

IoT Worksheet 3 – Webserver to send Realtime sensory Data

Aim of this project is to send sensory data from the microbit , in this example the light level, to a simple web server in real time. This project involves three things, the microbit, and its serial data connection and the webserve.

How it works:

1. The microbit sends data through uart module which makes use of the serial usb connection
2. The data is received through the python pyseries module by making a connection to the usb port. (PLEASE NOTE THAT you need to change the 'COM3' to your usb port. To do this go to device manager and go to Ports and check which one has Microsoft one then change accordingly)
3. The data is received from a flask application. Flask is used to create a webserver
4. Flask then updates the html in real time with the sensory data received
5. The light level is displayed on your webpage

Explanation of the code:

1. Importing Libraries:

```
import serial
from flask import Flask, render_template
from flask_socketio import SocketIO
from threading import Lock
```

The code starts by importing necessary libraries. **serial** is used for serial communication with the Micro:bit, **Flask** is the web framework, **render_template** is used to render HTML templates, and **SocketIO** is used for real-time communication. The **threading** module is used for thread synchronization.

2. Creating Flask App and SocketIO Instance:

```
app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins='*')
```

Here, a Flask app instance is created, and a SocketIO instance is initialized with the app. The **cors_allowed_origins** parameter is set to allow cross-origin requests from any origin.

3. Serial Connection:

```
# serial connection to the microbit usb please change accordingly
ser = serial.Serial('COM3', 115200)
```

This line establishes a serial connection to the Micro:bit, assuming it is connected to the COM3 port at a baud rate of 115200.

4. Background Task:

```
# generate background task
1 usage  ▲ Asyl-7
✓ def background_task():
✓     try:
✓         while True:
            sensor_data = ser.readline().decode('utf-8').strip() # Read serial data from Micro:bit
            # socketio.emit('data_update', sensor_data, broadcast=True)
            socketio.emit(event='data_update', *args: {'data': sensor_data})

        except Exception as e:
            return f'Error: {e}'
```

This function runs in the background as a separate thread. It constantly reads lines from the serial connection with the Micro:bit, decodes them using UTF-8 encoding, and sends the data to connected clients using SocketIO's **emit** function. The event name used here is **'data_update'**, and the data is sent as a dictionary with a **'data'** key.

5. Flask App Route:

```
#flask app route basically homepage renders html file
Asyl-7
@app.route('/')
def index(): # put application's code here
    return render_template('index.html')
```

This route handles the root URL ("/"). It renders an HTML template named "index.html". This template contains the client-side code that will display the real-time sensor data.

6. SocketIO Connection:

```
#flask socketio connection on connect get data
Asyl-7
@socketio.on('connect')
def connect():
    global thread
    print('Client connected')

    global thread
    with thread_lock:
        if thread is None:
            thread = socketio.start_background_task(background_task)
```

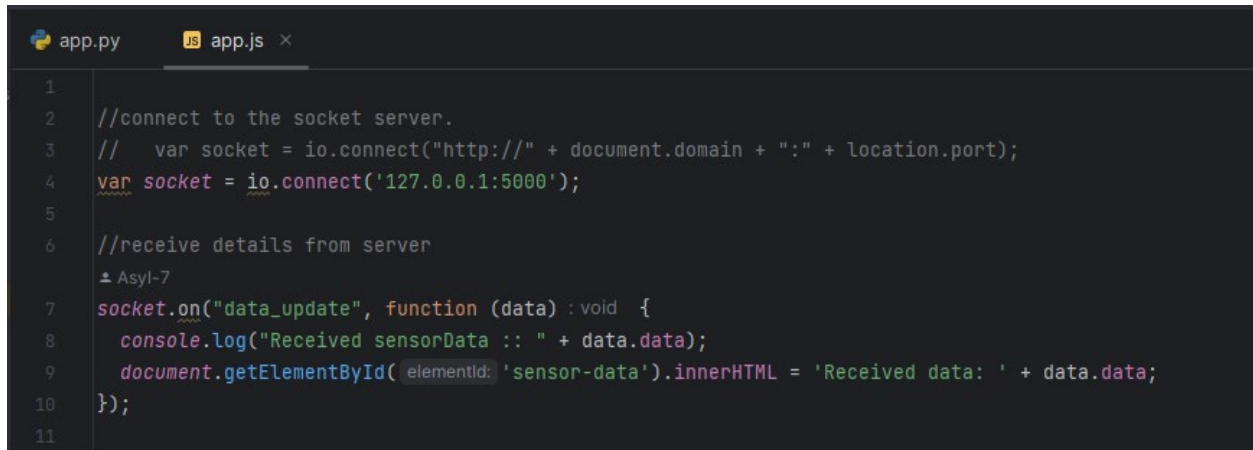
This is a SocketIO event handler that's triggered when a client connects. It starts the background task (defined earlier) in a separate thread to continuously read data from the Micro:bit and emit it to the connected clients.

7. Main Block:

```
if __name__ == '__main__':
    socketio.run(app, host='127.0.0.1', port=5000)
|
```

This block ensures that the application is run only when the script is executed directly (not imported as a module). It starts the Flask-SocketIO server with the specified host and port.

Overall, this code creates a web application that communicates with a Micro:bit through serial communication and sends the sensor data to connected clients in real-time using Flask and SocketIO. The web page has JavaScript code that listens for the **'data_update'** event and updates the displayed data accordingly.

A screenshot of a code editor with two tabs: 'app.py' and 'app.js'. The 'app.js' tab is active, showing JavaScript code. The code includes a comment about connecting to a socket server, a commented-out line for connecting to a generic http:// address, and an active line connecting to '127.0.0.1:5000'. It then defines an event listener for 'data_update' that logs the received data and updates the innerHTML of an element with the id 'sensor-data'.

```
1 //connect to the socket server.  
2 // var socket = io.connect("http://" + document.domain + ":" + location.port);  
3   
4 var socket = io.connect('127.0.0.1:5000');  
5   
6 //receive details from server  
7   
8 socket.on("data_update", function (data) :void {  
9     console.log("Received sensorData :: " + data.data);  
10    document.getElementById('sensor-data').innerHTML = 'Received data: ' + data.data;  
11 });
```

In the JS it connects to the socket server and receives updated data

Here is the microbit code

A screenshot of a code editor showing Micro:bit code. The code is in Python-like syntax and includes a comment '# Micro:bit code (simplified)'. It imports everything from the 'microbit' module and enters a 'while True' loop. Inside the loop, it reads the light level, scrolls the display with the sensor data, writes the sensor data to the UART, and sleeps for 1000 milliseconds.

```
# Micro:bit code (simplified)  
from microbit import *  
  
while True:  
    sensor_data = str(display.read_light_level())  
    display.scroll(sensor_data)  
    uart.write(sensor_data + '\n')  
    sleep(1000)
```

The threading in python helps to get the real time connection needed without making the code blocking

Video

I have included a video file named sensor data video, where it shows the microbit capturing the light level and the data is sent to webserver in real time

How to Run

- install the necessary libraries needed
- Create a virtual environment

```
python -m venv venv      # Create a virtual environment
source venv/bin/activate # Activate the virtual environment (Linux/macOS)
# OR
venv\Scripts\activate   # Activate the virtual environment (Windows)
```
- pip install Flask
- flash the microbit code
- once its done to run the app, run the code
- python app.py