Laboratório VISGRAF

Instituto de Matemática Pura e Aplicada

The Ecosystem IOT.WEB.AI for XR

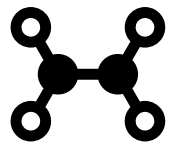
Matteo Moriconi
(collaborator Luiz Velho)

Technical Report TR-24-07 Relatório Técnico

August - 2024 - Agosto

The Ecosystem IOT.WEB.AI for XR

by Matteo Moriconi
(with collaboration of Luiz Velho)



Part I - Introduction 3

Sensor / Cloud / AI 3

Part II - Output 4

LED Display 4

Part III - Input 9

Eye 9

Mic 13

Part IV - I/O Sensor App 16

Weather Channel 16

Part V - Agents Applications 22

Generative Weather Person 22

Celestia Borealis 23

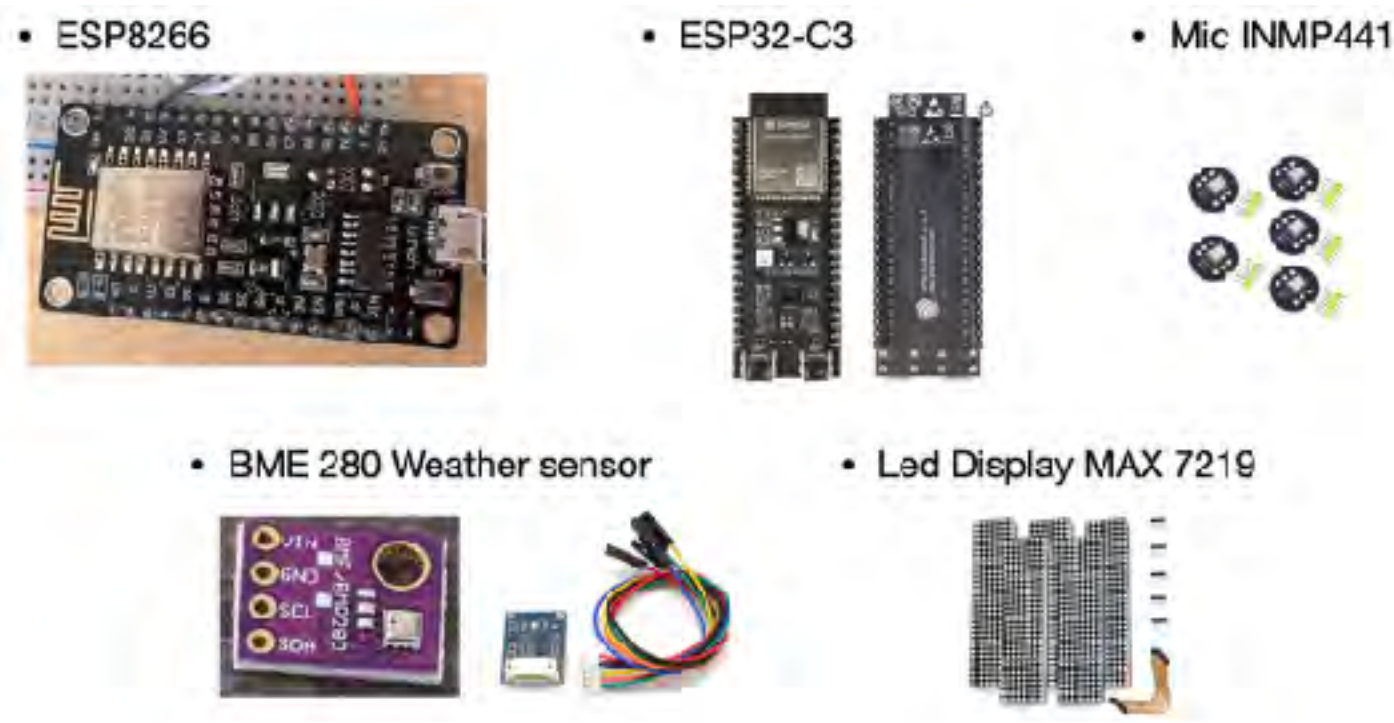
PART I - INTRODUCTION

SENSOR / CLOUD / AI

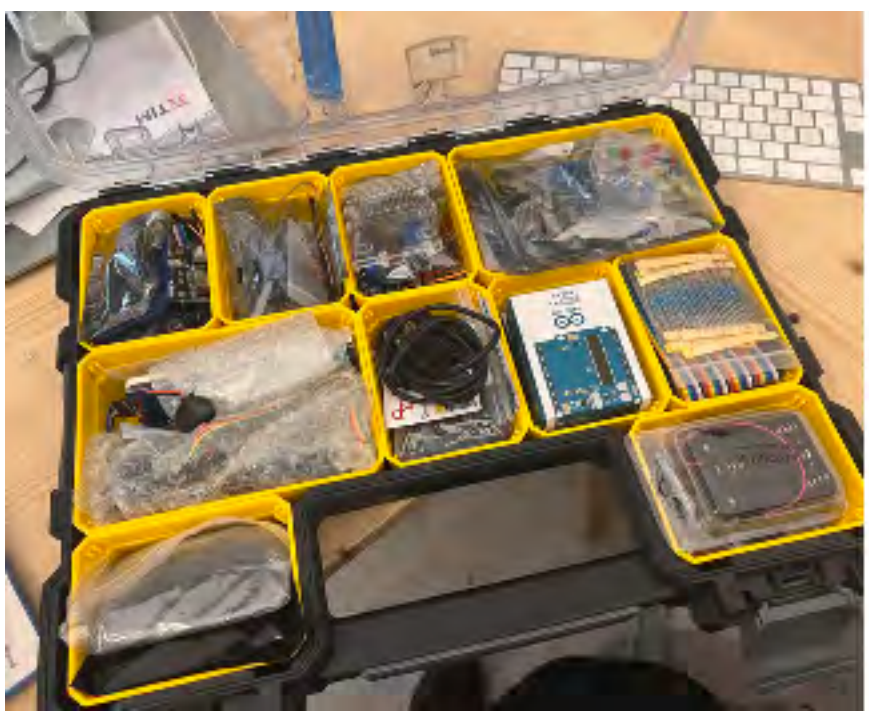
This Document is an update on the progress of the research into the convergence of AI, the Web, and IoT for XR.

Widgets:
- The Weather Channel
- The Mic
- The Eye
- The Led Display

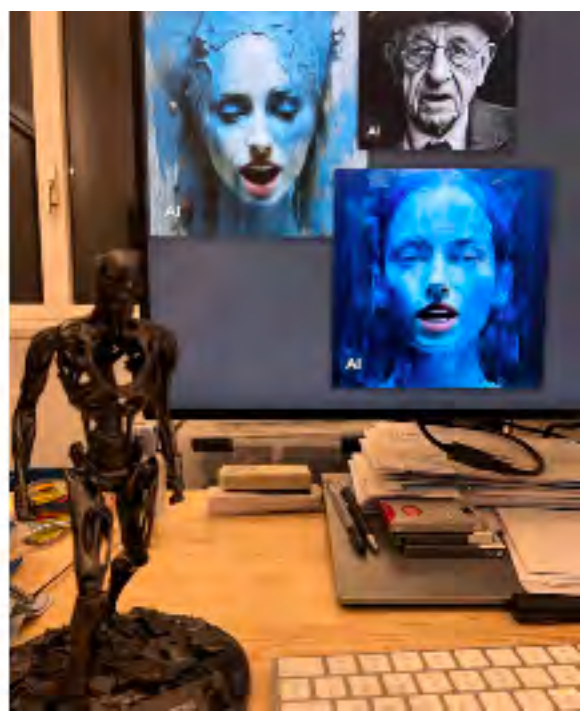
Concept:
- Virtual Assistant (Celestia Borealis)



IoT Components



Parts



Celestial Borealis

PART II - OUTPUT

The Output is done by a LED Display.

LED Display

LED DISPLAY

The LED Display is composed of an Arduino IoT to display text and an App to send the contents.

Arduino IoT

Matrix Display

ESP8266 Matrix Display Message Scroller

This Arduino sketch is designed for ESP8266-based modules to fetch and display messages and animations on a matrix LED display. It demonstrates connecting to a WiFi network, making HTTP requests, and dynamically updating the display content.

Features

- Connects to a specified WiFi network.
- Fetches text messages from a specified URL.
- Displays predefined animations and fetched text on a matrix LED display.
- Cycles through animations and messages with dynamic transitions.

Hardware Requirements

- ESP8266 module (e.g., NodeMCU, Wemos D1 Mini)
- Matrix LED display compatible with the MD_MAX72xx library
- Optional: External power source for the LED display

Software Requirements and Libraries

This sketch requires the Arduino IDE with support for the ESP8266 platform and the following libraries:

- `ESP8266WiFi` for WiFi connectivity.
- `ESP8266HTTPClient` for making HTTP requests.
- `WiFiClient` for managing WiFi connections.
- `MD_Parola` for controlling the matrix display.
- `MD_MAX72xx` for interfacing with the matrix hardware.
- `SPI` for Serial Peripheral Interface communication.

Installing Libraries

1. Open the Arduino IDE.
2. Go to **Sketch** > **Include Library** > **Manage Libraries**.
3. In the Library Manager, search for and install the following libraries:
   - `MD_Parola`
   - `MD_MAX72xx`
   - For `ESP8266WiFi` and `ESP8266HTTPClient`, these are included with the ESP8266 board package, so no additional installation is required.

ESP8266 Board Setup

Ensure you have the ESP8266 boards package installed in your Arduino IDE:

1. Go to **File** > **Preferences**.
2. In the Additional Board Manager URLs field, add the following URL: `http://arduino.esp8266.com/stable/package_esp8266com_index.json`
3. Go to **Tools** > **Board** > **Boards Manager**, search for ESP8266, and install the package.
4. Select your ESP8266 board from **Tools** > **Board**.

Configuration

Before uploading the sketch, ensure to:

- Update `ssid` and `pass` with your WiFi network's SSID and password.
- Adjust `HARDWARE_TYPE`, `MAX_DEVICES`, `CLK_PIN`, `DATA_PIN`, and `CS_PIN` according to your matrix display's specifications.

Uploading the Sketch

1. Connect your ESP8266 module to your computer.
2. Select the correct board and port in the Arduino IDE.
3. Upload the sketch.

Upon successful upload, the ESP8266 will connect to the specified WiFi network, fetch the message from the configured URL, and display the animations and messages on the matrix display.

Troubleshooting

- Ensure your WiFi network is operational and the SSID and password are correctly entered in the sketch.
- Verify the wiring between the ESP8266 and the matrix display.
- Check if the specified URL for fetching messages is accessible.

For further assistance, consult the documentation for the ESP8266 core for Arduino and the libraries used in this sketch.

ESP8266 Module

Matrix LED Display Wiring

ESP8266 to Matrix LED Display Wiring Guide

Components Required

- ESP8266 module (e.g., NodeMCU, Wemos D1 Mini)
- Matrix LED display (compatible with MD_MAX72xx, using MAX7219 chip)
- Jumper wires

Wiring Guide

Connect the ESP8266 GPIO pins to the matrix display as follows:

1. **CLK_PIN (GPIO 14)** to the CLK pin on the matrix display.
2. **DATA_PIN (GPIO 12)** to the DIN (Data In) pin on the matrix display.
3. **CS_PIN (GPIO 13)** to the CS (Chip Select or LOAD) pin on the matrix display.

Power Connections

- **VCC** of the display to a 5V output on the ESP8266 (VIN pin if available).
- **GND** of the display to a GND pin on the ESP8266.

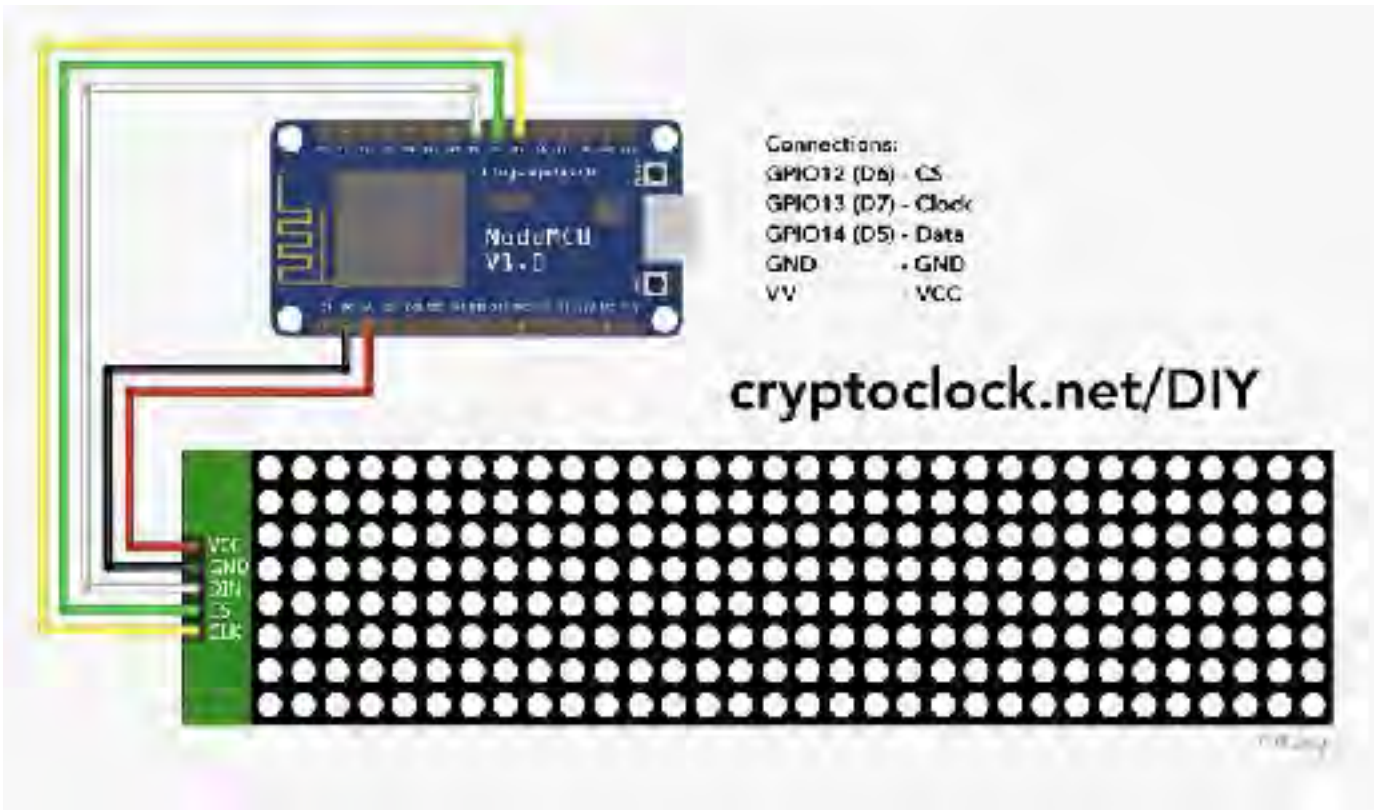
ASCII Diagram

ESP8266 Module   Matrix LED Display

```
+-----------------+                 +--------------------+
|                 |                 |                    |
|             5V -|---------------->| VCC                |
|                 |                 |                    |
|            GND -|---------------->| GND                |
|                 |                 |                    |
|        GPIO 14 -|---------------->| CLK                |
|                 |                 |                    |
|        GPIO 12 -|---------------->| DIN                |
|                 |                 |                    |
|        GPIO 13 -|---------------->| CS                 |
|                 |                 |                    |
+-----------------+                 +--------------------+
```

Additional Notes

- Ensure the matrix display is compatible with the ESP8266's voltage levels.
- If the display draws significant power, consider using an external 5V power source, connecting the GNDs together.



Wiring ESP8266 to LED Matrix Display

App

PHP Script

This PHP script performs several actions related to receiving, processing, and storing data, most likely from an ESP8266 device, as well as managing a simple file operation. Here's a step-by-step explanation of what this script does:

1. **Collects Data via POST Requests**: The script starts by collecting data sent to it via HTTP POST requests. It expects two pieces of data:
   - `$vmsg`: A variable that collects data from the POST request with the name `f_msg`. This data could be any form of message coming from the ESP8266 device.
   - `$MCU`: Another variable that collects data from the POST request with the name `f_mcu`. This could be an identifier or any other data related to the ESP8266 device.

2. **Preparation for Database Connection**: It sets up variables for connecting to a MySQL database. These include:
   - `$servername`: The hostname of the database server, which is "localhost" in this case, indicating the database is hosted on the same server as the script.
   - `$username`: The username for the database login, "metamessag".
   - `$password`: The password for the database login.
   - `$dbname`: The name of the database, "metamessag".

3. **Database Connection**: The script then attempts to connect to the MySQL database using the `mysqli` PHP extension, creating a new instance of the `mysqli` class with the database connection details.

4. **Error Handling for Database Connection**: It checks if the connection was successful. If there's an error (`connect_errno` is true), the script dies and prints an error message using `connect_error`.

5. **Database Insert Operation**: It constructs an SQL query to insert the message received from the ESP8266 device (`$vmsg`) into a table named `frontend_data`. The field being inserted into is named `formulario`.

6. **Execution and Feedback for Database Operation**: The script executes the SQL query. If the insertion is successful, it prints a confirmation message ("Aggiornamento OK") along with two line breaks. If there's an error, it prints an error message detailing why the record update failed.

7. **File Operation**: After interacting with the database, the script opens (or creates if it doesn't exist) a file named "codec.txt" in write-binary mode (`"wb"`). It then writes the string contained in `$vmsg` into this file. After writing, it closes the file. This operation effectively saves the last message received from the ESP8266 device into a text file, overwriting any existing content.

8. **Final Output**: Lastly, the script echoes the message received from the ESP8266 device (`$vmsg`), prefixed by the identifier or data (`$MCU`) followed by a colon and a space.

In summary, this PHP script is designed to receive messages and identifiers from an ESP8266 device via POST requests, insert these messages into a MySQL database, write the message to a text file, and provide feedback about these operations through the web server's response.

HTML File

This HTML file creates a simple web page designed to send messages to an ESP (presumably an ESP8266 or similar microcontroller) and a database through a backend script named `backend-spx.php`. Here's a breakdown of the file's structure and functionality:

- **`<head>` Section**: Contains metadata about the document, including its title set to "Messaggero," which is displayed on the browser tab.

- **`<body>` Section**: Defines the content of the web page.

  - **Heading (`<h2>`)**: Displays a heading "Invia messaggio ad un ESP" which translates to "Send a message to an ESP" in English, indicating the purpose of the web page.
  - **Form (`<form>`)**: Contains a form element that users can interact with to send a message. The form's attributes and elements include:
    - `action="backend-spx.php"`: Specifies that the form data should be sent to a backend script named `backend-spx.php` for processing when the form is submitted.
    - `method="post"`: Indicates that the form data will be sent through the POST method, which is a secure way of transmitting data as it doesn't expose the data in the URL.

    - **Message Input (`<input type="text">`)**: A text input field named `f_msg` where users can enter a message. The field is pre-populated with a "?" and allows text up to 200 characters long. This is the message that will be sent to the ESP8266 and stored in the database.

    - **Hidden Field (`<input type="hidden">`)**: An invisible field with the name `f_mcu` and a value of "0". This could be used to send fixed or session-specific information to the backend script, like an identifier for the ESP device or a user session ID, without displaying it to the user.

    - **Submit Button (`<input type="submit">`)**: A button that, when clicked, sends the form data to the specified backend script for processing.

  - **Paragraph (`<p>`)**: Provides additional information to the user, stating "Il messaggio sara inviato ad um ESP e Banco Dati." This translates to "The message will be sent to an ESP8266 and Database," clarifying the action that will occur upon form submission.

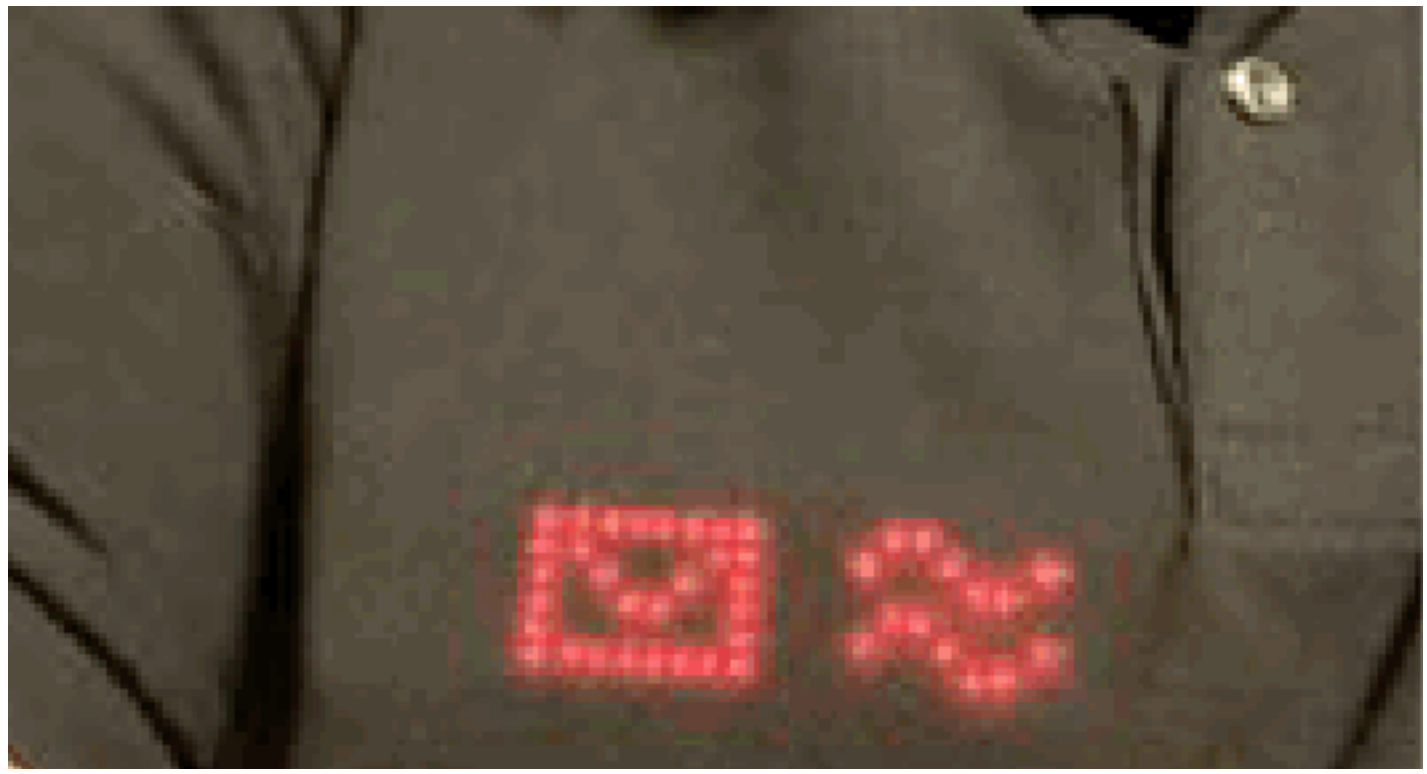
This HTML page serves as a user interface for sending messages to an ESP8266 device and a database through a backend PHP script. The process involves user interaction with the form, entering a message, and then submitting it, which triggers the form's action to send the data to `backend-spx.php` for further processing.

System Overview

- NodeMCU ESP8266 ESP-12F WiFi
- Led Display: MAX7219

IoT Setup

Web App to Send a Text



Wearable LED Usage

PART III - INPUT

The input is done by a camera (The Eye) and a microphone (The Mic).

EYE

The communication of the Eye IoT is done with a packet-based scheme.

Raspberry Pi Camera

The camera is attached to a Raspberry Pi computer.

Raspberry Pi Camera

Camera Configuration

Requirements

1) To enable the camera on a Raspberry Pi in the BIOS, you typically don't need to do anything directly within the BIOS settings. Instead, you'll handle this through the Raspberry Pi's operating system configuration.

Here's how you can do it:

Ensure the Camera Module is Connected: First, ensure that the camera module is connected correctly to the Raspberry Pi's camera port.

Enable the Camera Interface: By default, the camera interface is disabled in the Raspberry Pi's configuration. You need to enable it. Here's how:

a. Open a terminal on your Raspberry Pi or SSH into it.

b. Run sudo raspi-config.

c. Navigate to Interfacing Options.

d. Select Camera.

e. Choose Yes to enable the camera interface.

f. Reboot your Raspberry Pi for the changes to take effect.

2) Install Camera Software: Depending on your needs, you might need to install additional software to interact with the camera. For instance, if you're planning to use Python, you may want to install the picamera module.

```
sudo apt-get update
sudo apt-get install python3-picamera
```

3) Test the Camera: After enabling the camera interface and installing any necessary software, you can test if the camera is working properly. You can use the raspistill command to take a still image or raspivid to capture video.

For example, to take a picture:

bash

raspistill -o test.jpg

3) On terminal on your Raspberry Pi or SSH into it.

Run the following command to install paramiko:

pip install paramiko

sudo apt update
sudo apt install python3-pip
pip3 install paramiko

4) Azure SDK
pip install azure-cognitiveservices-speech

5) install fswebcam and its dependencies on your Raspberry Pi.
sudo apt update
sudo apt install fswebcam

Ready to run the c-takephoto-button-trigger-withqueue-upload-v1.py

---

Python Script

takephoto-button-trigger-withqueue-upload

```
# Takephoto Script

import os
import paramiko
import time
import requests
import azure.cognitiveservices.speech as speechsdk
import xml.etree.ElementTree as ET
from gpiozero import Button
from queue import Queue
import threading

# Define a global queue for image processing
image_queue = Queue()

# Function to take a picture, save it with a unique name, and return the image name
def take_picture():
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    image_name = f"image_{timestamp}.jpg"
    os.system(f"fswebcam -r 1280x720 --no-banner {image_name}")
    return image_name

# Function to upload the image to the SFTP server
def upload_to_sftp(local_file, remote_path, hostname, username, password):
    transport = paramiko.Transport((hostname, 22))
    transport.connect(username=username, password=password)
    sftp = paramiko.SFTPClient.from_transport(transport)
    sftp.put(local_file, os.path.join(remote_path, os.path.basename(local_file)))
    sftp.close()
    transport.close()

# Function to generate the image URL
def generate_image_url(image_name):
    return f"https://metamessage.id34.com/TheEye/uploads/{image_name}"

# Function to append image information to an XML file
def append_to_xml(image_name, description, timestamp, xml_file):
    root = None
    if os.path.exists(xml_file):
        tree = ET.parse(xml_file)
        root = tree.getroot()
    else:
```

```
        root = ET.Element("Images")

    image_elem = ET.SubElement(root, "Image")
    image_elem.set("name", image_name)
    image_elem.set("description", description)
    image_elem.set("timestamp", timestamp)

    tree = ET.ElementTree(root)
    tree.write(xml_file)

# Function to convert text to speech
def text_to_speech(text, key, region):
    speech_config = speechsdk.SpeechConfig(subscription=key, region=region)
    speech_synthesizer = speechsdk.SpeechSynthesizer(speech_config=speech_config)

    result = speech_synthesizer.speak_text_async(text).get()

    if result.reason == speechsdk.ResultReason.SynthesizingAudioCompleted:
        print("Speech synthesized to speaker for text [{}]".format(text))
    elif result.reason == speechsdk.ResultReason.Canceled:
        cancellation_details = result.cancellation_details
        print("Speech synthesis canceled: {}".format(cancellation_details.reason))
        if cancellation_details.reason == speechsdk.CancellationReason.Error:
            print("Error details: {}".format(cancellation_details.error_details))

# Function to process images from the queue
def process_images():
    while True:
        if not image_queue.empty():
            image_name = image_queue.get()
            upload_to_sftp(image_name, sftp_remote_path, sftp_host, sftp_username, sftp_password)
            image_url = generate_image_url(image_name)

            # Fetch image description using OpenAI's API and read it out using Azure's text-to-speech service
            api_key = os.getenv('OPENAI_API_KEY')
            azure_api_key = os.getenv('AZURE_SPEECH_KEY')
            azure_region = os.getenv('AZURE_REGION')

payload = {
    "model": "gpt-4-vision-preview",
    "messages": [
        {
            "role": "system",
            "content": [{
                "type": "text",
                "text": "You are a cool image analyst. Your goal is to describe what is in this image."
            }],
        },
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "What is in the image?"
                },
                {
                    "type": "image_url",
                    "image_url": {
                        "url": image_url
                    }
                }
            ]
        }
    ],
    "max_tokens": 500
}

headers = {
    "Authorization": f"Bearer {api_key}",
    "Content-Type": "application/json"
}

response = requests.post('https://api.openai.com/v1/chat/completions', headers=headers, json=payload)
r = response.json()
if "choices" in r and len(r["choices"]) > 0:
    description = r["choices"][0]["message"]["content"]
```

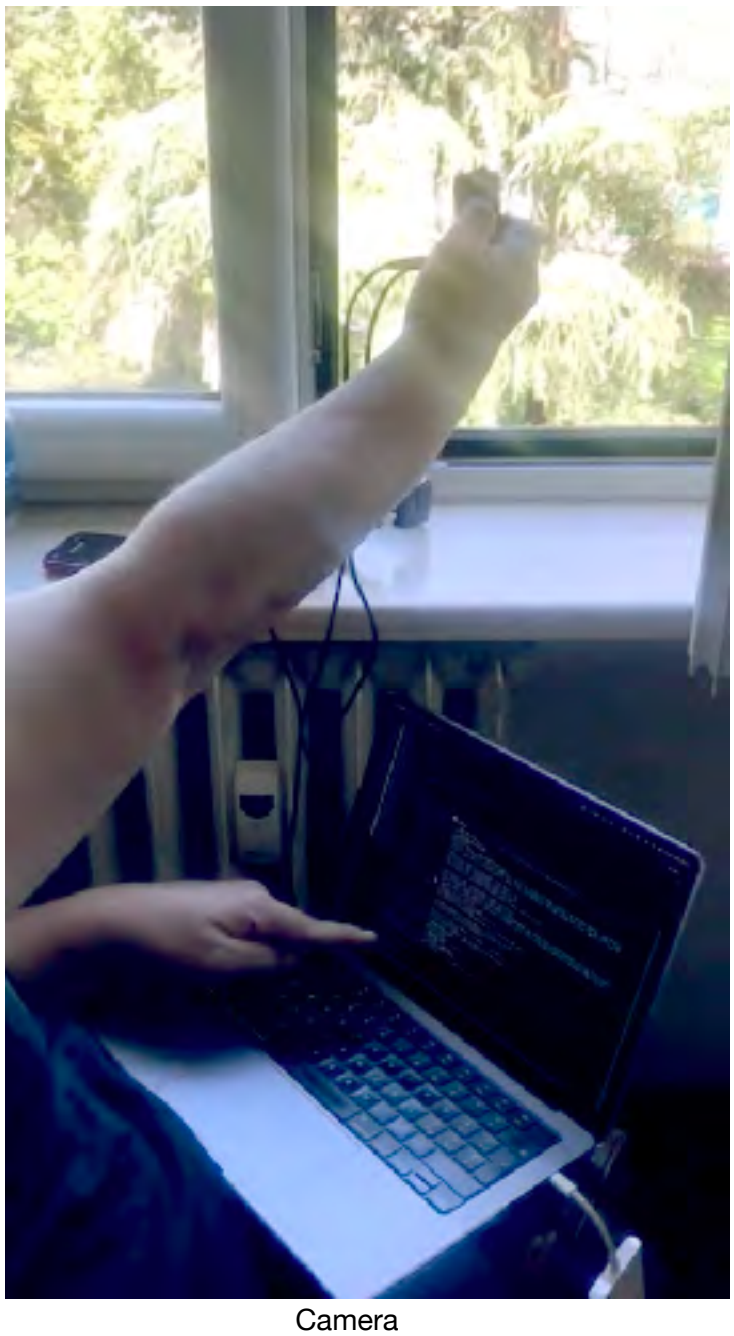
```
            print("Description:", description)
            text_to_speech(description, azure_api_key, azure_region)
        else:
            print("No response or unexpected format received.")

        timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
        append_to_xml(image_name, description, timestamp, xml_file)

# Main function
if __name__ == "__main__":
    sftp_host = "metame-16624.bolt70.servebolt.com"
    sftp_username = "metame_16624"
    sftp_password = "g&ru<.9f>~F*<3]J"
    sftp_remote_path = "/kunder/yahooc_10911/metame_16624/public/TheEye/uploads"
    xml_file = "image_data.xml"
    button = Button(17)  # Assuming the button is connected to GPIO17

    # Start a thread for image processing
    image_processing_thread = threading.Thread(target=process_images)
    image_processing_thread.start()

    while True:
        button.wait_for_press()  # Wait for the button to be pressed
        image_name = take_picture()
        image_queue.put(image_name)  # Add the image to the queue for processing
```



Camera

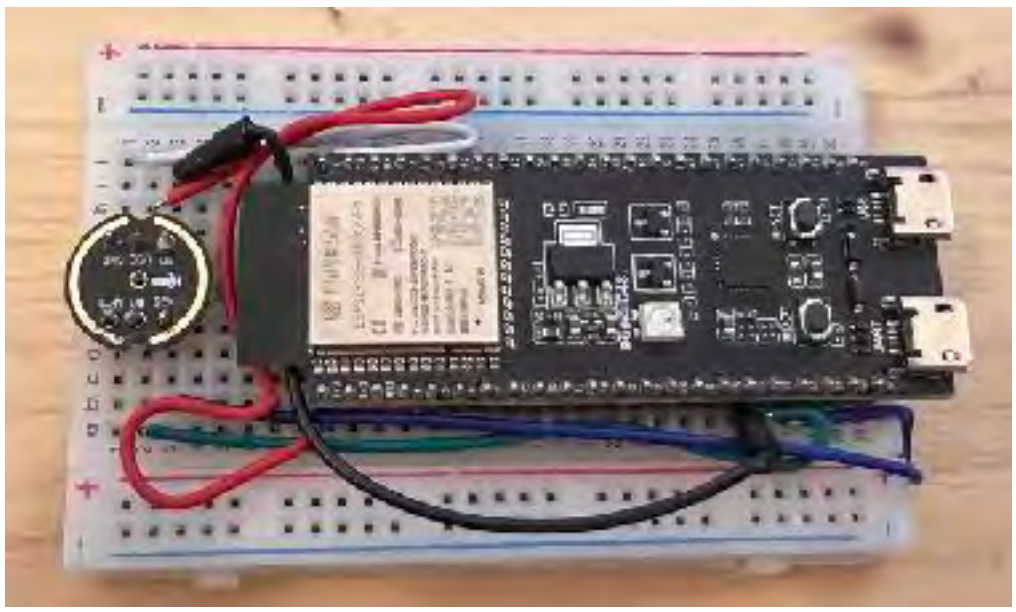
Image

Taking a Picture of a Water Bottle

MIC

The Mic is an IoT based on Arduino. It uses an ESP32-S3.

The communication of the Mic is done with streaming.



ESP32-S3 and Microphone

Arduino Microphone

Installation Guide

To install the ESP32-S3 board support package in the Arduino IDE, you will need to follow a series of steps to add the ESP32 boards, including the ESP32-S3, to the Arduino Board Manager. This allows you to develop and upload programs to the ESP32-S3 using the Arduino development environment. Here's an outline of the process:

1. Open Arduino IDE
Ensure you have the latest version of the Arduino IDE installed on your computer. If not, download and install it from the [official Arduino website](https://www.arduino.cc/en/Main/Software).

2. Access Preferences
- In the Arduino IDE, go to **File > Preferences** (on Windows/Linux) or **Arduino > Preferences** (on macOS).

3. Add ESP32 Board Manager URL
- In the Preferences window, find the "Additional Board Manager URLs" field.
- Add the following URL to the field: `https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`
  - If there are already URLs in the field, separate them with commas.
- Click "OK" to close the Preferences window.

4. Open Board Manager
- Navigate to **Tools > Board > Boards Manager...** from the top menu.
- In the Boards Manager window, type "ESP32" in the search bar.

5. Install ESP32 Package
- Find the "ESP32 by Espressif Systems" package in the search results.
- Click the "Install" button to install the package.
- Once installation is complete, click "Close" to exit the Boards Manager.

6. Select Your ESP32-S3 Board
- Go to **Tools > Board** and scroll down to the ESP32 section. You should now see a list of ESP32 boards.
- Select the specific ESP32-S3 model you are using (e.g., "ESP32S3 Dev Module" or a similar entry). If you're unsure, "ESP32S3 Dev Module" is a good starting point for generic ESP32-S3 development boards.

7. Configure Board Settings (Optional)
Depending on your project, you might need to adjust settings under **Tools** like the upload speed, partition scheme, or other options specific to the ESP32-S3. These settings vary based on your particular project and board.

8. Write Your Program
Now, you can start writing your program in the Arduino IDE or load one of the examples specifically tailored for the ESP32-S3 to test your setup.

9. Connect Your ESP32-S3
Using a USB cable, connect your ESP32-S3 board to your computer.

10. Select Port
- Go to **Tools > Port** and select the COM port your ESP32-S3 is connected to. (On macOS, it will be listed as a `/dev/cu` device.)

11. Upload Your Program
- Click the "Upload" button (right arrow icon) in the Arduino IDE to compile and upload your program to the ESP32-S3 board.

If everything is set up correctly, your Arduino IDE is now configured to develop and upload sketches to the ESP32-S3 board. If you encounter issues, make sure your USB drivers are correctly installed, and the board is properly connected to your computer.

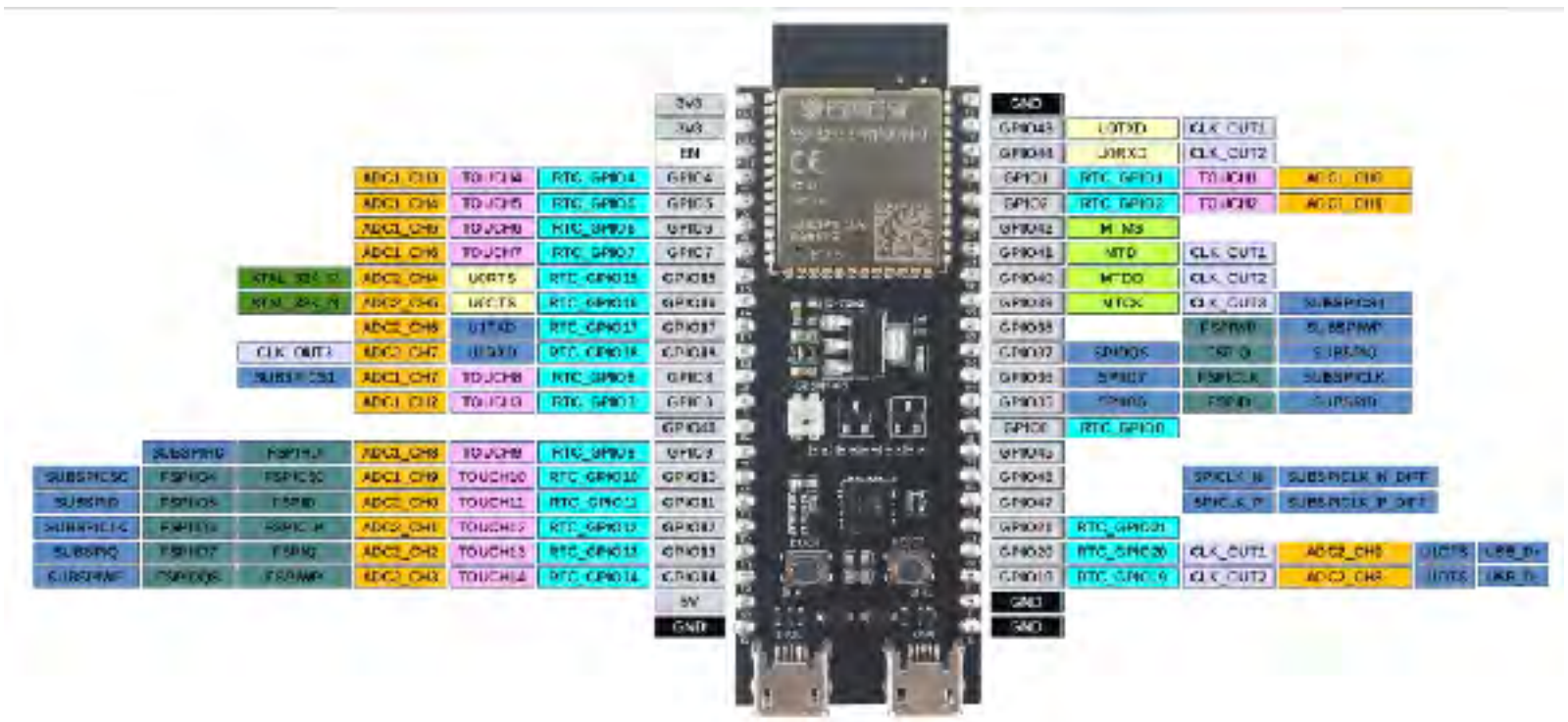
Wiring

```
ESP32-S3-DevKitC-1                Breadboard                    INMP441 Microphone
+------------------+              +---------+                   +---------------+
|                  |              |         |                   |               |
|              3.3V|--------------|+      VDD|-------------------|VDD            |
|                  |              |         |                   |               |
|              GND |--------------|-      GND|-------------------|GND            |
|                  |              |         |                   |               |
|           GPIO10 |--------------|         |-------------------|SD             |
|                  |              |         |                   |               |
|           GPIO11 |--------------|         |-------------------|WS             |
|                  |              |         |                   |               |
|           GPIO12 |--------------|         |-------------------|SCK            |
|                  |              |         |                   |               |
+------------------+              +---------+                   +---------------+
```

an ASCII diagram to represent the connection between an ESP32-S3-DevKitC-1, an INMP441 microphone, and a breadboard involves simplifying the components and their connections into text representation. Here's a simplified ASCII diagram illustrating the setup based on your wiring requirements:

- `3.3V` and `GND` from the ESP32-S3 are connected to the power (+) and ground (-) rails on the breadboard, respectively, and then to the `VDD` and `GND` on the INMP441 to power the microphone.
- `GPIO10` (I2S_SD) from the ESP32-S3 is connected to the `SD` (Serial Data) pin on the INMP441 for audio data transmission.
- `GPIO11` (I2S_WS) from the ESP32-S3 is connected to the `WS` (Word Select) pin on the INMP441 to toggle between left and right channels (if applicable).
- `GPIO12` (I2S_SCK) from the ESP32-S3 is connected to the `SCK` (Serial Clock) pin on the INMP441 for clock synchronization.

This diagram is a simplified representation and focuses on the primary connections necessary for audio data transmission. The actual connections on a breadboard and component layout might differ based on the specific models and additional components in your setup.



ESP32-S3 Pin Layout

Arduino Script

This script is designed for an Arduino environment and incorporates various functionalities, including Wi-Fi connectivity, WebSocket communication, and audio input through an I2S interface. It's targeted for platforms like the ESP32 that support these features natively. Here's a breakdown of how the script works:

Header Files
- `#include <driver/i2s.h>`: Includes the I2S driver library for handling audio input.
- `#include <WiFi.h>`: Includes the Wi-Fi library to enable Wi-Fi functionality.
- `#include <ArduinoWebsockets.h>`: Incorporates the WebSocket library for real-time communication over a WebSocket protocol.

Definitions and Global Variables
- **I2S Pins**: Defines the pins for I2S communication, essential for audio input.
- **Buffer Configuration**: Defines the size of the audio buffer for processing audio data.
- **Wi-Fi Credentials**: Specifies the SSID and password for connecting to a Wi-Fi network.
- **WebSocket Server**: Details of the WebSocket server, including the host address and port.
- **WebSocket Client**: An instance of `WebsocketsClient` is created for handling WebSocket communication.
- **Audio Buffer**: An integer array `sBuffer` is defined to store audio data.

Event Callback Function
`onEventsCallback`: A function designed to handle events from the WebSocket, such as opening or closing connections, and receiving pings or pongs.

I2S Configuration Functions
- `i2s_install()`: Configures the I2S driver with parameters like mode, sample rate, bits per sample, and DMA buffer settings.
- `i2s_setpin()`: Configures the physical pins to be used by the I2S interface for audio input.

Main Functions
- `setup()`: Initializes serial communication, connects to Wi-Fi, connects to the WebSocket server, and creates a task for microphone input processing.
- `loop()`: An empty loop function, as the main functionality is event-driven and handled by tasks and callbacks.

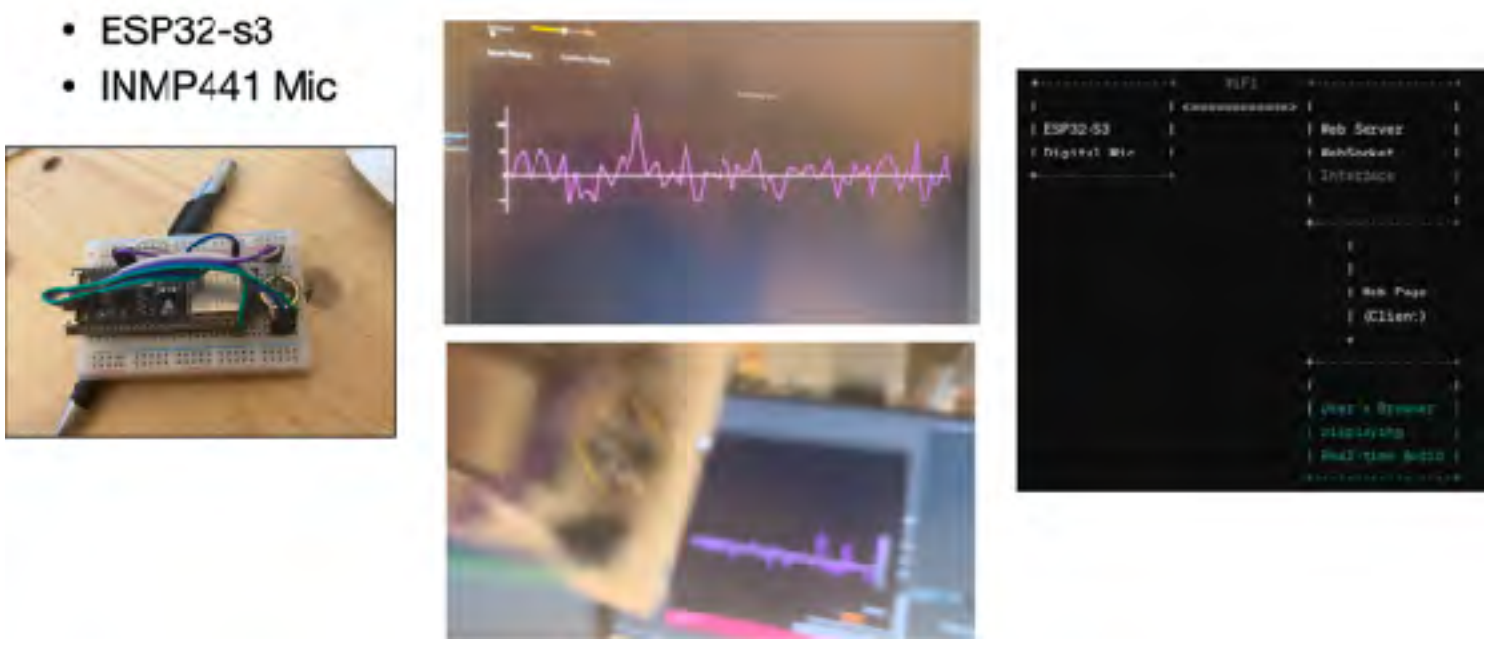
Wi-Fi and WebSocket Connection Functions
- `connectWiFi()`: Attempts to connect to the specified Wi-Fi network, retrying until successful.
- `connectWSServer()`: Connects to the WebSocket server, retrying until a connection is established.

Microphone Task Function
`micTask(void* parameter)`: A function designed to run as a separate task. It initializes the I2S interface, starts reading audio data into `sBuffer`, and sends this data to the WebSocket server if a connection is established.

Process Flow
1. **Initialization**: The `setup()` function sets up serial communication, connects to Wi-Fi, and establishes a WebSocket connection. It also starts a dedicated task for handling microphone input.
2. **Wi-Fi Connection**: `connectWiFi()` continuously attempts to connect to the specified Wi-Fi network.
3. **WebSocket Connection**: `connectWSServer()` tries to connect to the WebSocket server using the provided host and port information.
4. **Audio Processing Task**: Once both connections are established, the `micTask` continuously reads audio data from the I2S interface and sends it to the WebSocket server when available and connected.

This script effectively demonstrates integrating hardware-level audio input with network communication in an IoT or embedded system context, utilizing the ESP32's capabilities for real-time audio data transmission over the internet.

System Overview



PART IV - I/O SENSOR APP

WEATHER CHANNEL

This I/O Sensor Application is a proof-of-concept project that implements a Weather Channel. It works as a back-end for the Display Apps.

I/O Sensor Application

The Weather Channel Application is based on Arduino and a Web backend. It is based on a synchronous and asynchronous communication.

Arduino Environment Data Sensor

ESP8266 Installation

Install ESP8266 Add-on in Arduino IDE

To use the ESP8266 module with the Arduino IDE, you must first install the ESP8266 add-on. This add-on enables the Arduino IDE to compile and upload code to the ESP8266 module. Follow the steps below to install the ESP8266 add-on:

1. Open the Arduino IDE on your computer.
(download here: https://www.arduino.cc/en/software)

2. Go to File > Preferences (Arduino > Preferences on Mac) to open the Preferences window.

3. In the "Additional Board Manager URLs" field, enter the following URL:
   http://arduino.esp8266.com/stable/package_esp8266com_index.json
   If there are already URLs in this field, separate them with commas.

4. Click "OK" to close the Preferences window.

5. Go to Tools > Board: "xxx" > Boards Manager...

6. In the Boards Manager, type "ESP8266" into the search bar. (GENERIC ESP8266)

7. Find the "ESP8266 by ESP8266 Community" entry and click the "Install" button next to it.

8. Once the installation is complete, you can select your ESP8266 board under Tools > Board: "xxx" (you'll find it under the "ESP8266 Boards" section).

9. Make sure to select the correct board and its settings according to your specific module.

By completing these steps, you have successfully installed the ESP8266 add-on in the Arduino IDE. You can now proceed to write and upload programs to your ESP8266 module.

Wiring

BME280 wiring to ESP8266

The ESP8266 I2C pins are:

   GPIO 5 (D1): SCL (SCK)
   GPIO 4 (D2): SDA (SDI)

Assemble your circuit as in the next schematic diagram if you’re using an ESP8266 board:

IMAGE REFERENCE (ARDUINO-IDE-CODE-ESP8266/READMEFIRST):
ESP8266-BME280-Arduino-IDE.jpeg

Libraries for Arduino Projects

Installing Libraries for Arduino Projects

This guide will help you install two essential libraries for working with the BME280 sensor on Arduino: the BME280 library and the Adafruit_Sensor library.

1. Installing the BME280 Library:
   - Open your Arduino IDE.
   - Go to Sketch > Include Library > Manage Libraries. The Library Manager should open.
   - In the search box, type "adafruit bme280" and press enter.
   - Find the library in the search results and click the "Install" button to install the library.

2. Installing the Adafruit_Sensor Library:
   - To use the BME280 library effectively, you also need the Adafruit_Sensor library.
   - Again, go to Sketch > Include Library > Manage Libraries in your Arduino IDE.
   - This time, in the search box, type "Adafruit Unified Sensor" and press enter.
   - Scroll through the results to find the Adafruit_Sensor library, then click the "Install" button to add it to your IDE.

After completing these steps, you'll have both the BME280 and Adafruit_Sensor libraries installed in your Arduino IDE, ready for use in your projects.

Arduino IDE Libraries Installation

How to Install Required Arduino IDE Libraries

Step 1: Setting up the ESP8266 Board in Arduino IDE
1. Open the Arduino IDE.
2. Go to File > Preferences (on Windows) or Arduino > Preferences (on macOS).
3. In the "Additional Board Manager URLs" field, add the following URL: http://arduino.esp8266.com/stable/package_esp8266com_index.json. If there are already URLs present, separate them with commas.
4. Click "OK" to close the Preferences window.
5. Go to Tools > Board > Boards Manager...
6. Type "ESP8266" in the search bar and press Enter.
7. Find the "ESP8266" by "ESP8266 Community" and click the "Install" button.
8. After installation, select your ESP8266 board from Tools > Board > ESP8266 Boards.

Step 2: Installing Libraries

For ESP8266WiFi, ESP8266HTTPClient, WiFiClientSecureBearSSL:
These libraries come pre-installed with the ESP8266 board package, so no additional steps are required for their installation.

For Wire (I2C communication library):
The Wire library is included with the Arduino IDE, so there's no need for a separate installation.

For Adafruit_Sensor and Adafruit_BME280:
1. Go to Sketch > Include Library > Manage Libraries...
2. In the Library Manager, type "Adafruit Unified Sensor" in the search bar.
3. Find the "Adafruit Unified Sensor" library in the list and click the "Install" button next to it.
4. Next, search for "Adafruit BME280" in the Library Manager.
5. Find the "Adafruit BME280" library and click the "Install" button.

Step 3: Include the Libraries in Your Sketch
After installing the required libraries, you can include them in your Arduino sketch as follows:

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClientSecureBearSSL.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
```

Step 4: Verify Installation
To verify that the libraries are correctly installed, try compiling an example sketch that uses these libraries. You can find example sketches for the BME280 sensor in the Arduino IDE under File > Examples > Adafruit BME280 Library.

Code Overview

This code is designed for an ESP8266 microcontroller to collect environmental data using a BME280 sensor and send this data to a web server using HTTP POST requests. Here's a detailed breakdown:

1. Libraries:
- ESP8266WiFi: Facilitates WiFi connectivity on the ESP8266.
```
#include <ESP8266WiFi.h>
```

- ESP8266HTTPClient: Enables making HTTP requests.
```
#include <ESP8266HTTPClient.h>
```

- WiFiClientSecureBearSSL: Allows secure SSL/TLS connections, necessary for HTTPS communication.
```
#include <WiFiClientSecureBearSSL.h>
```

- Wire: Used for I2C communication with devices like the BME280 sensor.
#include <Wire.h>

- Adafruit_Sensor and Adafruit_BME280: Libraries for interfacing with the BME280 sensor.
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

2. WiFi Credentials:
Defines the SSID and password required for connecting to a WiFi network.
const char* ssid   = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

3. Server Information:
Specifies the URL for the web server where sensor data is POSTed.
const char* serverName = "https://yourserver.com/data.php";

4. API Key and Sensor Information:
An API key and sensor details are defined for sending with the sensor readings.
String apiKeyValue = "your_api_key";
String sensorName = "BME280";
String sensorLocation = "Office";

5. Sensor Setup:
The BME280 sensor is initialized for I2C communication.
Adafruit_BME280 bme;

6. setup() Function:
Initializes serial communication, connects to WiFi, and initializes the BME280 sensor.
```
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  if (!bme.begin()) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}
```

7. loop() Function:
Checks WiFi connection, prepares data, and sends an HTTP POST request to the server.
```
void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient https;
    https.begin(serverName);
    String httpRequestData = "api_key=" + apiKeyValue + "&sensor=" + sensorName + "&location=" + sensorLocation + "&value1=" +
String(bme.readTemperature());
    int httpResponseCode = https.POST(httpRequestData);
    https.end();
  }
  delay(30000);
}
```

This code is useful for IoT projects that require environmental monitoring with data sent remotely for analysis or display.

Weather Channel App

The Weather Channel App uses i) a PHP script to place HTTP post request into an SQL database; ii) a PHP script to interact with the database ; and an HTML code to create the Web interface.

Sensor Post Data

The provided PHP script, `esp-post-data.php`, is designed to handle HTTP POST requests for inserting sensor data into a database. Here's a step-by-step explanation of its components and functionality:

1. **Including Database Connection File**: The `include_once('esp-database.php');` statement includes the `esp-database.php` file once during the script execution. This file is expected to contain functions for database operations, such as connecting to the database and inserting data.

```php
include_once('esp-database.php');
```

2. **API Key Validation**: The script defines a hardcoded API key (`$api_key_value = "tPmAT5Ab3j7F9";`). This key is intended to secure the endpoint by ensuring that only requests containing the correct API key can insert data into the database.

```php
$api_key_value = "tPmAT5Ab3j7F9";
```

3. **Variable Initialization**: Variables for the API key, sensor identifier, location, and three sensor values (`$value1`, `$value2`, `$value3`) are initialized as empty strings. These variables will later store the sanitized input from the POST request.

```php
$api_key= $sensor = $location = $value1 = $value2 = $value3 = "";
```

4. **Handling POST Requests**: The script checks if the request method is POST using `if ($_SERVER["REQUEST_METHOD"] == "POST")`. If it is not a POST request, the script responds with "No data posted with HTTP POST."

```php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Process POST request
}
else {
    echo "No data posted with HTTP POST.";
}
```

5. **Data Sanitization**: Input data received through the POST request is sanitized by the `test_input()` function. This function trims whitespace from the beginning and end of the input, removes backslashes, and converts special characters to HTML entities to prevent XSS (Cross-Site Scripting) attacks.

```php
function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

6. **API Key Verification**: The provided API key in the POST request is compared with the hardcoded API key value. If they match, the script proceeds to process the data; otherwise, it responds with "Wrong API Key provided."

7. **Data Processing**: Upon successful API key verification, the script extracts and sanitizes the sensor identifier, location, and sensor values from the POST data. It then calls the `insertReading()` function (expected to be defined in `esp-database.php`) with these values as arguments to insert the sensor data into the database.

8. **Database Insertion Result**: The result of the database insertion operation (`$result`) is echoed back to the client. This result could be a success message, error message, or any other feedback generated by the `insertReading()` function.

9. **Response for Unmatched API Key or GET Request**: If the API key does not match or if the request method is not POST, the script provides an appropriate error message.

This script is tailored for IoT (Internet of Things) applications where sensor data is collected and transmitted to a server. The use of an API key adds a layer of security, albeit a very basic one, by ensuring that only authorized sources can submit data. For enhanced security, consider using more robust authentication mechanisms and HTTPS to protect data in transit.

Database Connection

This PHP script is designed to interact with a database to perform operations related to sensor data. It demonstrates how to insert new sensor readings into the database, retrieve all readings, get the last reading, and calculate minimum, maximum, and average readings over a specified number of recent entries. Below is a breakdown of its components:

1. Database Connection Variables:
The script starts by defining variables for the database connection - server name, database name, username, and password.

Example:
```
    $servername = "localhost";
    $dbname = "espdata";
    $username = "espdata";
    $password = "password";
```
2. insertReading Function:
This function inserts a new reading into the 'SensorData' table. It takes sensor details and values as parameters, establishes a database connection, and executes an INSERT SQL query.

Example:
```
    function insertReading($sensor, $location, $value1, $value2, $value3) {
        ...
        $sql = "INSERT INTO SensorData (sensor, location, value1, value2, value3)
        VALUES ('" . $sensor . "', '" . $location . "', '" . $value1 . "', '" . $value2 . "', '" . $value3 . "')";
        ...
    }
```

3. getAllReadings Function:
Retrieves a specified number of recent sensor readings from the database, ordering them by reading time in descending order.

Example:
```
    function getAllReadings($limit) {
        ...
        $sql = "SELECT id, sensor, location, value1, value2, value3, reading_time FROM SensorData order by reading_time desc limit " .
$limit;
        ...
    }
```
4. getLastReadings Function:
Fetches the most recent sensor reading from the database.

Example:
```
    function getLastReadings() {
        ...
        $sql = "SELECT id, sensor, location, value1, value2, value3, reading_time FROM SensorData order by reading_time desc limit 1";
        ...
    }
```
5. minReading, maxReading, and avgReading Functions:
These functions calculate the minimum, maximum, and average values of a specified column over a limited number of recent readings. They demonstrate the use of SQL aggregate functions MIN, MAX, and AVG within subqueries.

Example:
```
    function minReading($limit, $value) {
        ...
        $sql = "SELECT MIN(" . $value . ") AS min_amount FROM (SELECT " . $value . " FROM SensorData order by reading_time desc
limit " . $limit . ") AS min";
        ...
    }
```
Each function ensures the database connection is opened before executing a query and closed afterward. They use global variables for database connection details and handle potential connection errors by terminating the script with an error message.

Web-Based Interface

The provided PHP and HTML code implements a web-based interface for displaying weather data collected by an ESP (Espressif Systems) device, such as temperature and humidity readings. The code is structured into several key components:

1. **PHP Backend Logic**: The PHP code at the beginning includes the 'esp-database.php' file, which presumably contains functions for interacting with a database that stores the readings from the ESP device. It handles user input through the GET request parameter 'readingsCount', sanitizes this input, and sets a default value if none is provided.

Example:
```
if (isset($_GET["readingsCount"])){
  ...
} else {
  $readings_count = 20;
}
```
2. **Fetching Data**: The script retrieves the last reading from the database and computes the minimum, maximum, and average readings for both temperature (`value1`) and humidity (`value2`) based on the specified number of recent readings.

Example:
```
$last_reading = getLastReadings();
...
$min_temp = minReading($readings_count, 'value1');
```

3. **HTML Structure**: The HTML part defines the structure of the webpage, including a header with a form for updating the number of readings to display, and sections to display the latest temperature and humidity readings along with statistical data (min, max, average).

Example:
```
<form method="get">
  <input type="number" name="readingsCount" min="1" placeholder="Number of readings (<?php echo $readings_count; ?>)">
</form>
```
4. **Dynamic Content Display**: PHP code embedded within the HTML dynamically generates content based on the data fetched from the database, such as the latest readings and statistical summaries for temperature and humidity.

Example:
```
<p>Last reading: <?php echo $last_reading_time; ?></p>
```

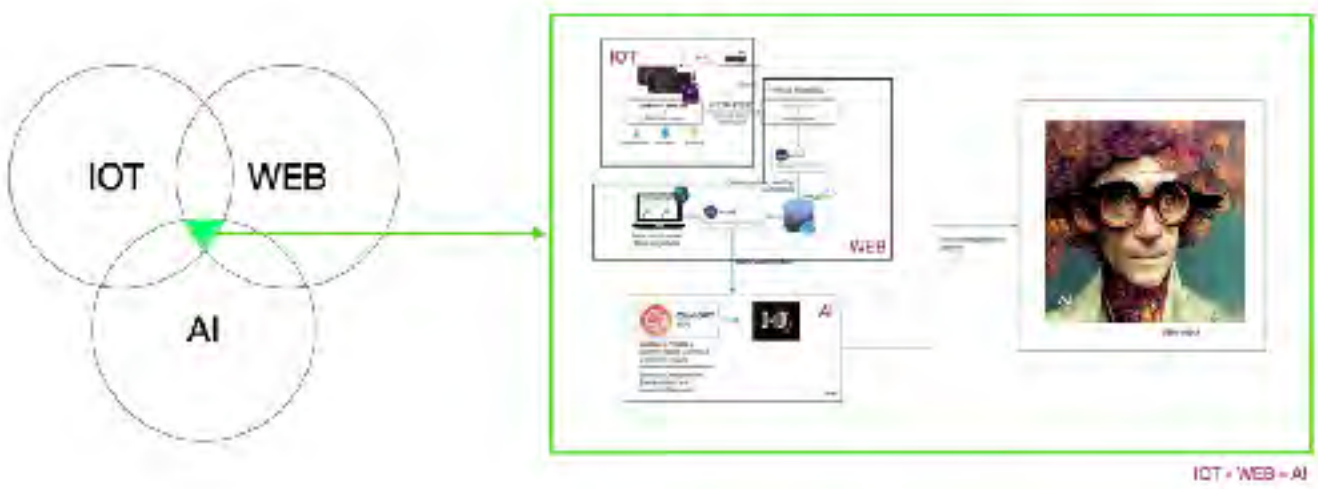
5. **JavaScript for Visualization**: The script at the bottom of the file defines functions to visually represent the temperature and humidity on gauges using CSS transformations based on the last reading values.

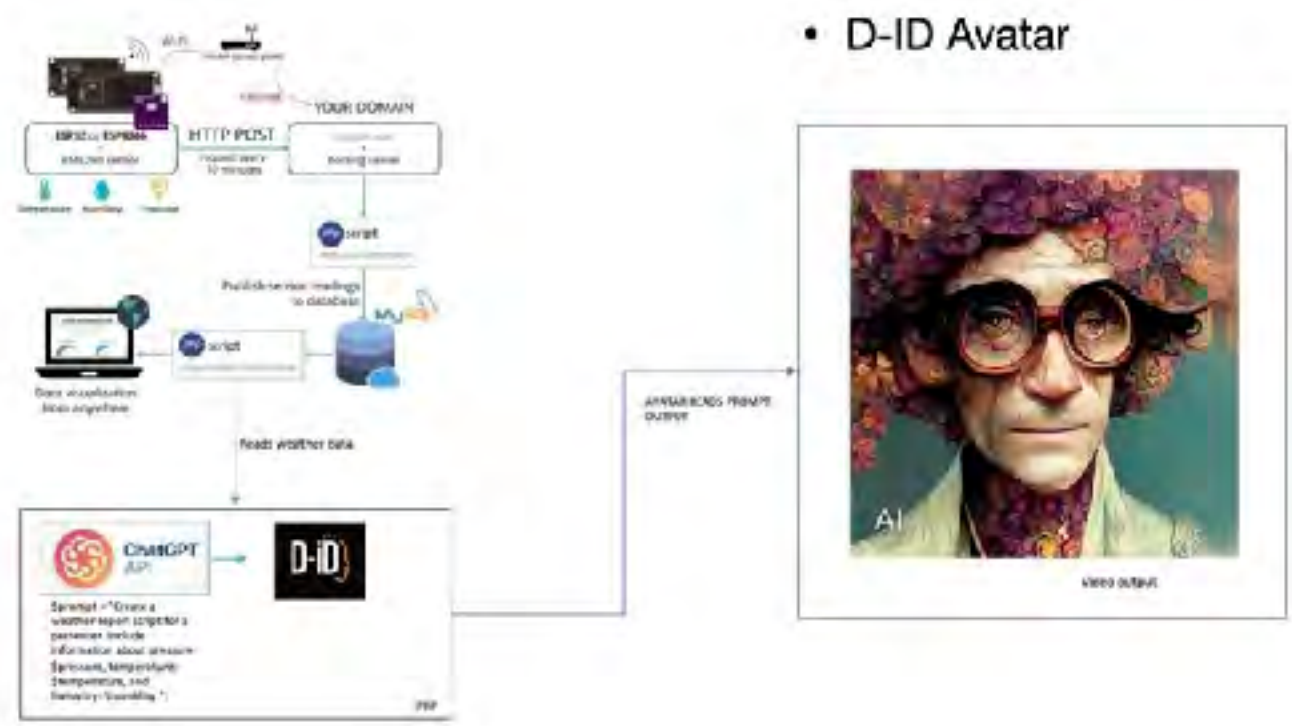
Example:
```
function setTemperature(curVal){
  ...
  $("#temp").text(curVal + ' ºC');
}
```
This code provides a comprehensive example of how to create a simple web application to display and update environmental readings from an ESP device, incorporating back-end data processing, front-end presentation, and user interaction through a web interface.

System Overview

PART V - AGENTS APPLICATIONS

GENERATIVE WEATHER PERSON

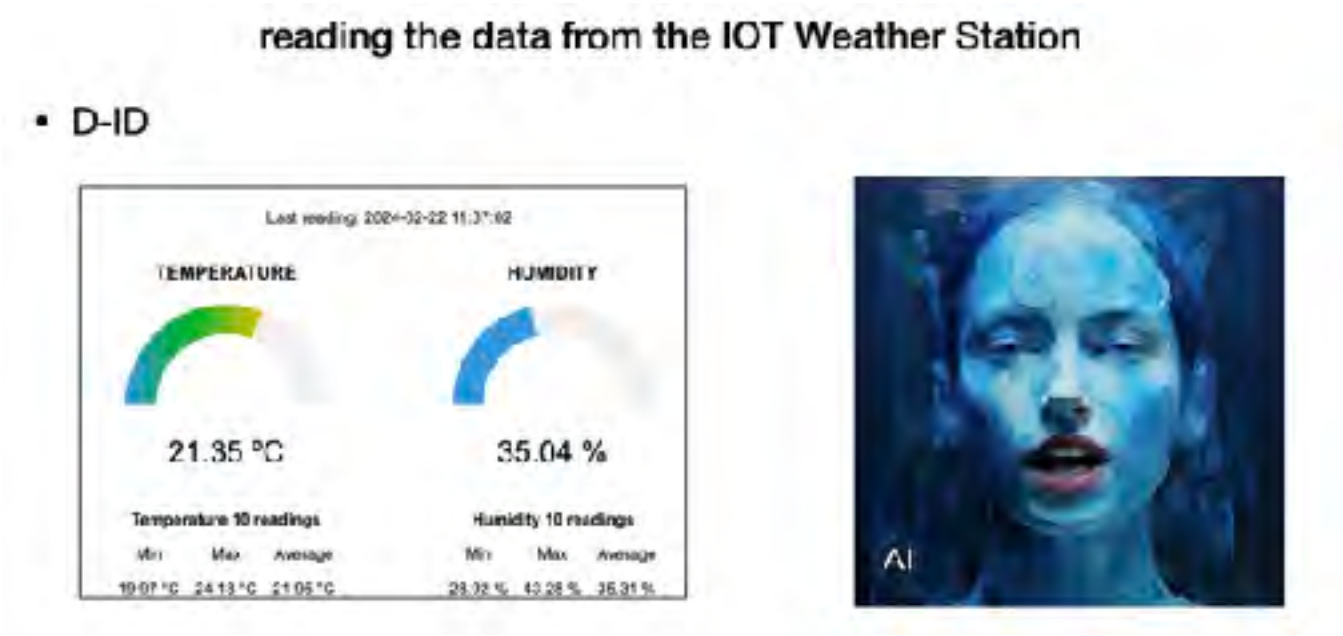




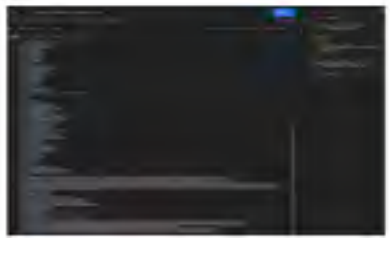
Weather Channel – IOT – GPT - TALKING AVATAR
proof of concept

CELESTIA BOREALIS

The App is a front-end for the Weather Station. It is derived from the Generative Weather Pearson Project.



Link to Open.AI



PHP Script

```
<?php
$apiKey = 'xx'; // Replace with your actual API key

// Prepare the temperature and humidity data
$temperature = '24.35 °C'; // Example temperature value
$humidity = '35.04 %'; // Example humidity value

$prompt = "Create a weather report text with lots of humor for a presenter. the name of the presneter Noah the Apocaplipic weather presenter  Include information about temperature:$temperature degrees, humidity: $humidity%";

$headers = [
    'Content-Type: application/json',
    'Authorization: Bearer ' . $apiKey
];

$data = [

    'prompt' => $prompt,
    'max_tokens' => 150
];

 $curl = curl_init();

    curl_setopt_array($curl, array(
      CURLOPT_URL => 'https://api.openai.com/v1/chat/completions',
      CURLOPT_RETURNTRANSFER => true,
      CURLOPT_ENCODING => '',
      CURLOPT_MAXREDIRS => 10,
      CURLOPT_TIMEOUT => 0,
      CURLOPT_FOLLOWLOCATION => true,
      CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
      CURLOPT_CUSTOMREQUEST => 'POST',
      CURLOPT_POSTFIELDS =>'{
          "model": "gpt-3.5-turbo",
          "messages": [
           {
             "role": "user",
             "content": "'.$prompt.'"
           }
          ]
         }',
       CURLOPT_HTTPHEADER => array(
         'Authorization: Bearer '.$apiKey,
         'Content-Type: application/json'
       ),
    ));

    $response = curl_exec($curl);

    curl_close($curl);

    curl_close($curl);

    // Process the response from the OpenAI API
    $json = json_decode($response);
    $completion = $json->choices[0]->message->content;
//     echo $response;
//     echo $completion;

if (curl_errno($ch)) {
    echo "Curl error: " . curl_error($ch);
} else {
    echo "Raw response: " . $response; // Display raw response
    $responseArray = json_decode($response, true);
    if (isset($responseArray['choices'][0]['text'])) {
        echo "Processed response: " . $responseArray['choices'][0]['text']; // Display processed response
    } else {
        echo "No processed response found.";
    }
}

curl_close($ch);
?>
```

ChatGPT Prompt

```
<!DOCTYPE html>
<html lang="en">
<head>

<title>Homemade chatGPT prompt</title>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<meta name="description" content="Create your own chatGPT prompt">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
</head>
<body>
<h1>chatGPT Prompt</h1>
<p>Ask a question to chatGPT</p>
<div class='result'>
<p>
<?php
session_start();
// Set your API key and the endpoint URL for the OpenAI API
$api_key = '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'; //add API key here
$endpoint_url = 'https://api.openai.com/v1/chat/completions';

// Check if the form has been submitted
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
// Get question
$input = $_POST['input'];

// Set up the request data for the OpenAI API
$data = array(
//'temperature' => 0.5,
'max_tokens' => 60,
//'frequency_penalty' => 0,
//'presence_penalty' => 0,
'model' => 'gpt-3.5-turbo', // Specify the model to use
'messages' => [
[
'role' => 'user',
'content' => $input
]
]
);

//add chat history to data input array
if (isset($_SESSION['history'])) {
array_unshift($data['messages'], ...$_SESSION['history']);
$data['messages'] = array_values($data['messages']);
//echo json_encode($_SESSION['history']) . "<br><br>"; //un-comment to show chat history
//echo json_encode($data) . "<br><br>"; //un-comment to show data input array
}

// Set up the HTTP headers for the request
$headers = array(
'Content-Type: application/json',
'Authorization: Bearer ' . $api_key
);

// Send the request to the OpenAI API using cURL
$ch = curl_init($endpoint_url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
$response = curl_exec($ch);
curl_close($ch);

// Parse the response from the OpenAI API
$result = json_decode($response, true);
$answer = $result['choices'][0]['message']['content'];

// Output the answer to the user
echo $answer;
```

```
//save to message history
$user_input = array([
'role' => 'user',
'content' => $input]
);
$user_output = array([
'role' => 'assistant',
'content' => $answer]
);
$history = array_merge($user_input, $user_output);

if (isset($_SESSION['history'])){
$_SESSION['history'] = array_merge($_SESSION['history'], $history);
} else {$_SESSION['history'] = $history;}
}
?>
</p>
</div>
<div class="prompt-form">
<form method="post">
<input type="text" name="input" id='input' placeholder="Ask chatGPT" required="">
<input type="submit" name="submit" value="Ask chatGPT">
</form>
</div>
</body>
</html>
```

Running the App