

**LAPORAN TUGAS *HANDS-ON* MQTT, *WebSocket*, HTTP
II3240 – REKAYASA SISTEM DAN TEKNOLOGI INFORMASI**



**Disusun Oleh:
Kelompok 9 (K02)**

Clement Nathanael Lim	/ 18222032
Mattheuw Suciadi Wijaya	/ 18222048
Irfan Musthofa	/ 18222056
Farah Aulia	/ 18222096
Ervina Limka	/ 18222100

**PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2025

1. Pengantar

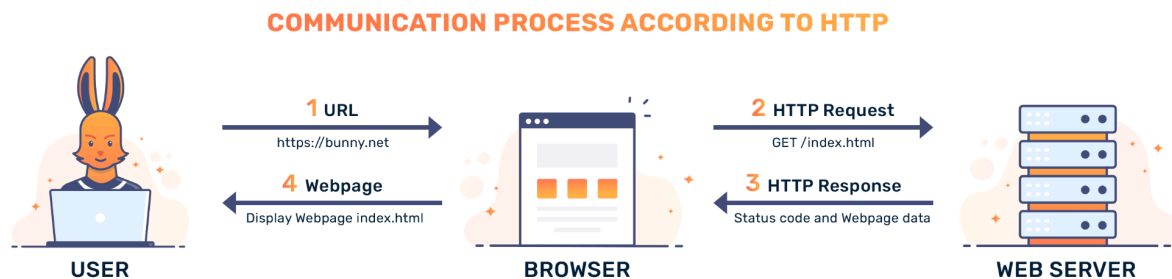
HTTP (*Hypertext Transfer Protocol*) merupakan protokol berbasis *request-response* yang digunakan untuk komunikasi antara *client* (seperti *browser*) dan *server*. Setiap pertukaran data dimulai dengan permintaan dari *client*, diikuti respons dari *server*, lalu koneksi biasanya ditutup. Model ini bersifat *stateless*, artinya *server* tidak menyimpan informasi tentang interaksi sebelumnya. Untuk aplikasi yang memerlukan pembaruan data secara *real-time*, teknik seperti *polling* (*client* meminta data berulang) atau *long polling* (*client* menunggu hingga *server* siap merespons) digunakan. Meski HTTP/2 dan HTTP/3 memperkenalkan fitur seperti *multiplexing* atau *server push*, protokol ini tetap kurang efisien untuk komunikasi dua arah secara *real-time* karena *overhead header* yang besar dan ketergantungan pada pola permintaan-respons. HTTP cocok untuk skenario tradisional seperti mengakses halaman *web*, REST API, atau mengirim formulir, tetapi kurang optimal untuk aplikasi yang memerlukan pertukaran data instan dan *real-time*.

WebSocket dirancang untuk mengatasi keterbatasan HTTP dalam komunikasi *real-time*. Protokol ini membuka koneksi persisten antara *client* dan *server* melalui proses *handshake* awal berbasis HTTP, lalu meningkatkan koneksi ke protokol *WebSocket* (dengan kode status 101 *Switching Protocols*). Setelah terhubung, *client* dan *server* bisa saling mengirim data kapan saja tanpa perlu *header* HTTP berulang, mengurangi latensi dan *overhead*. Komunikasi bersifat *full-duplex*, artinya kedua pihak dapat mengirim dan menerima data secara bersamaan. *WebSocket* ideal untuk aplikasi seperti obrolan daring, *dashboard live*, atau *game multiplayer* yang memerlukan pertukaran data *real-time*, cepat, dan dua arah. Namun, protokol ini kurang cocok untuk skala besar dengan banyak penerima (*broadcast*) karena memerlukan manajemen koneksi manual.

MQTT (*Message Queuing Telemetry Transport*) adalah protokol ringan berbasis *publish-subscribe* yang populer di dunia IoT. Protokol ini mengandalkan *broker* sebagai perantara: perangkat (*publisher*) mengirim pesan ke topik tertentu, dan *broker* meneruskannya ke semua perangkat (*subscriber*) yang berlangganan topik tersebut. MQTT dirancang untuk jaringan tidak stabil atau *bandwidth* terbatas, dengan *header* pesan hanya 2 Byte dan dukungan *Quality of Service* (QoS) untuk menjamin pengiriman pesan. Misalnya, QoS 0 mengirim pesan sekali tanpa konfirmasi, QoS 1 memastikan pesan sampai (meski mungkin duplikat), dan QoS 2 menjamin pengiriman tepat sekali. Kelebihan utama MQTT adalah skalabilitasnya untuk jutaan perangkat IoT hemat daya, seperti sensor suhu atau sistem otomasi rumah. Namun, protokol ini kurang cocok untuk transfer data besar (seperti streaming video) dan bergantung pada broker sebagai titik pusat.

2. *Hands – on* HTTP

Berikut adalah gambar alur proses percobaan *HTTP* yang akan dilakukan:



Gambar 1: Alur Protokol HTTP

Pada protokol HTTP, semua pengiriman data bersifat *stateless*, sehingga *server* tidak mengingat hasil *request-response* sebelumnya dengan *client* tertentu. Selain itu, perubahan data dapat terlihat ketika *client* meminta *request* kepada *server*.

Berikut adalah kode program percobaan protokol HTTP menggunakan Python:

```
# server.py
from flask import Flask, request, jsonify,
render_template_string

app = Flask(__name__)
latest_suhu = None

# Halaman utama
@app.route('/', methods=['GET', 'POST'])
def index():
    global latest_suhu
    if request.method == 'POST':
        suhu = request.form.get('suhu')
        latest_suhu = suhu
        return render_template_string(HTML_TEMPLATE,
suhu=suhu)
    return render_template_string(HTML_TEMPLATE,
suhu=latest_suhu)

# Endpoint API untuk client.py
@app.route('/sensor', methods=['POST'])
def receive_data():
    global latest_suhu
    data = request.json
    suhu = data.get("suhu")
    latest_suhu = suhu
    print(f>Data diterima: Suhu = {suhu} °C")
    return jsonify({"status": "success", "message": f"Suhu
{suhu} diterima"}), 200

HTML_TEMPLATE = '''
<!DOCTYPE html>
<html>
<head>
    <title>Monitoring Suhu</title>
</head>
<body>
    <h1>Form Input Suhu Manual</h1>
    <form method="POST">
        <label>Masukkan Suhu (°C):</label>
        <input type="number" step="0.01" name="suhu"
required>
        <button type="submit">Kirim</button>
    </form>
```

```

{% if suhu %}
    <h2>Data Terakhir Diterima:</h2>
    <p>Suhu: <strong>{{ suhu }} °C</strong></p>
{% else %}
    <p>Belum ada data dikirim.</p>
{% endif %}
</body>
</html>
'''

if __name__ == '__main__':
    app.run(debug=True, port=5000)

```

```

# client.py
import requests
import random
import time

url = 'http://localhost:5000/sensor'

for i in range(5):
    suhu = round(random.uniform(25.0, 35.0), 2)
    payload = {"suhu": suhu}
    response = requests.post(url, json=payload)
    print(f"[Client] Mengirim suhu: {suhu} °C")
    print(f"[Server] Respon: {response.json()}")
    time.sleep(1)

```

Program ini adalah implementasi sederhana terhadap protokol HTTP menggunakan Python yang terdiri 2 *file*, *server.py* sebagai server yang memiliki sebuah *Endpoint API* (*/sensor*) dengan *method* POST untuk *client* terpisah yang ingin mengirim data suhu terbaru ke *server* dan akan ditampilkan dalam *web* dengan tampilan html biasa. *Server* juga memiliki *form* untuk mengirim data suhu terbaru secara langsung, yakni pada halaman utama (*/[root]*). *Server* yang diimplementasikan pada *hands-on* ini bersifat lokal, sehingga URL yang digunakan adalah <http://localhost:5000> dengan port 5000.

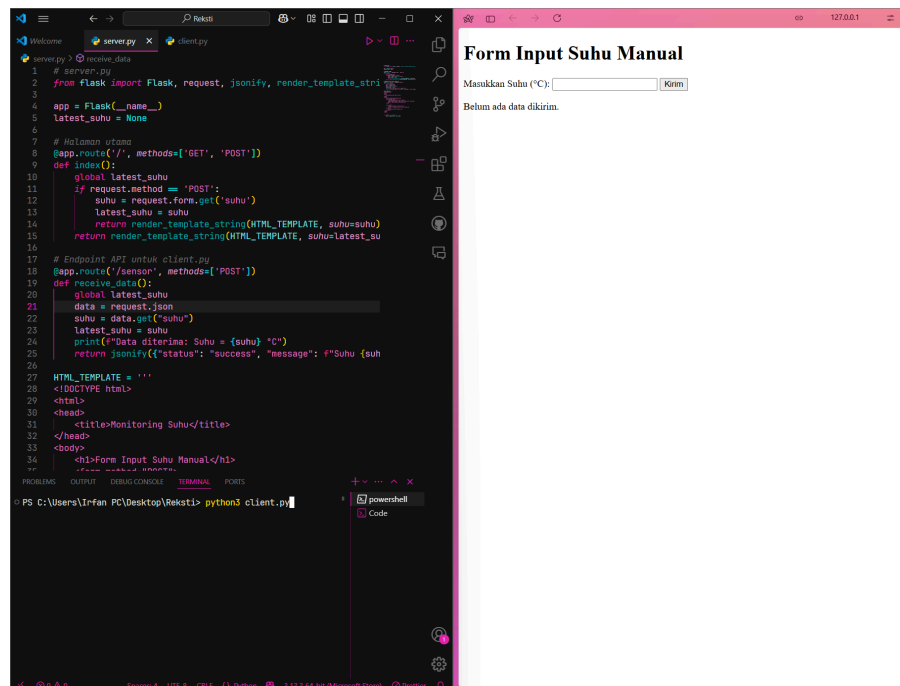
Sedangkan file *client.py* diibaratkan sebagai *client* terpisah yang ingin mengirim data suhu terbaru tanpa menggunakan fitur *form* pada halaman utama, yakni menggunakan protokol HTTP dengan *method POST* ke <http://localhost:5000/sensor>. Pada kode, program *client.py* akan mengirimkan angka *float* random dari 25.0 - 32.0 sebagai suhu ke *server* sebanyak 5x.

Langkah-Langkah

Berikut langkah-langkah yang dilakukan dalam percobaan ini:

1. Masuk ke direktori folder HTTP.
2. Jalankan kode program *server.py* terlebih dahulu (dengan *command* **python3 server.py** di terminal).
3. Buka *web browser* lalu akses link <http://localhost:5000/>
4. Jalankan kode program *client.py* di terminal terpisah tanpa menghentikan program *server.py* (dengan *command* **python3 client.py** di terminal terpisah).
5. Refresh *web browser* lalu *refresh* link <http://localhost:5000/> secara berkala dan lihat perubahan data yang diterima.

Hasil



```
# server.py
1 # Import data
2 from flask import Flask, request, jsonify, render_template_string
3
4 app = Flask(__name__)
5 latest_suhu = None
6
7 # Halaman index
8 @app.route('/', methods=['GET', 'POST'])
9 def index():
10     global latest_suhu
11     if request.method == 'POST':
12         suhu = request.form.get('suhu')
13         latest_suhu = suhu
14         return render_template_string(KTML_TEMPLATE, suhu=suhu)
15     return render_template_string(KTML_TEMPLATE, suhu=latest_suhu)
16
17 # Endpoint API untuk client.py
18 @app.route('/sensor', methods=['POST'])
19 def receive_data():
20     global latest_suhu
21     data = request.json
22     suhu = data.get('suhu')
23     latest_suhu = suhu
24     print(f'Data diterima: Suhu = {suhu} °C')
25     return jsonify({'status': 'success', 'message': f'Suhu {suhu}'})
26
27 KTMPL_TEMPLATE = '''
28 <!DOCTYPE html>
29 <html>
30 <head>
31 <title>Monitoring Suhu</title>
32 </head>
33 <body>
34 <div>Form Input Suhu Manual</div>
35 </body>
36 </html>'''
37
38 if __name__ == '__main__':
39     app.run(debug=True)
```

```
PS C:\Users\Irfan\PC\Desktop\hands-on> python client.py
[Client] Mengirim suhu: 36.52 °C
[Server] Respon: {'message': 'Suhu 36.52 diterima', 'status': 's
success'}
[Client] Mengirim suhu: 28.59 °C
[Server] Respon: {'message': 'Suhu 28.59 diterima', 'status': 's
success'}
```

Pengiriman Data ke-2 dari *Client.py*

```
# server.py
1 # Import data
2 from flask import Flask, request, jsonify, render_template_string
3
4 app = Flask(__name__)
5 latest_suhu = None
6
7 # Halaman index
8 @app.route('/', methods=['GET', 'POST'])
9 def index():
10     global latest_suhu
11     if request.method == 'POST':
12         suhu = request.form.get('suhu')
13         latest_suhu = suhu
14         return render_template_string(KTML_TEMPLATE, suhu=suhu)
15     return render_template_string(KTML_TEMPLATE, suhu=latest_suhu)
16
17 # Endpoint API untuk client.py
18 @app.route('/sensor', methods=['POST'])
19 def receive_data():
20     global latest_suhu
21     data = request.json
22     suhu = data.get('suhu')
23     latest_suhu = suhu
24     print(f'Data diterima: Suhu = {suhu} °C')
25     return jsonify({'status': 'success', 'message': f'Suhu {suhu}'})
26
27 KTMPL_TEMPLATE = '''
28 <!DOCTYPE html>
29 <html>
30 <head>
31 <title>Monitoring Suhu</title>
32 </head>
33 <body>
34 <div>Form Input Suhu Manual</div>
35 </body>
36 </html>'''
37
38 if __name__ == '__main__':
39     app.run(debug=True)
```

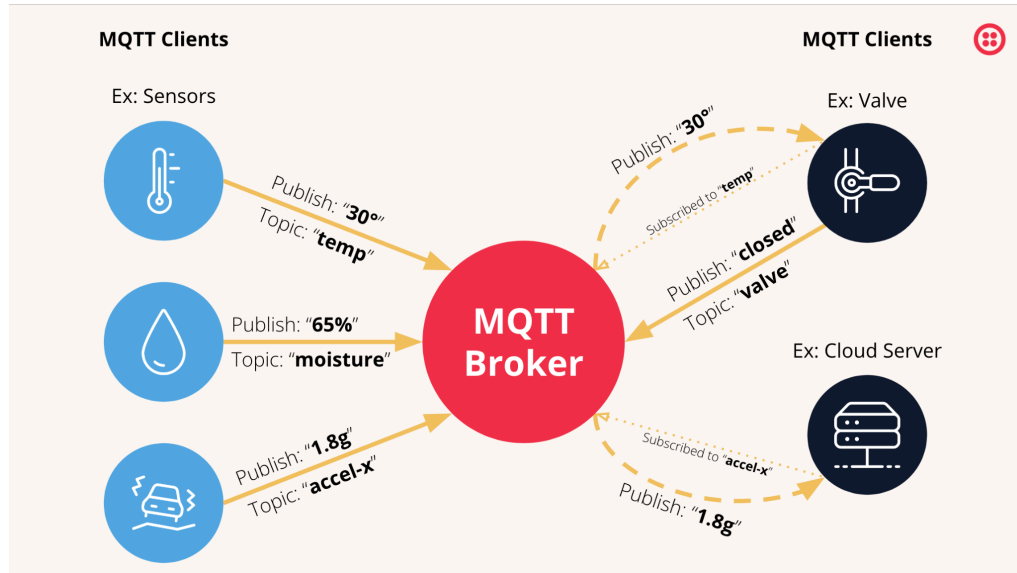
```
PS C:\Users\Irfan\PC\Desktop\hands-on> python client.py
[Client] Mengirim suhu: 36.52 °C
[Server] Respon: {'message': 'Suhu 36.52 diterima', 'status': 's
success'}
[Client] Mengirim suhu: 28.59 °C
[Server] Respon: {'message': 'Suhu 28.59 diterima', 'status': 's
success'}
[Client] Mengirim suhu: 27.34 °C
[Server] Respon: {'message': 'Suhu 27.34 diterima', 'status': 's
success'}
[Client] Mengirim suhu: 27.71 °C
[Server] Respon: {'message': 'Suhu 27.71 diterima', 'status': 's
success'}
```

Pengiriman Data ke-5 dari *Client.py*

Pada percobaan *hands-on* protokol HTTP, tampilan perubahan data akibat pengiriman data dari *Client.py* tidak secara *real-time* langsung *ter-update*, sehingga perlu melakukan *refresh* terus-menerus untuk melihat perbedaan. Hal ini terjadi karena protokol bersifat *stateless* dan langsung menutup komunikasi untuk setiap siklus *request-response*. Protokol HTTP ini juga memerlukan *header* yang besar untuk *request* bersifat *stateless* seperti REST/GraphQL dan pengunduhan berkas. Sehingga protokol HTTP banyak digunakan di *website* normal, *website* yang membutuhkan REST API, dan *file transfer*.

3. Hands – on MQTT

Berikut ini merupakan gambar alur proses percobaan MQTT yang akan dilakukan:



Gambar 2: Alur Protokol MQTT

Untuk *hands – on* protokol MQTT sendiri memerlukan 3 buah *file*, antara lain *index.html* (tampilan), *Publisher MQTT*, dan juga *Subscriber MQTT*. Berikut ini merupakan kode untuk masing – masing *file*.

1) mqtt_publisher.py

```
from flask import Flask, request, render_template
import paho.mqtt.publish as publish

app = Flask(__name__)
broker = "localhost"
port = 1883
topic = "web_topic"

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        message = request.form['message']
        try:
            # Kirim pesan ke broker lokal
            publish.single(
```



```

        topic,
        message,
        hostname=broker,
        port=port
    )
    return render_template('/index.html',
status=f"Sent: {message}")
    except Exception as e:
        return render_template('/index.html',
status=f"Error: {str(e)}")

    return render_template('index.html', status="Ready")

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000, debug=True)

```

2) mqtt_subscriber.py

```

import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, reason_code,
properties):
    if reason_code == 0:
        print("Connected to broker!")
        client.subscribe("web_topic")
    else:
        print(f"Connection failed! Code: {reason_code}")

def on_message(client, userdata, msg):
    print(f"Received: {msg.payload.decode()}")

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2,
"Subscriber_V2")
client.on_connect = on_connect
client.on_message = on_message
try:
    client.connect("localhost", 1883)
    client.loop_forever()
except ConnectionRefusedError:
    print("\nERROR: Broker tidak aktif!")
    print("1. Pastikan Mosquitto sudah diinstall")
    print("2. Jalankan broker di terminal: mosquitto -v")
except KeyboardInterrupt:
    print("\nSubscriber dihentikan")

```

3) index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>MQTT Web Publisher</title>
  <style>
    body { font-family: Arial, sans-serif; max-width:
800px; margin: 0 auto; padding: 20px; }
    .container { border: 1px solid #ccc; padding: 20px;
border-radius: 5px; }
    input[type="text"] { width: 70%; padding: 8px;
margin-right: 10px; }
    button { padding: 8px 20px; background: #4CAF50;
color: white; border: none; border-radius: 3px; }
    .status { margin-top: 15px; color: #666; }
  </style>
</head>
<body>
  <div class="container">
    <h1>Local MQTT Publisher</h1>
    <form method="POST">
      <input type="text" name="message"
placeholder="Enter message..." required>
      <button type="submit">Publish</button>
    </form>
    <div class="status">
      Status: {{ status }}
    </div>
  </div>
</body>
</html>
```

Langkah-Langkah

Berikut ini merupakan langkah – langkah untuk mengakses MQTT:

- 1) Jalankan *mosquitto* untuk mengaktifkan *broker*.

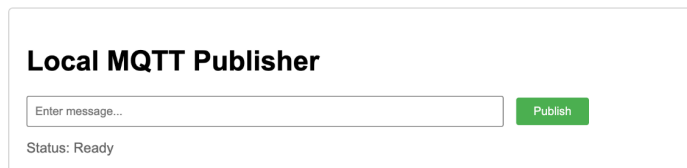
```
Mosquitto -v
```

- 2) Jalankan masing – masing kode program.

```
python3 mqtt_subscriber.py  
python3 mqtt_publisher.py
```

- 3) Buka <http://127.0.0.1:8000> pada *website* untuk mengakses tampilan.

Berikut ini merupakan tampilan antarmuka dari MQTT:



Local MQTT Publisher

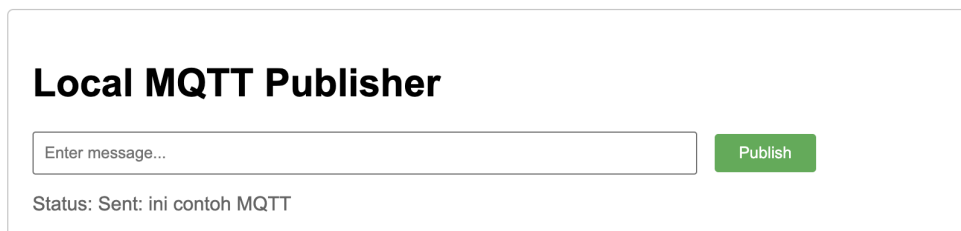
Enter message...

Publish

Status: Ready

- 4) Ketikkan contoh pesan yang ingin dikirim kepada *subscriber*, lalu klik “*Publish*”.

Status pesan akan berubah setelah dikirim.



Local MQTT Publisher

Enter message...

Publish

Status: Sent: ini contoh MQTT

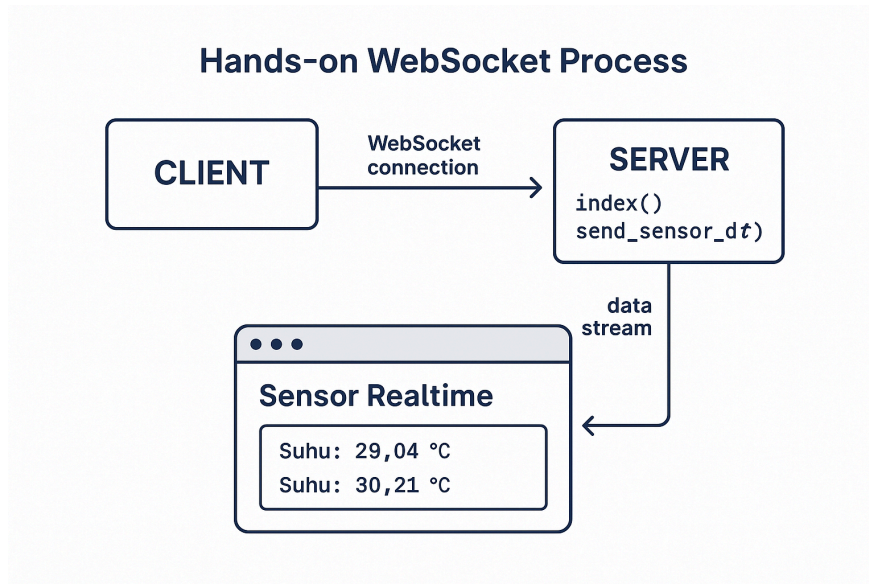
- 5) Cek *terminal subscriber*, maka pesan akan muncul di *terminal subscriber*. Berikut ini merupakan contoh tampilannya:

```
○ clementnathanael@Clems-MacBook-Pro-M1 MQTT % python3 mqtt_subscriber.  
py  
Connected to broker!  
Connected to broker!  
Received: ini contoh MQTT  
□
```

Hasil dari percobaan ini menunjukkan bahwa MQTT merupakan salah satu protokol komunikasi yang *seamless* dan tidak memerlukan *refresh* per detik. MQTT memungkinkan pesan yang dikirim akan langsung diterima secara otomatis layaknya sebuah notifikasi dalam *smartphone*. MQTT menerapkan arsitektur yang bersifat *event – driven*, di mana sistem akan langsung mendeteksi dan menerima *event* secara *real – time*.

4. *Hands – on WebSocket*

Berikut adalah gambar alur proses percobaan *WebSocket* yang akan dilakukan:

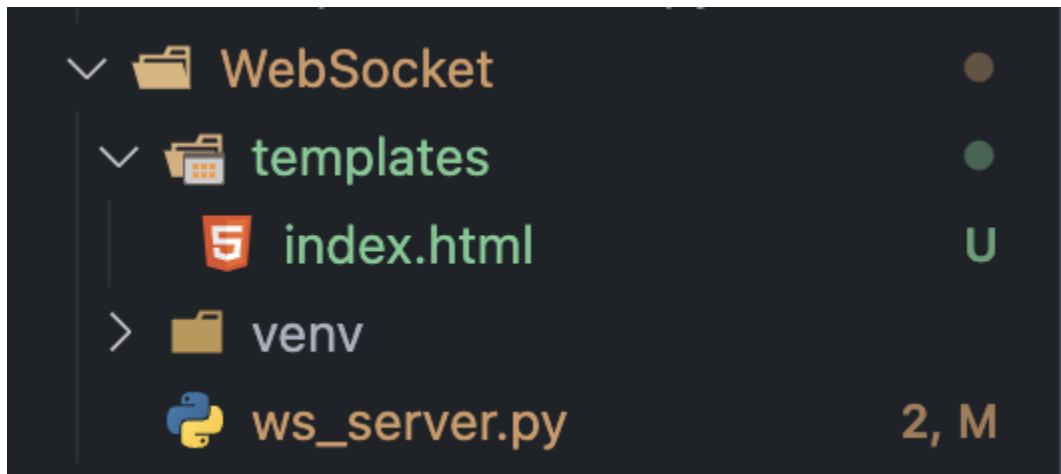


Gambar 3: Alur Protokol WebSocket

Hands-on WebSocket ini menunjukkan bagaimana server dapat mengirim data **secara *real-time*** ke browser tanpa perlu permintaan dari *client*. Saat browser membuka halaman, ia membentuk koneksi WebSocket ke server. Server kemudian secara otomatis mengirimkan data suhu acak setiap 2 detik melalui koneksi tersebut. Data yang diterima ditampilkan langsung di halaman web. Ini menggambarkan komunikasi dua arah yang efisien antara client dan server, yang sangat berguna untuk aplikasi seperti *dashboard* IoT atau notifikasi *live*.

Kode Program dan Struktur File

Berikut adalah struktur folder dari kode program untuk *hands-on* ini:



Berikut untuk kode programnya:

```
## ws_server.py
from flask import Flask, render_template
from flask_socketio import SocketIO
import random
import threading
import time

app = Flask(__name__)
socketio = SocketIO(app)

@app.route('/')
def index():
    return render_template('index.html')

def send_sensor_data():
    while True:
        suhu = round(random.uniform(25.0, 35.0), 2)
        socketio.emit('sensor_data', {'suhu': suhu})
        time.sleep(2)

# Jalankan thread untuk kirim data otomatis
threading.Thread(target=send_sensor_data,
                  daemon=True).start()

if __name__ == '__main__':
    socketio.run(app, debug=True)
```

Program ini adalah implementasi sederhana komunikasi real-time menggunakan WebSocket dengan Flask dan Flask-SocketIO. Saat pengguna membuka halaman web, server akan mengirimkan data suhu acak setiap dua detik melalui koneksi WebSocket. Data ini dikirim menggunakan fungsi `send_sensor_data()` yang berjalan di background thread agar tidak mengganggu proses utama server. Saat data dikirim dengan `socketio.emit()`, client akan menerimanya secara langsung tanpa perlu refresh. Hands-on ini menunjukkan bagaimana server dapat aktif mengirimkan data ke client, cocok untuk aplikasi real-time seperti monitoring sensor atau dashboard IoT.

Berikut untuk kode tampilannya:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Sensor Realtime</title>
</head>
<body>
  <h2>Data Sensor Suhu</h2>
  <ul id="data-list"></ul>

  <script
src="https://cdn.socket.io/4.0.0/socket.io.min.js"></scrip
t>

  <script>
    const socket = io();

    socket.on('sensor_data', function(data) {
      const li = document.createElement("li");
      li.textContent = "Suhu: " + data.suhu + " °C";

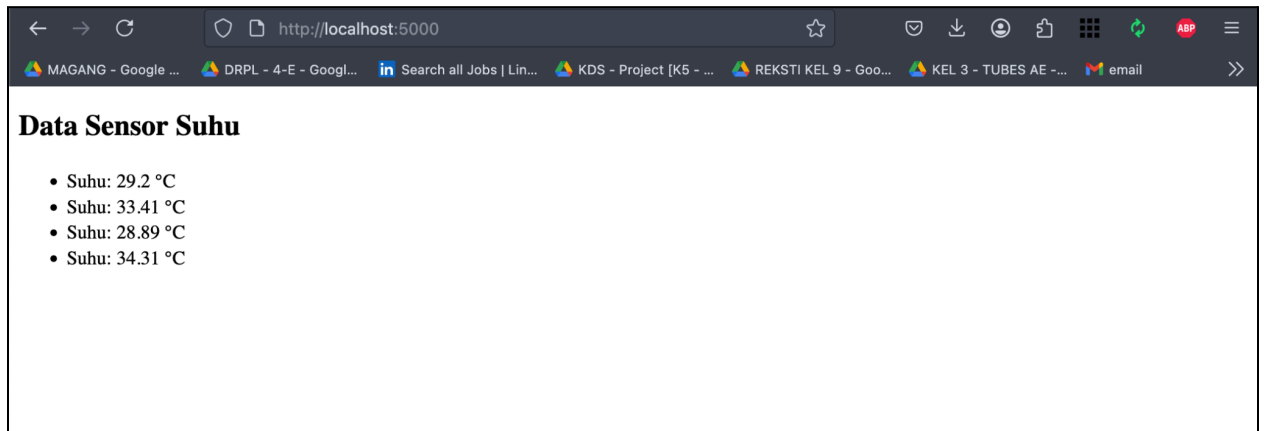
      document.getElementById("data-list").appendChild(li);
    });
  </script>
</body>
</html>
```

Langkah-Langkah

Berikut langkah-langkah yang dilakukan dalam percobaan ini:

1. lankan kode program `ws_server.py`
2. Jalankan main program dengan *command* `python3 ws_server.py`
3. Masuk ke *link* <http://localhost:5000/>

Hasil



Hasil dari percobaan ini menunjukkan bahwa komunikasi *real-time* menggunakan WebSocket berhasil diterapkan dengan baik. Setelah halaman web dibuka, browser langsung menerima data suhu acak yang dikirim oleh server setiap dua detik tanpa perlu melakukan refresh. Data tersebut langsung ditampilkan secara dinamis di halaman, membuktikan bahwa koneksi WebSocket aktif dan berjalan. Ini menunjukkan bahwa WebSocket sangat efektif untuk mengirim data secara terus-menerus dan instan, cocok untuk aplikasi yang membutuhkan *update* data *real-time* seperti monitoring sensor, notifikasi langsung, atau dashboard interaktif.