 9;
const int red_led = 10;

void setup() {
  // Set direction of input-output pins
  pinMode (green_led, OUTPUT);
  pinMode (blue_led, OUTPUT);
  pinMode (red_led, OUTPUT);

  digitalWrite(green_led, LOW);
  digitalWrite(blue_led, LOW);
  digitalWrite(red_led, LOW);

}

void loop() {
    if ((aqi_ppm >= 0) && (aqi_ppm <= 50))
    {
        digitalWrite(green_led, HIGH);
        digitalWrite(blue_led, LOW);
        digitalWrite(red_led, LOW);
    }
    ...
    else if (aqi_ppm >= 301)
    {
        digitalWrite(green_led, LOW);
        digitalWrite(blue_led, LOW);
        digitalWrite(red_led, HIGH);
    }
}
```

Abbildung 4: Quellcode der LED-Ampel-Anzeige

**Buzzer**

Der Quellcode in Abbildung 5 ergänzt das Programm für den Arduino um einen Buzzer, der den Nutzer auf sehr schlechte Luftqualität akustisch aufmerksam machen soll. Neben visuellen Mitteln ist ein akustisches Mittel sehr effektiv, um die Aufmerksamkeit des Nutzers zu bekommen.



```
#include <Arduino.h>

// Define pins and variable for input sensor and output led and buzzer
const int buzzer = 11;

void setup() {
  // Set direction of input-output pins
  pinMode (buzzer, OUTPUT);

  digitalWrite(buzzer, LOW);
}

void loop() {
    if ((aqi_ppm >= 0) && (aqi_ppm <= 50))
    {
        digitalWrite(buzzer, LOW);
    }
    ...
    else if (aqi_ppm >= 301)
    {
        tone(blue_led, 1000, 200);
        digitalWrite(buzzer, HIGH);
    }
}
```

Abbildung 5: Buzzer für sehr schlechte Luftqualität

Ähnlich wie bei den LEDs muss der Buzzer als Output definiert und sein Signal im Setup vorerst auf LOW gesetzt werden. Wenn der Buzzer aktiviert wird, dann wird der tone-Funktion aus der Arduino Library eine bestimmte Frequenz und Dauer mitgegeben, in der der Ton erzeugt werden soll. Dieser ist bewusst sehr schrill gewählt, kann all auch bei Bedarf deaktiviert werden. Für unseren Anwendungsfalls wurde eine Frequenz von 1000 Hertz und einer Dauer von 200 Millisekunden gewählt, um einen schnell wiederkehrenden Ton durch den Buzzer abzugeben im Falle einer zu schlechten Luftqualität.

## 4.3 Datenvorverarbeitung und Sourcecode

Die Datenvorverarbeitung findet größtenteils im Quellcode des Arduino Megas statt. Alle gemessenen Daten werden im Loop des Programms auf numerische Werte hin überprüft. Ist dies nicht der Fall, so werden im Produktionsreifen Programm keine Daten über die serielle Schnittstelle an den ESP und damit an das The Tings Network (TTN) gesendet. Ebenfalls wird über IF-Verzweigungen die Luftqualität in Stufen eingeteilt und an das Backend gesendet. Vom Backend werden die Daten für den Nutzer-Display bezogen. Das kann aus dem vorherigen Kapitel 4.1 in Abbildung 1 und Abbildung 2 eingesehen werden.

## 4.4 Übertragung der erfassten und vorverarbeiteten Daten vom Arduino zur IoT-Serveranwendung

Die Daten werden vom Arduino Mega zum Wifi LoRa ESP32 Board mithilfe der seriellen Schnittstelle übertragen zur Verringerung der Komplexität der Datenübertragung. Jedes dieser Boards hat zu diesem Zwecke sogenannte RX- und TX-Pins. Da nur in eine Richtung Daten übertragen werden, wird nur der TX-Pin des Arduino Megas und der RX-Pin des ESPs benötigt. TX steht dabei für Transmit, senden, und RX für Receive, empfangen.

Auf der Arduino-Seite müssen nur print-Ausgaben zum Senden der Daten über die serielle Schnittstelle gemacht werden wie in Abbildung 6 zu sehen ist.

```
140   Serial.print(temperature);
141   Serial.print(",");
142   Serial.println(humidity);
```

Abbildung 6: Arduino Serieller Output

Um die Daten auf dem ESP zu empfangen ist etwas mehr Code notwendig, da hier mitgeteilt werden muss, wer der Sender ist und was mit den empfangenen Daten gemacht werden soll. In Abbildung 7 wird dafür eine If-Verzweigung eingesetzt, um zu überprüfen, ob der serielle RX-Pin des ESP verfügbar ist und Daten empfangen kann. Alle Werte sind Komma-separiert und werden bis zum Zeilenumbruch gelesen. Liegt bei der seriellen Schnittstelle ein Fehler vor, so wird das Programm mit einer Fehlermeldung beendet. Über LoRa werden keine Werte an das TTN übertragen.

```
281   if (Serial.available() > 0) {
282     String data = Serial.readStringUntil('\n'); // read the incoming data until a newline character is received
283     temperature = data.substring(0, data.indexOf(',')).toFloat(); // extract the first value before the comma delimiter and convert it to a float
284     humidity = data.substring(data.indexOf(',') + 1).toFloat(); // extract the second value after the comma delimiter and convert it to a float
285     //float pressure = data.substring(data.indexOf(',') + 2).toFloat(); // extract the second value after the comma delimiter and convert it to a float
286
287     Serial.print("Value 1: ");
288     Serial.println(temperature);
289     Serial.print("Value 2: ");
290     Serial.println(humidity);
291     //Serial.print("Value 3: ");
292     //Serial.println(pressure);
293   } else {
294     Serial.println("Error at Serial");
295     return;
296   }
```

Abbildung 7: Empfangen der Daten auf dem ESP

## 4.5 Praktische Umsetzung der Serveranwendung

Die Daten werden im TTN mit einem Payload formatter entpackt und in eine JSON-Datei konvertiert. Diese JSON-Datei wird über einen Webhook an einen Webserver gesendet. Wir verwenden einen Webserver des Webhosters bplaced. Auf dem Webserver existiert eine MySQL Datenbank, auf die über phpMyAdmin zugegriffen werden kann. Die Datenbank wird über ein php-Skript konfiguriert. Ein weiteres php-Skript kümmert sich um die Verarbeitung der per Webhook gesendeten JSON-Datei. Das Skript liest die JSON-Datei aus und schreibt die darin enthaltenen Daten in die MySQL-Datenbank.

Um die Sicherheit der Datenbank zu erhöhen, sind Zugriffe von außerhalb des Webservers auf die Datenbank nicht gestattet. In der Datenbank erstellen wir eine Tabelle namens „dhbwiot_airquality“. Der Aufbau ist in Tabelle 1 dargestellt.

| Name | Datentyp |
|---|---|
| id | int |
| datetime | datetime |
| app_id | text |
| dev_id | text |
| ttn_timestamp | text |
| gtw_id | text |
| gtw_rssi | float |
| gtw_snr | float |
| dev_raw_payload | text |
| dev_value_1 | float |
| dev_value_2 | float |
| dev_value_3 | float |
| dev_value_4 | float |

Tabelle 1. Aufbau der MySQL Tabelle.

Die Spalten dev_value 1 bis 4 enthalten die Messdaten unserer Sensoren. Also Luftqualität, Luftdruck, Temperatur und Luftfeuchtigkeit. Jedes Mal, wenn Daten an den Webserver gesendet werden, wird in der Datenbank eine neue Zeile angelegt, in der die neuen Messdaten hinterlegt werden.

Neben der Datenbank läuft auf dem Webserver eine Webseite, die den aktuellen Eintrag in der Datenbank darstellt. Auf der Webseite werden der Timestamp, die Luftqualität, die Luftfeuchtigkeit, die Temperatur und die Luftqualität dargestellt.

Da die Datenbank externe Zugriffe blockiert, können Anwendungen wie Apps nicht auf die Daten zugreifen. Die Kernfunktion, der in Flutter geschriebenen App, besteht darin, Umgebungsdaten von einer externen Webseite zu extrahieren. Dies geschieht, indem die App eine Anfrage an die Webseite sendet, die Antwort verarbeitet und die relevanten Informationen aus dem empfangenen Inhalt extrahiert.

Um eine dynamische Aktualisierung der Benutzeroberfläche zu ermöglichen, verwendet die Flutter-App einen Stream, der den Datenstrom verwaltet und die Oberfläche bei Verfügbarkeit neuer Daten automatisch aktualisiert. Auf diese Weise kann der Benutzer stets die aktuellen Informationen einsehen, ohne die App manuell aktualisieren oder neu laden zu müssen.

Da die Datenbank, auf der die Umgebungsdaten gespeichert sind, keine direkten Zugriffe von außerhalb zulässt, ist die Methode zur Gewinnung von Daten von der HTML-Webseite von entscheidender Bedeutung. Die App kann somit nicht direkt mit der Datenbank kommunizieren, sondern muss die Daten über die Webseite beziehen, die als Schnittstelle zwischen der Datenbank und der App fungiert.


Abbildung 8: Screenshot der App

Abbildung 8 zeigt mit einem Screenshot der ersten Version der App die Integration der extrahierten Umgebungsdaten in die Benutzeroberfläche. Die App präsentiert die Informationen in verschiedenen, übersichtlichen Bereichen. Der Zeitstempel, die Luftqualität, die Luftfeuchtigkeit, die Gesundheit und die Temperatur werden in separaten, gut lesbaren Elementen dargestellt, die dem Benutzer helfen, die aktuellen Messwerte schnell und einfach zu erfassen.

Der gezeigte Bildschirm der App ist flexibel gestaltet und kann um weitere Informationskacheln ergänzt werden, sodass zusätzliche Umgebungsdaten oder weitere Kategorien von Informationen in die Benutzeroberfläche integrierbar sind. Die Anordnung der Kacheln ist übersichtlich und leicht verständlich, sodass der Benutzer auch bei einer größeren Anzahl von angezeigten Daten stets den Überblick behält.

# 5 Fazit

In dieser Arbeit haben wir „AirMetrics“, eine IoT-Lösung zur Messung der Luftqualität, vorgestellt und erfolgreich implementiert. Die Lösung bietet ein umfassendes System zur Echtzeitüberwachung der Luftverschmutzung.

Durch die Kombination von Arduino und LoRa-Technologie konnten wir eine effiziente und energiearme Datenübertragung gewährleisten und dabei eine weitreichende Abdeckung ermöglichen.

Zusätzlich wurde ein LCD-Display in Form eines Ampelsystems integriert, das den Benutzern eine schnelle und einfache Interpretation der aktuellen Luftqualität ermöglicht. Dieses visuelle System trägt dazu bei, das Umweltbewusstsein zu erhöhen.

Schließlich wurde eine App entwickelt, mit der die gesammelten Daten ausgewertet und visualisiert werden. Die App ermöglicht den Benutzern, die Luftqualität in ihrer Umgebung zu überwachen, und bietet eine Plattform für fundierte Entscheidungen in Bezug auf Umweltschutz und Gesundheit.