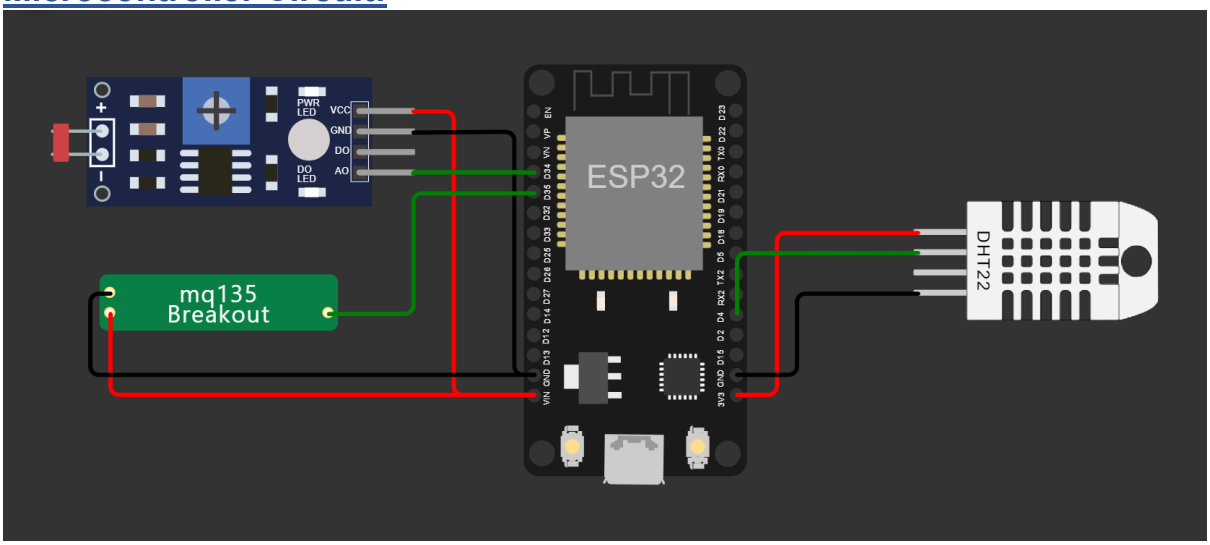
IoT Weather and Environment Monitoring System

Microcontroller Circuit:



Microcontroller Program:

```
import machine
import dht
import time
import requests
import json
from base64 import b64decode as b64d
from base64 import b64encode as b64e

ldr_pin = machine.ADC(34)
mq135_pin = machine.ADC(35)
dht_sensor = dht.DHT22(4)

def get_ist_datetime():
    response = requests.get("https://timeapi.io/api/Time/current/zone?timeZone=Asia/Kolkata")
    dt = response.json()
    date  = dt['date']
    time = dt['time']
    return date, time

def push(dat):
```

```
    time = get_ist_datetime()
    repository_owner = 'josephsharon07'
    repository_name = 'CAD_Phase1'
    file_path = 'Database/Database.json'
    branch_name = 'main'
    file_url = f'https://api.github.com/repos/{repository_owner}/{repository_name}/contents/{file_path}'
    access_token = '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'
    headers = {
    'Authorization': f'token {access_token}',
    'Accept': 'application/vnd.github.v3+json',
    'User-Agent': 'IBM-Project'
}
    try:
        response = requests.get(file_url, headers=headers)
        if response.status_code == 200:
            data = response.json()
            content_base64 = data['content']
            content = b64d(content_base64).decode('utf-8')
            data_to_update = json.loads(content)
            data_to_update[str(time[0])+' '+str(time[1])] = dat
            data_to_update = json.dumps(data_to_update).encode('utf-8')
            updated_content = b64e(data_to_update).decode()
            commit_message = str(time[0])+' '+str(time[1])
            payload = {
                "message": commit_message,
                "content": updated_content,
                "sha": data['sha'],
                "branch": branch_name
            }
            response = requests.put(file_url, headers=headers, json=payload)

            if response.status_code == 200:
                print("Database updated successfully.")
                print("--------------------------------------------------------------------------------")
```

```
            else:
                print("File update failed with status code:", response.status_code)
                print(response.json())
        else:
            print("GitHub request failed with status code:", response.status_code)

    except Exception as e:
        print("An error occurred:", e)

def push_(dat):
    repository_owner = 'josephsharon07'
    repository_name = 'CAD_Phase1'
    file_path = 'Database/liveData.json'
    branch_name = 'main'
    file_url = f'https://api.github.com/repos/{repository_owner}/{repository_name}/contents/{file_path}'
    access_token = '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'
    headers = {
        'Authorization': f'token {access_token}',
        'Accept': 'application/vnd.github.v3+json',
        'User-Agent': 'IBM-Project'
    }

    try:
        response = requests.get(file_url, headers=headers)
        data = response.json()
        data_to_update = {}
        time = get_ist_datetime()
        data_to_update[str(time[0]) + ' ' + str(time[1])] = dat
        data_to_update = json.dumps(data_to_update).encode('utf-8')
        updated_content = b64e(data_to_update).decode()
        commit_message = str(time[0]) + ' ' + str(time[1])
        payload = {
            "message": commit_message,
            "content": updated_content,
```

```
            "sha": data['sha'],
            "branch": branch_name
        }
        response = requests.put(file_url, headers=headers, json=payload)
        if response.status_code == 200:
            print("Live data updated successfully.")
            print("--------------------------------------------------------------------------------")
        else:
            print("File update failed with status code:", response.status_code)
            print(response.json())

    except Exception as e:
        print("An error occurred:", e)



def read_ldr():
  value = ldr_pin.read()
  if value > 4063 or value < 32:
    raise ValueError("Value is not within the valid range.")
  converted_value = (value - 32) / 4031 * 100000
  converted_value = 100000 - converted_value
  return int(converted_value)

def read_mq135():
    value = mq135_pin.read()
    values = (value / 4095) * 1000
    return int(values)

def read_dht22():
    dht_sensor.measure()
    temp_celsius = dht_sensor.temperature()
    humidity = dht_sensor.humidity()
    return temp_celsius, humidity
```

```
while True:
  print("Temperature : "+str(int(read_dht22()[0]))+"°C")
  print("Humidity : "+str(int(read_dht22()[1]))+ "%")
  print("Light Intensity : "+str(int(read_ldr()))+" LUX")
  print("Air Quality : "+str(int(read_mq135()))+" PPM")
  print("-------------------------------------------------------------------------------")
  data = {
    'temp' : str(int(read_dht22()[0])),
    'humi' : str(int(read_dht22()[1])),
    'lig_inten' : str(int(read_ldr())),
    'air_qual'  : str(int(read_mq135()))
  }
  push_(data)
  push(data)
time.sleep(10)
```

Description:

This project is a Python script designed to run on a microcontroller or embedded system. It collects environmental data from various sensors, including temperature, humidity, light intensity, and air quality, and sends this data to a GitHub repository. The code is suitable for monitoring and logging environmental conditions over time.

Components and Dependencies:

- Microcontroller (or embedded system)
- DHT22 Temperature and Humidity Sensor
- Light Intensity Sensor (LDR)
- Air Quality Sensor (MQ135)
- Python Libraries: **machine**, **dht**, **time**, **requests**, **json**, **base64**

Code Explanation:

1. Importing Modules: The code starts by importing necessary Python modules, including machine control, sensor interaction, and web requests.
2. Sensor Setup: The code initializes and configures the sensors, such as the DHT22 sensor, light intensity sensor (LDR), and air quality sensor (MQ135).

3. get_ist_datetime Function: This function makes an HTTP request to an API to obtain the current date and time in the Asia/Kolkata time zone, which is later used for timestamping data.
4. push Function: Responsible for updating a JSON file in a GitHub repository. It retrieves existing data, appends the current sensor data with a timestamp, and updates the repository with the new data.
5. push_ Function: Similar to the **push** function but specifically for updating a separate JSON file in the same GitHub repository with live data.
6. Sensor Reading Functions: These functions read and process data from the various sensors, such as light intensity, air quality, and temperature/humidity from the DHT22 sensor.
7. Main Loop: The core of the program. It continuously reads data from the sensors, prints the values, updates the GitHub repository with the collected data, and then repeats the process. The loop includes a 10-second delay between data readings.

Usage and Configuration

1. Ensure that the microcontroller is properly connected to the sensors and has access to the internet.
2. Update the following variables with your GitHub repository and access token details:
    - **repository_owner**: GitHub repository owner
    - **repository_name**: Name of the GitHub repository
    - **access_token**: GitHub personal access token
    - **file_path**: Path to the JSON file in the repository where data will be stored
3. Upload the script to your microcontroller and run it.

Expected Output:

The script will continuously print the environmental data (temperature, humidity, light intensity, air quality) to the console. It will also periodically update the specified GitHub repository with this data.

Repository Structure:

- The data is stored in two separate JSON files within the GitHub repository:
    - **Database/Database.json**: Historical environmental data with timestamps.
    - **Database/liveData.json**: Real-time data with the most recent timestamp.

Conclusion:

The Project almost completed except the Web Application. I will submit the Web application and complete the project in the next phase.

This is the URL of the Wokwi Project https://wokwi.com/projects/378982308199682049