2021

Informatiklabor
2.Kolloquium

PROJEKT 4 – GRUPPE 4

TIMO STORM, LISA WEIDLER, MATTHIAS STARCK

Inhaltsverzeichnis

Arduino Code

Projekt4_Arduino

```
/*
Funktion: DHT 22-, SDS011-Sensoren auslesen, in Json-String speichern, auf WebSocket-Server hochladen.
          Zusätzlich Status mit LED-Streifen anzeigen lassen.

Autoren: Lisa Weidler, Timo Storm, Matthias Starck

Erstelldatum: 22.10.2021
Letzte Änderung: 07.12.2021

Titel: Informatiklabor, Projekt 4 - Gruppe 4
*/


#include <ESP8266WiFi.h>                                  //Import von Bibliothek für das Herstellen der Wlan-Verbindung
#include <WebSocketsServer.h>                             //Import von Bibliothek für Serverkommunikation
#include <ArduinoJson.h>                                  //Import von Bibliothek um JSON Dokumente erstellen zu können
#include <ESPDateTime.h>                                  //Import von Bibliothek um aktuelles Datum und Uhrzeit einzubinden
#include <DHT.h>                                          //Import von Bibliothek für den DHT22
#include <SDS011.h>                                       //Import von Bibliothek für SDS011
#include <Adafruit_NeoPixel.h>                            //Import von Bibliothek für den LED Streifen WS2812

#define LED D0                                            //LED auf Microcontroller wird über Ein-/Ausgang D0 gesteuert
#define DHT_TYPE DHT22                                    //genauen Typ des DHT festelegen
#define txd D2                                            //TXD des SDS011 mit Eingang D2 verbunden
#define rxd D3                                            //RXD des SDS011 mit Eingang D3 verbunden
#define dht_pin_2 D1                                      //DHT ist mit dem Eingang D1 verbunden
#define ssid "Xperia SL"                                  //Name des Wlans/Hotspots
#define password "29032409"                               //Passwort für das Wlan/Hotspot
#define Pin D6                                            //Pin für LED Streifen

const int output_timer = 5000;                            //Timer für das Messen der Daten (Messintervall)
const int capacity = JSON_OBJECT_SIZE(13);                //Größe des JSON Objekts auf 13 festgelegt
int led_status = HIGH;
WebSocketsServer server = WebSocketsServer(80);           //Server über den Kommunikation mit Clienten geschieht wird mit server angesprochen
int client_num;                                           //Variable für Clienten-ID
DHT dht(dht_pin_2, DHT_TYPE);                             //Der Temperatursensor wird ab jetzt mit „dht“ angesprochen, zugehöriger Pin wird übergeben
SDS011 sds;                                               //Der Feinstaubsensor wird mit "sds" angesprochen
int numPixels = 8;                                        //Anzahl der LEDs auf LED-Streifen
Adafruit_NeoPixel strip = Adafruit_NeoPixel(numPixels, Pin, NEO_GRB + NEO_KHZ800); //Übergabe des Pins und der Anzahl an LEDs auf dem Streifen
uint32_t red = strip.Color(255, 0, 0);                    //RGB Farbe Rot
uint32_t green = strip.Color(0, 255, 0);                  //RGB Farbe Grün
uint32_t yellow = strip.Color(255, 255, 0);               //RGB Farbe Gelb
uint32_t out = strip.Color(0,0,0);                        //LED aus
float p10, p25;                                           //Variablen für Feinstaubwerte
unsigned long op_rt = 0;                                  //Stand der zuletzt gespeicherten Millisekunden
long mil;
int c = 0;                                                //Variable für Index im Array "storage"
int i;                                                    //Zählvariable für LEDs
//bool error = false;
bool over_ten = false;                                    //Wahrheitswert für die Bildung des Durchschnitts auf falsch gesetzt
String json_string;
float avg[4];                                             //Array für die gebildeten Durchschnittswerte deklariert

float limits[][2] = {                                     //Array um kritische Werte zu definieren (temp,lf,pm10,pm2.5)
  {24.0, 0},
  {70.0, 0},
  {25.0, 0},
  {25.0, 0}
};

float storage[][10] = {                                   //Array um Messwerte einzuspeichern
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}};


void connect_to_network(){                                //Funktion: ESP soll sich mit dem Wlan/Hotspot verbinden
  WiFi.begin(ssid, password);
  while(WiFi.status() != WL_CONNECTED){                   //Schleife solange keine Verbindung besteht
    delay(1000);
    Serial.println("Connecting to WiFi...");              //Jede Sekunde ein neuer Verbindungsversuch + nette Ausgabe
  }


  Serial.println("Connected to Network");
  Serial.println(WiFi.localIP());                         //Bei Erfolgreicher Verbindung wird die IP-Adresse auf seriellem Monitor ausgegeben
}


void average(){                                           //Funktion zum Filtern der Messwerte (Durchschnitt bilden)
  int count = c + 1;
  float sum;
  for(int i=0; i<=3; i++){                                //Index für Zeilen wird pro Durchgang erhöht
    for(int j=0; j<=9; j++){                              //Index für Spalten wird pro Durchgang erhöht
      sum = sum + storage[i][j];                          //Für jede Zeile werden die Werte der Spalten aufsummiert
    }
    if(over_ten){                                         //Wenn jede Spalte der Zeile einen Wert hatte, dann wird die vorher gebildete Summe
      avg[i] = sum / 10;                                  //durch 10 geteilt
    }
    else{                                                 //Wenn nicht wird durch die Anzahl der Spalten mit einem Wert geteilt
      avg[i] = sum / count;
    }
    sum = 0;
  }
}
```

```
void pack_json(){                                                    //Funktion, um Messwerte in ein JSON Objekt zu wandeln
  json_string = "";
  StaticJsonDocument<capacity> doc;                                  //JSON Dokument wird angelegt (Objekte und Werte können hierin gespeichert werden)
  doc["timestamp"] = DateTime.toString();                            //Jeder Messreihe wird Datum und Uhrzeit hinzugefügt, in der das JSON Objekt erstellt wird
  JsonObject filtered = doc.createNestedObject("filtered");          //Erstes JSON Objekt für die aktuellen Durchschnittswerte (gefiltert)
    filtered["temperature"] = avg[0];                                //Mittelwert-Temperatur zu JSON Dokument hinzufügen
    filtered["humidity"] = avg[1];                                   //Mittelwert-Luftfeuchtigkeit zu JSON Dokument hinzufügen
    filtered["pm25"] = avg[2];                                       //Mittelwert-PM2.5 zu JSON Dokument hinzufügen
    filtered["pm100"] = avg[3];                                      //Mittelwert-PM10 zu JSON Dokument hinzufügen
  JsonObject unfiltered = doc.createNestedObject("unfiltered");      //Zweites JSON Objekt für die aktuellen Messwerte (ungefiltert)
    unfiltered["temperature"] = storage[0][c];                       //Temperatur zu JSON Dokument hinzufügen
    unfiltered["humidity"] = storage[1][c];                          //Luftfeuchtigkeit zu JSON Dokument hinzufügen
    unfiltered["pm25"] = storage[2][c];                              //PM2.5 zu JSON Dokument hinzufügen
    unfiltered["pm100"] = storage[3][c];                             //PM10 zu JSON Dokument hinzufügen

  serializeJson(doc, json_string);                                   //Json Dokument in Bytestring umwandeln
  Serial.println(json_string);
}



//Funktion um Aktionen auf Websocket zu bearbeiten braucht Klientennummer, Aktionstyp, optional Daten zum Übertragen und deren Länge
void onWebSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length){
  switch(type){                                                      //Fallunterscheidung
    //Verbindung von Client unterbrochen
    case WStype_DISCONNECTED:                                        // Fall 1: vorhandene Verbindung abgebrochen
      Serial.printf("[%u] Disconnected!\n", num);                    //Gibt auf Monitor aus, welche Client nicht mehr Verbunden ist
      break;

    // Neue Verbindung von Client
    case WStype_CONNECTED:                                           // Fall 2: Neue Verbindung hergestellt
    {
      IPAddress ip = server.remoteIP(num);                           //Holt IP-Adresse des verbundenen Clients
      Serial.printf("[%u] Connection from ", num);
      Serial.println(ip.toString());                                 //Gibt IP-Adresse aus
      client_num = num;                                              //Speichert Nummer des verbundene Clients
    }
    break;

    //Alle anderen Aktionen ingorieren
    case WStype_TEXT:
    case WStype_BIN:
    case WStype_ERROR:
    case WStype_FRAGMENT_TEXT_START:
    case WStype_FRAGMENT_BIN_START:
    case WStype_FRAGMENT:
    case WStype_FRAGMENT_FIN:
    default:
      break;

  }
}
void Okay (){                                                        //LED-Status-Funktion alle Werte sind okay
  for (i=1;i<=3; i++){                                               //LED 2-4 Farbe auf grün gestellt
    strip.setPixelColor(i,green);

  }
  for (i 4;i<=numPixels;i++){                                        //restliche LEDs (5-8) auf aus gestellt
    strip.setPixelColor(i,out);
  }
    strip.show();                                                    //LEDs wie eingestellt leuchten lassen
}

void Increased(){                                                    //LED-Status-Funktion mindestens ein Wert ist erhöht
   for (i=1;i<=3; i++){                                              //LED 2-4 Farbe auf grün gestellt
    strip.setPixelColor(i,green);

      }

    strip.setPixelColor(4,yellow);                                   //LED 5 und 6 auf gelb gestellt
    strip.setPixelColor(5,yellow);

    strip.setPixelColor(6,out);                                      //LED 7 und 8 auf aus gestellt
    strip.setPixelColor(7,out);
    strip.show();                                                    //LEDs wie eingestellt leuchten lassen
}


void Dangerous(){                                                    //LED-Status-Funktion mindestens ein Wert ist viel zu hoch
   for (i=1;i<=3; i++){                                              //LED 2-4 Farbe auf grün gestellt
    strip.setPixelColor(i,green);
      }

    strip.setPixelColor(4,yellow);                                   //LED 5 und 6 auf gelb gestellt
    strip.setPixelColor(5,yellow);

    strip.setPixelColor(6,red);                                      //LED 7 und 8 auf rot gestellt
    strip.setPixelColor(7,red);
    strip.show();                                                    //LEDs wie eingestellt leuchten lassen
}


void status_LED(){                                                  //Funktion für LED Streifen, wann welcher Fall eintritt(okay,increased,dangerous)
  for(int j=0; j<=3; j++){                                          // Prüft ob ein oder mehrere Werte ihren Grenzwert überschreiten
      if(avg[j] > limits[j][0]){
         float dif = avg[j] - limits[j][0];                         //Betrag der Abweichung ermitteln
         limits[j][1] = 1;
         if(dif > 0 && dif <= 3){                                   //Fallunterscheidung: bei 0-3: increased, >3: dangerous
          increased();
         }
```

```
        }
        else if(dif > 3){
         Dangerous();
        }
      }
      else{
          limits[j][1] = 0;
      }
    }
    float sum = 0;
    for(int j=0; j<=3; j++){
      sum = sum + limits[j][1];                          //gucken ob keine Grenze überschritten ist
    }
    if(sum == 0){
       Okay();                                           //Wenn keine Grenze überschritten: okay
    }
}



void setup(){
  Serial.begin(115200);                                  //seriellen Monitor starten mit Baudrate 115200Bd
  dht.begin();                                           //DHT22 starten
  sds.begin(txd, rxd);                                   //SDS011 starten und Pins übergeben
  strip.begin();                                         //LED Streifen starten

  DateTime.setTimeZone("CET-1");                         //Zeitzone auswählen
  DateTime.begin();                                      //Zeit Modul starten

  connect_to_network();                                  //Mit Wlan verbinden

  server.begin();                                        //Server starten
  server.onEvent(onWebSocketEvent);                      //


  strip.setBrightness(75);                               //Helligkeit der LEDs von 0 dunkel bis 255 ganz hell auswählen
  strip.setPixelColor(0,strip.Color(0,0,255));           //erste LED soll blau sein, um Funktionstüchtigkeit des LED-Streifens anzuzeigen
  strip.show();

  pinMode(LED, OUTPUT);                                  //Pin der Onboard LED als Ausgang geschaltet (standardmäßig auf Eingang geschaltet)
}


void loop(){
  mil = millis();                                        //aktuelle Anzahl an Millisekunden, seit dem das Programm gestartet wurde in Variable gespeichert

  if(mil > op_rt + output_timer){                        //alle 5 Sekunden (entsprechend "output_timer") soll eine Messung der Werte durchgeführt werden
    op_rt = mil;
    sds.read(&p25, &p10);                                //Feinstaubwerte auslesen
    storage[0][c] = dht.readTemperature();               //Temperatur auslesen und zu Filter-Array hinzufügen
    storage[1][c] = dht.readHumidity();                  //Luftfeuchtigkeit auslesen und zu Filter-Array hinzufügen
    storage[2][c] = p25;                                 //Feinstaubwerte zu Filter-Array hinzufügen
    storage[3][c] = p10;

    average();                                           //Durchschnitt der letzten 10 Werte bilden
    pack_json();                                         //Messwerte und Durchschnitte als JSON Dokument speichern
    server.sendTXT(client_num, json_string);             //JSON Dokument über Server an Client übersenden
    status_LED();                                        //LED Streifen entsprechend der Werte aufleuchten lassen

    c++;                                                 //Wenn 10 Werte in Storage-Matrix gespeichert sollen die ersten wieder überschrieben werden
    if(c == 10){                                         //desahlb wird die Zählvariable c nach 10 durchgängen wieder auf 0 gesetzt
      c = 0;
      over_ten = true;
    }
  }
  server.loop();                                         //Server aktiv halten
}
```

Client und Bot Code in Python

```python
# Funktion: Daten über einen Websocket-Server eines Mikrocontrollers auslesen diese in
#           einer MySQL-Datenbank speichern.
#           Einen Telegram-Bot steuern
#
# Autoren: Lisa Weilder, Timo Storm, Matthias Starck
#
# Erstelldatum: 26.10.2021
# Letzte Änderung: 12.12.2021
#
# Titel: Informatiklabor, Projekt 4 - Gruppe 4


# imports
from time import sleep
import websocket
import json
import mysql.connector
import bot
from threading import Thread


# Dictionary zum speichern von Attributen der verschiedenen Werte, hier sparsamer als eine Klasse
grenzwerte = {
    'temperature': [24, 10, False, 'Temperatur', '°C', '🔥'],
    'humidity': [70.0, 0, False, 'Luftfeuchtigkeit', '%', '🏜'],
    'pm25': [25.0, 0, False, 'pm2.5', 'µg / m³', '🚨'],
    'pm100': [25.0, 0, False, 'pm10', 'µg / m³', '🚨']
}


# ================ WS/DB-Funktionen ===================

# Funktion zum Abrufen der aktuelle Messdaten in Form eines JSON-Strings, gibt Dictionary zurück
def getData():
    # Verbinde mit WebSocket-Server
    ws = websocket.WebSocket()
    ws.connect("ws://192.168.169.122")

    # Erhalte Daten vom Server
    data = json.loads(ws.recv())

    # Schließe die Verbindung
    ws.close()
    return data


# Erstellt die Verbindung zur Datenbank, gibt diese zurück
def connection():
    db = mysql.connector.connect(
        host="192.168.169.21",
        user="Labor2021",
        password="loveit",
        database="testdb"
    )
    return db


# Funktion zum Speichern der Daten in der Datenbank,
# nimmt das vom Server erhaltene Dictionary als Argument, kein Rückgabewert
def insertData(data):
    with connection() as db:
        c = db.cursor()
        com = "INSERT INTO weather_uair VALUES(0, %s, %s, %s, %s, %s)"
        val = [data["timestamp"]]
        for value in data["unfiltered"].values():
            val.append(value)
        c.execute(com, val)
        db.commit()

        c = db.cursor()
        com = "INSERT INTO weather_fair VALUES(0, %s, %s, %s, %s, %s)"
        val = [data["timestamp"]]
        for value in data["filtered"].values():
            val.append(value)
        c.execute(com, val)
        db.commit()


# ==================== Bot-Funktionen ==========================

# Funktion überprüft, ob die aktuellen Werte im Gültigkeitsbereich liegen,
# wenn nicht wird über den Telegram-Bot eine Warnung geschickt
# nimmt Dictionary der aktuellen werte als Argument, keine Rückgabewerte
def bot_sends_warning(data):
    for key in data.keys():
        if grenzwerte[key][0] > data[key] > grenzwerte[key][1]:
            msg = f"{grenzwerte[key][3]}-Messwerte wieder optimal: {data[key]} {grenzwerte[key][4]} ✅"
            if grenzwerte[key][2]:
                bot.send_message(text=msg)
```

```
                grenzwerte[key][2] = False
        else:
            msg = f"{grenzwerte[key][3]}-Messwerte außerhalb des gültigen Bereiches: {data[key]} {grenzwerte[key][4]} {grenzwerte[key][5]}"
            if not grenzwerte[key][2]:
                bot.send_message(text=msg)
                grenzwerte[key][2] = True


# Funktion zum Reagieren auf Anfragen des Botnutzers
# Nimmt Dictionary der aktuellen Werte und den Text der Nachricht des Nutzers als Argument, keine Rückgabewert
def bot_reacts(data, msg):
    msg = msg.lower()
    if msg == 'aktuelle werte':
        bot.send_message(temp=data['temperature'], hum=data['humidity'], pm25=data['pm25'], pm100=data['pm100'])
    elif msg == 'temperatur':
        bot.send_message(temp=data['temperature'])
    elif msg == 'luftfeuchtigkeit':
        bot.send_message(hum=data['humidity'])
    elif msg == 'feinstaub':
        bot.send_message(pm25=data['pm25'], pm100=data['pm100'])


# ================================ Threading-Funktionen ================================================================

current = {}                                                    # Dictionary zum Speichern der aktuellen Daten


# Schleife zu aufrufen der Bot-Funktionen
def botHandler():
    msg_index = 0
    while True:
        global current                                          # Ruft aktuelle Daten ab
        if current != {}:
            filtered = current['filtered']                      # Filtert die ungefilterten Daten heraus
            bot_sends_warning(filtered)                         # Ruft Warnfunktion des Bots auf
            try:                                                # Hilft Programmabbruch bei Fehler vorzubeugen
                msg, index = bot.get_message()                  # Ruft Inhalt und Index der letzten Nachricht des Nutzers ab
                if index > msg_index:                           # Guckt ob Nachricht schon bearbeitet wurde
                    msg_index = index                           # Aktualisiert letzten Index
                    bot_reacts(filtered, msg)                   # Ruft Reaktionsfunktion des Bots auf
            except:
                pass
```

bot.py > ...

```
# Libraries importieren
import requests
import json


# Dictionary zum speichern von Attributen der verschiedenen Werte, hier sparsamer als eine Klasse
grenzwerte = {
    'temp': [24, 10, False, 'Temperatur', '°C', '🔥'],
    'hum': [70.0, 0, False, 'Luftfeuchtigkeit', '%', '🏜️'],
    'pm25': [25.0, 0, False, 'pm2.5', 'µg / m³', '🚨'],
    'pm100': [25.0, 0, False, 'pm10', 'µg / m³', '🚨']
}

token = "2045348848:AAEFEyoIZ6r-e7I-LfR461ykUhalWPJlFT0"                            # Spezifischer Token für Bot

# Funktion zum abrufen der letzten Nachricht des Nutzers, gibt Inhalt der Nachricht und Index dieser zurück
def get_message():
    answer = requests.get(f"https://api.telegram.org/bot{token}/getUpdates")        # Get-request an Telegram-Bot-Chat,
                                                                                    # gibt uns unten benötigte chat_id
    content = answer.content
    data = json.loads(content)                                                      # Speichern der Daten in einem Dictionary

    try:
        message = data['result'][len(data['result'])-1]['message']['text']          # Herrausfiltern der letzten Nachricht
    except:
        return ''
    return message, len(data['result'])                                             # Rückgabe des Inhalts und des Indexes


# Funktion zum Versenden von Nachrichten durch den Bot
def send_message(**args):
    # die chat_id ist die aus der obigen Response
    msg = ''
    for key in args.keys():                                                         # Fügt für jedes übergeben Argument den passenden Text
                                                                                    # zum msg-string hinzu
        if key == 'text':
            msg = msg + args[key] + '\n'
        else:
            msg = msg + f"{grenzwerte[key][3]}: {args[key]} {grenzwerte[key][4]}\n"

    params = {"chat_id":"2057046705", "text": msg}
    url = f"https://api.telegram.org/bot{token}/sendMessage"
    message = requests.post(url, params=params)                                     # POST-request an den Telegram-Bot-Chat
    data = json.loads(message.content)
```

Django-Backend in Python

```python
# Verbindung zu Datenbank hinterlegen
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'testdb',
        'USER': 'Labor2021',
        'PASSWORD': 'loveit',
        'HOST': '127.0.0.1',
    }
}
```

```python
from django.db import models


# Django Models als Python Klassen zum Laden von Datenbankeinträgen
class uAir(models.Model):
    timestamp = models.DateTimeField()          # Datentyp DateTime
    temperature = models.FloatField()           # Datentyp Float
    humidity = models.FloatField()              # Datentyp Float
    pm25 = models.FloatField()                  # Datentyp Float
    mp100 = models.FloatField()                 # Datentyp Float


class fAir(models.Model):
    timestamp = models.DateTimeField()          # Datentyp DateTime
    temperature = models.FloatField()           # Datentyp Float
    humidity = models.FloatField()              # Datentyp Float
    pm25 = models.FloatField()                  # Datentyp Float
    mp100 = models.FloatField()                 # Datentyp Float

```

```python
from django.urls import path
from . import views

# Legt fest, welche funktionen zu welcher URL gehören. URL bekommen 'Spitznamen' zum leichteren Aufrufen
app_name = 'weather'
urlpatterns = [
    path('', views.index_view, name='index'),
]


```

```python
from django.shortcuts import render                         # Benötigte Imports
from plotly.offline import plot
from plotly.graph_objs import Scatter, Bar
from .models import uAir, fAir


def index_view(request):
    current = fAir.objects.order_by('-timestamp')[0]            # Lädt zuletzt gespeichertes Element aus der Datenbank,
    entries = [fAir.objects.order_by('-timestamp')[0:30],       # entsprechender SQL-Befehl: SELECT * FROM weather_fair SORT BY timestamp DESC
               uAir.objects.order_by('-timestamp')[0:30]]       # Lädt die 30 zuletzt gespeicherten Elemente aus der Datenbank

    t_graphs = []                                               # Arrays zum Speichern der Plotly-Daten
    h_graphs = []
    pm_graphs = []

    for i in range(2):
        x_data = []                                             # Arrays zum Sortieren der Daten nach Messwert
        temp = []
        humidity = []
        p25 = []
        p100 = []
        for x in entries[i]:
            x_data.append(x.timestamp)                          # Zuweisung der Messwerte der einzelnen Einträge zum passenden Array
            temp.append(x.temperature)
            humidity.append(x.humidity)
            p25.append(x.pm25)
            p100.append(x.mp100)

        print(humidity)

        if i == 0:                                              # Unterscheidung welcher Name dem Datensatz zugeordnet werden muss
            name = 'gefiltert'
        else:
            name = 'ungefiltert'

        # Ploten der Datensätze mit Plotly
        t_graphs.append(Scatter(x=x_data, y=temp, mode='lines+markers', name=name, opacity=1, marker_size=10))
        h_graphs.append(Bar(x=x_data, y=humidity, name=name, opacity=1))
        pm_graphs.append(Scatter(x=x_data, y=p25, mode='lines+markers', name='pm2.5 ' + name, opacity=1, marker_size=10))
        pm_graphs.append(Scatter(x=x_data, y=p100, mode='lines+markers', name='pm10 ' + name, opacity=1, marker_size=10))

    # Transformieren der Plots zur einfacheren Einbindung in das HTML Dokument
    plot_temp = plot({'data': t_graphs, 'layout': {'xaxis_title': 'Time', 'yaxis_title': 'Temperatur / °C'}}, output_type='div')
    plot_hum = plot({'data': h_graphs, 'layout': {'xaxis_title': 'Time', 'yaxis_title': 'Luftfeuchtigkeit / %'}}, output_type='div')
    plot_p25 = plot({'data': pm_graphs, 'layout': {'xaxis_title': 'Time', 'yaxis_title': 'Feinstaub / µg/m³'}}, output_type='div')

    # Aufrufen des zu rendernden HTML-Dokuments und Übergabe der Graphen an dieses
    return render(request, 'weather/index.html', {'current': current, 'plot_temp': plot_temp, 'plot_hum': plot_hum, 'plot_p25': plot_p25})

```

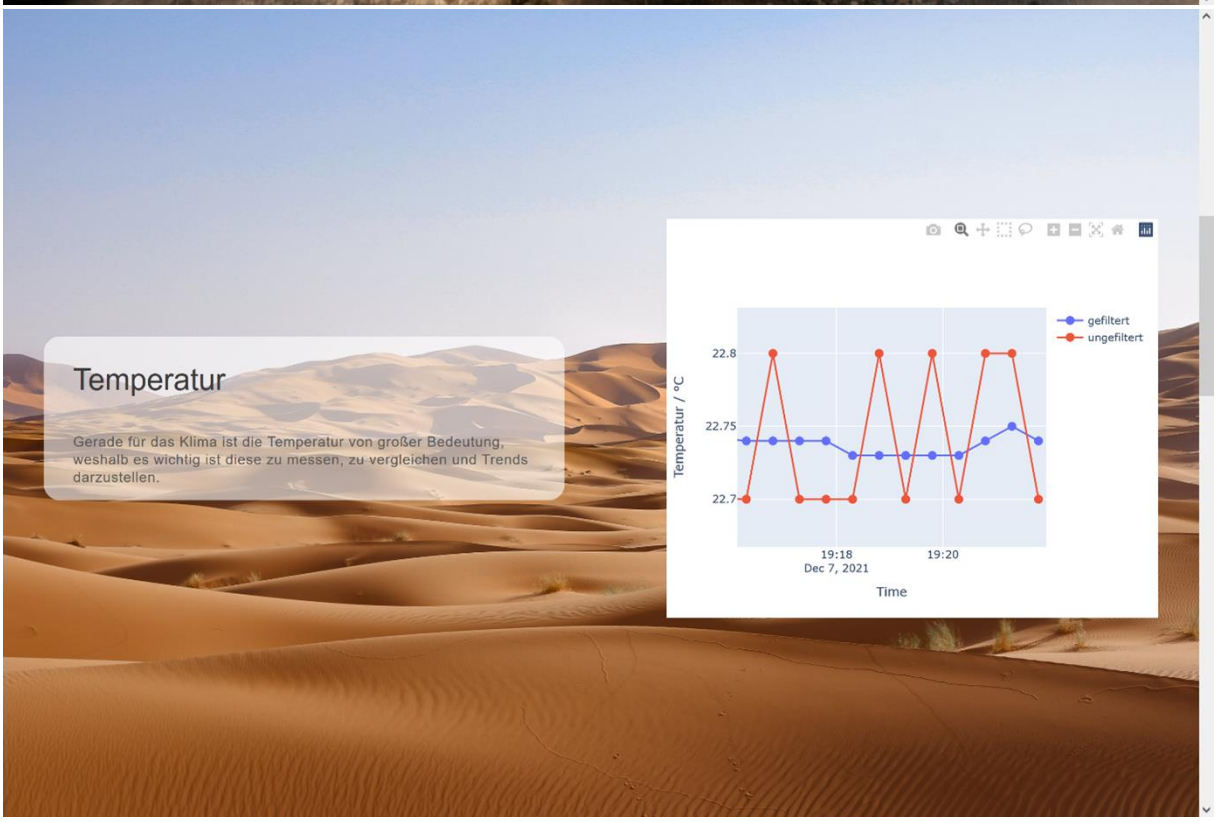
HTML und CSS der Website

```
{% load static %}                                                                   <!--Django-Tag zum Laden der statischen Daten, z.B. Bilder, css-Dokumente-->

<!DOCTYPE html>                                                                     <!--Dokumententyp und Sprache festlegen -->
<html lang="de">                                                                    <!--Beginn des HTML-Dokuments-->
    <head>
        <meta charset="UTF-8">                                                      <!--Zeichencodierung festlegen-->
        <title>Luftdaten</title>                                                    <!--Titel der Webseite-->
        <link rel="stylesheet" href="{% static 'weather/css/style3.css' %}">        <!--Einbindung des style-sheets-->
    </head>
    <header>                                                                        <!--Beginn Header-->
        <div id="start" style="background-image: url('static/weather/img/main_bg.jpg');">   <!--ID-Zuweisung, Einbindung des ersten Hintergrundbildes-->
            <div style="position: absolute; left: 50%">                             <!--Festlegen der Position der Übersicht der aktuellen Werte-->
                <div id="aktuell">
                    <h1>Aktuelle <br /> Werte</h1>                                  <!--Auf Titelseite dargestellter Text und mit Hyperlinks versehene Icons-->
                    <div class="icon">
                        <a href="#Temperatur"><img src="{% static 'weather/img/temp_icon.png' %}" width=100 height=100 style="margin: auto; display: block" alt="temperatur-icon"></a>
                        <h2 style="text-align: center;">{{ current.temperature }} °C</h2>
                    </div>
                    <div class="icon">
                        <a href="#Luftfeuchtigkeit"><img src="{% static 'weather/img/humidity_icn.png' %}" width=100 height=100 style="margin: auto; display: block" alt="temperatur-icon"></a>
                        <h2 style="text-align: center;">{{ current.humidity }} %</h2>
                    </div>
                    <div class="icon">
                        <a href="#Feinstaubbelastung"><img src="{% static 'weather/img/dust_icn.png' %}" width=100 height=100 style="margin: auto; display: block" alt="temperatur-icon"></a>
                        <h2 style="text-align: center; white-space: nowrap;">{{ current.pm25 }} µg/m³</h2>
                    </div>
                </div>
            </div>
        </div>
    </header>                                                                       <!--Ende Header-->
    <body>                                                                          <!--Beginn Body-->
        <div id="content">
            <section id="Temperatur" style="background-image: url('static/weather/img/dessert_bg.jpg');">
                <div class="description" style="margin-right: 115px;">              <!--Klassenzuweisung für Beschreibung, um Formatierung im Folgenden zu vereinheitlichen-->
                    <h1>Temperatur</h1>                                             <!--Überschrift des Absatzes-->
                    <p style="line-height: 20px">Gerade für das Klima ist die Temperatur von großer     <!--Inhalt des Absatzes-->
                        Bedeutung, weshalb es wichtig ist diese zu messen, zu vergleichen
                        und Trends darzustellen.</p>
                </div>
                <div class="graph">                                                 <!--Klassenzuweisung für Graph, um Formatierung im Folgenden zu vereinheitlichen-->
                    {% autoescape off %}                                            <!--Darstellung HTML-spezifischer Zeichen ermöglichen-->
                        {{ plot_temp }}                                             <!--Aufrufen des Temperatur-Diagramms-->
                    {% endautoescape %}                                             <!--Ende Darstellung HTML-spezifischer Zeichen-->
                </div>
            </section>
            <section id="Luftfeuchtigkeit" style="background-image: url('static/weather/img/rain_bg.jpg');">
                <div class="graph">
                    {% autoescape off %}
                        {{ plot_hum }}
                    {% endautoescape %}
                </div>
                <div class="description" style="margin-left: 115px;">
                    <h1>Luftfeuchtigkeit</h1>
                    <p style="line-height: 20px">Auch die Luftfeuchtigkeit spielt eine bedeutende Rolle, da sie einen
                                                 großen Einfluss auf den Menschen und seine Umwelt hat. So sind wir bei
                                                 hoher Luftfeuchtigkeit weniger belastbar, die Schimmelgefahr ist höher
                                                 und Autos korrodieren schneller.</p>
                </div>
            </section>
            <section id="Feinstaubbelastung" style="background-image: url('static/weather/img/dust_bg.jpg');">
                <div class="description" style="margin-right: 115px;">
                    <h1>Feinstaubbelastung</h1>
                    <p style="line-height: 20px">Feinstaub wirkt sich negativ auf die Gesundheit des Menschen aus.
                                                 Dieser trägt nicht nur Schwermetalle und Krebs erregende Kohlenwasserstoffe
                                                 auf seiner Oberfläche in den Körper, sondern kann auch nicht mehr abgeatmet
                                                 oder von den Reinigungszellen der Lunge bekämpft werden.</p>
                </div>
                <div class="graph">
                    {% autoescape off %}
                        {{ plot_p25 }}
                    {% endautoescape %}
                </div>
            </section>
        </div>
    </body>                                                                         <!--Ende Body-->
    <footer>                                                                        <!--Beginn Footer-->
        <a href="#top"><div class="arrow bounce" style="background-image: url('static/weather/img/arrow_top_icn.png');"></div></a>  <!--Hyperlink zum Anfang der Webseite in Form eines hüpfenden Pfeils-->
        <p>&copy; Timo Storm | Matthias Starck | Lisa Weidler</p>
    </footer>                                                                       <!--Ende Footer-->
</html>                                                                             <!--Ende des HTML-Dokuments-->
```

```css
/*
Funktion: Formatierung der HTML-Website

Autoren: Lisa Weidler, Timo Storm, Matthias Starck

Erstelldatum: 03.11.2021
Letzte Änderung: 12.12.2021

Titel: CSS für Informatiklaboraufgabe 4
*/

html, body{                                 /*Allgemein Abstände und Schriftart festlegen*/
    margin: 0;
    padding: 0;
    font-family: 'Montserrat', sans-serif;
}

#start{                                     /*Höhe und Positionierung des Hintergrundbildes für alles mit der ID "Start"*/
    height: 100vh;
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;
    position: relative;
}

#aktuell{                                   /*Style für Div mit der ID "aktuell" festlegen. Z.B. Höhe, Rand und Hintergrundfarbe)*/
     height: 20vh;
     width: 150%;
     display: block;
     margin-top: 60vh;
     left: -75%;
     padding: 3%;
     background: rgba(50, 50, 50, 0.8);
     border-radius: 5vh;
     position: relative;
 }

#aktuell .icon{                             /*Style für Icons im Div "aktuell". Invertierung der Farbe, Anordnung und Breite*/
    filter: invert(1);
    float: left;
    width: 25%;
}

#aktuell h1{                                /*Style für Überschrift der Größe 1 im Div "aktuell"*/
    margin-top: 5vh;
    width: 25%;
    float: left;
    color: white;
    vertical-align: middle;
}

section{                                    /*Style für alle Sections festlegen.*/
    height: 75vh;
    display: flex;
    justify-content: center;
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;
    padding: 120px;
}

.description{                               /*Style für Div's der Klasse "description": Anordnung, Hintergrundfarbe...*/
    flex: 0 0 50%;
    max-width: 50%;
    align-self: center;
    background: rgba(255, 255, 255, 0.6);
    border-radius: 2vh;
    padding-left: 3%;
}

.description h1{                            /*Style für Überschriften der Größe 1 der Texte*/
    color: #3b3a39;
    font-weight: 500;
    font-size: 34px;
    line-height: 1.2em;
    margin-bottom: 40px;
}

.description p{                             /*Style für Absätze der Texte*/
    letter-spacing: 0.6px;
    color: #595959;
    text-align: left;
    font-size: 16px;
}

.graph{                                     /*Style für Div's der Klasse "graph"*/
    flex: 0 0 50%;
    max-width: 50%;
```

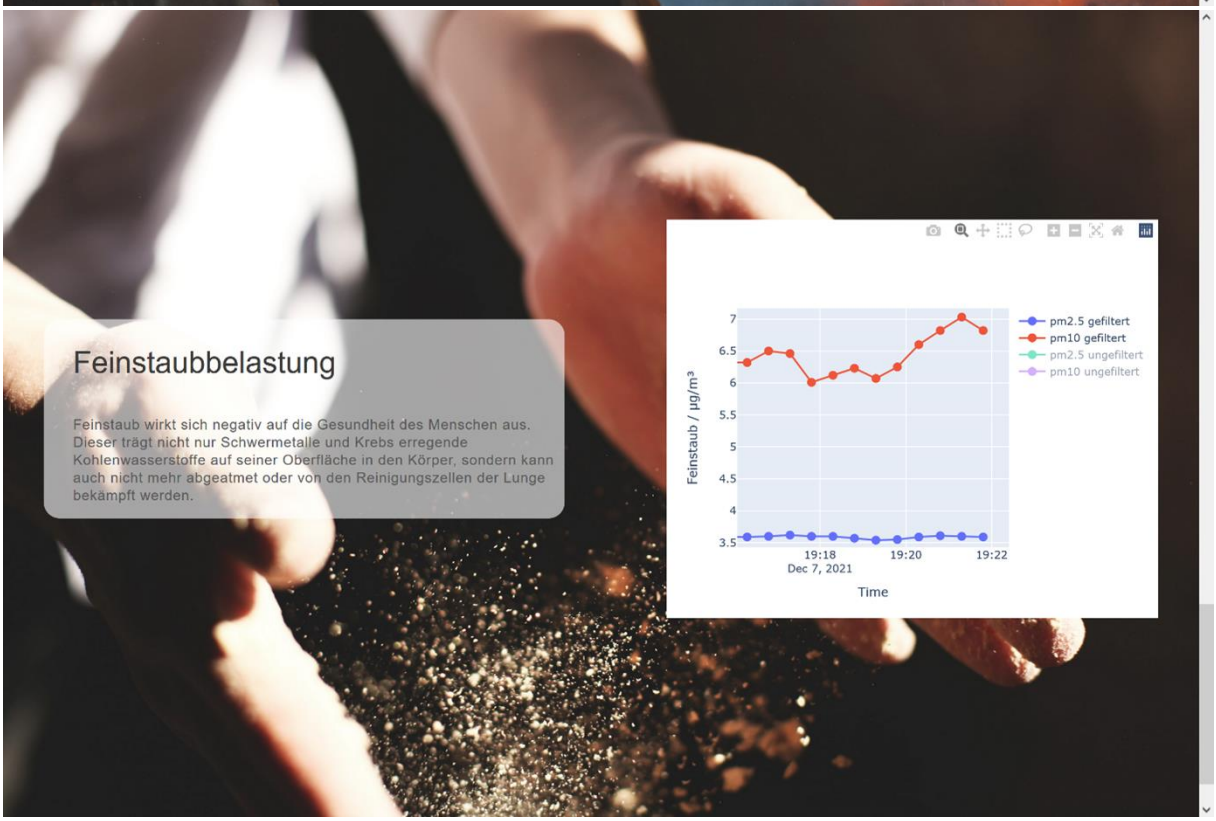
```
    align-self: center;
}

.arrow{                                     /*Style für Hüpfenden Pfeil im Footer*/
    position: relative;
    bottom: -2rem;
    left: 50%;
    margin-left: -20px;
    width: 60px;
    height: 40px;
    background-size: contain;
    background-repeat: no-repeat;
}

.bounce:hover{                              /*Aufrufen der Animation, wenn der Mauszeiger über dem Pfeil schwebt*/
    animation: bounce 2s infinite;
}

@keyframes bounce{                          /*Definierung der Keyframes der Animation (Verschiebung in Y-Richtung)*/
    0%, 20%, 50%, 80%, 100%{
        transform: translateY(0);
    }
    40%{
        transform: translateY(-30px);
    }
    60%{
        transform: translateY(-15px);
    }
}

footer{                                     /*Style des Footers*/
    text-align: center;
    padding-bottom: 3vh;
}
```

Website in Fotos



Luftfeuchtigkeit

Auch die Luftfeuchtigkeit spielt eine bedeutende Rolle, da sie einen großen Einfluss auf den Menschen und seine Umwelt hat. So sind wir bei hoher Luftfeuchtigkeit weniger belastbar, die Schimmelgefahr ist höher und Autos korrodieren schneller.



Feinstaubbelastung

Feinstaub wirkt sich negativ auf die Gesundheit des Menschen aus. Dieser trägt nicht nur Schwermetalle und Krebs erregende Kohlenwasserstoffe auf seiner Oberfläche in den Körper, sondern kann auch nicht mehr abgeatmet oder von den Reinigungszellen der Lunge bekämpft werden.