Report: Greenhouse Monitoring and Control Web Application

Objective

The primary goal of this project was to create a real-time web application for monitoring and controlling various greenhouse parameters, including temperature, humidity, motion, and distance. This application integrates IoT data collection, user authentication, and dynamic updates into a seamless system. By leveraging MQTT-based communication, it enables efficient data exchange between sensors and the backend, while a responsive user interface provides real-time updates and interactive control options.

System Overview

The system offers a range of features aimed at optimizing greenhouse management. Real-time monitoring ensures live tracking of environmental conditions, such as temperature and humidity, which are displayed on a user-friendly dashboard. The application includes a robust authentication mechanism powered by Flask-Login, allowing secure user registration and login. MQTT communication forms the backbone of sensor-server interactions, while WebSockets facilitate instant dashboard updates whenever sensor data changes. Users can configure acceptable ranges for temperature and humidity, ensuring tailored environmental control. Additionally, Cross-Origin Resource Sharing (CORS) is implemented to allow secure data sharing between different domains.

Technology Stack:

The backend, built using Flask, incorporates Flask-SocketIO for WebSocket communication and Flask-MQTT for managing IoT interactions. It processes REST API requests, manages user sessions, and handles MQTT subscriptions and publications. The frontend employs HTML, CSS, and JavaScript for a polished and responsive user interface, with Bootstrap ensuring adaptability across devices. Currently, the system uses in-memory data storage for user credentials, though future iterations could integrate a persistent database, such as SQLite or PostgreSQL. MQTT is utilized for real-time data exchange between sensors and the server, complemented by REST APIs for structured frontend-backend communication.

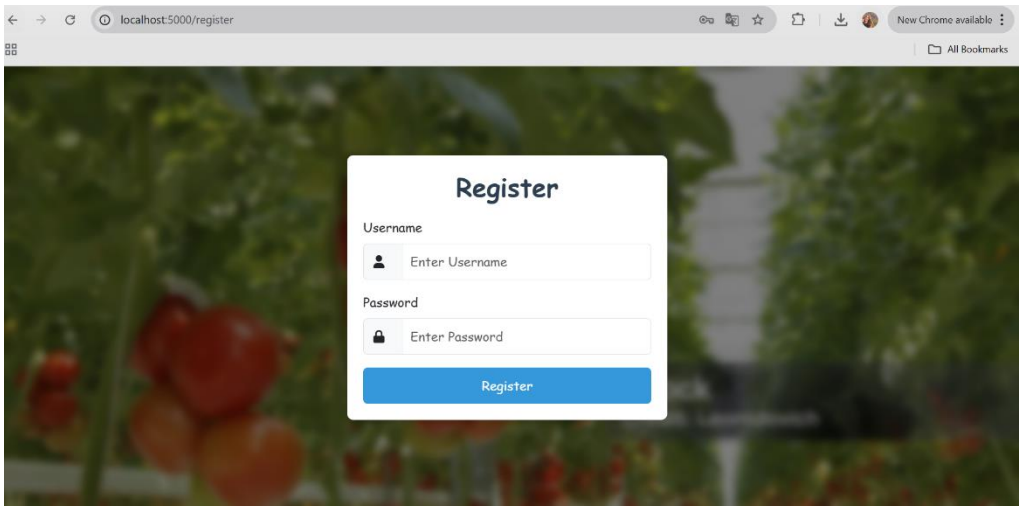
The application is hosted locally on http://localhost:5000, making it accessible for development and testing.

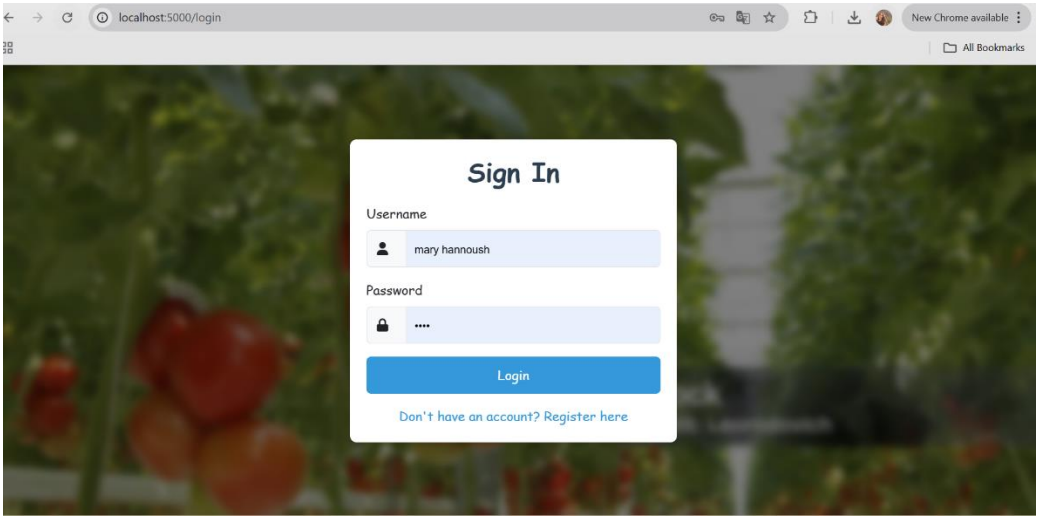
System Architecture

The system is structured into three main components: the frontend, backend, and IoT devices.

- The frontend provides a clean interface for users to register, log in, and monitor greenhouse parameters in real time. It communicates with the backend via REST APIs to fetch data and

WebSockets to receive updates dynamically. The dashboard also allows users to configure environmental thresholds for temperature and humidity.





- The backend plays a central role in managing user authentication, processing IoT sensor data, and facilitating WebSocket communication for real-time updates. It subscribes to MQTT topics to receive sensor data and processes it to determine the state of the greenhouse environment. REST APIs enable seamless data sharing and management of user settings.
- The IoT devices comprise sensors and actuators. Sensors publish real-time data on temperature, humidity, motion, and distance to MQTT topics, while actuators, such as relays, respond to commands from the server. These commands are based on predefined conditions, such as maintaining acceptable temperature and humidity ranges or detecting motion.
-

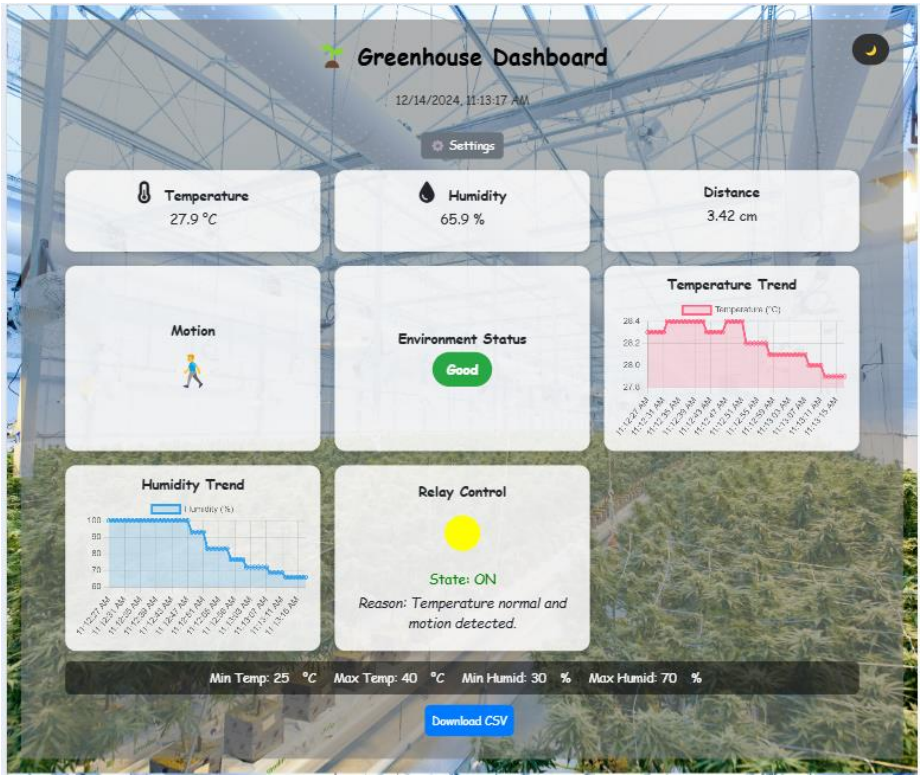
Frontend Implementation

The frontend implementation of the Greenhouse Monitoring and Control Web Application demonstrates an interactive, user-friendly interface for real-time environmental data tracking and actuator control. The design integrates modern web technologies and libraries to ensure responsiveness and functionality.

The application features dynamic data updates and seamless communication with the backend to ensure a responsive and interactive user experience. For real-time updates, the frontend leverages WebSocket integration, allowing live data streams from the server to update the dashboard instantly. This eliminates the need for page reloads when there are changes in parameters like temperature or humidity. JavaScript handles the incoming WebSocket messages, dynamically updating the DOM (Document Object Model) to reflect the latest sensor readings.

In addition to WebSocket functionality, the frontend communicates with the backend using REST APIs. These APIs facilitate user authentication, retrieval of historical sensor data, and fetching of current system settings. JavaScript processes API responses, rendering the data dynamically on the dashboard. Error handling mechanisms, such as alert boxes and modals, provide immediate feedback to users in cases of failed data retrieval, unauthorized access, or configuration errors, ensuring a smooth and transparent interaction with the system.

The **dashboard layout**, as illustrated in the provided code, offers intuitive visualizations of key greenhouse metrics, including temperature, humidity, motion, and distance. Each parameter is represented in individual cards, enhanced by icons and live updates fetched via an API. For example, temperature and humidity values dynamically update on their respective cards, with alerts changing the environment's status badge to “Good,” “Moderate,” or “Poor” based on predefined thresholds. Furthermore, the relay control card dynamically reflects real-time state changes triggered by sensor data, highlighting the reasons behind each state.



```
const statusBadge = document.getElementById("environmentStatus");
if (data.temperature < 30 && data.humidity > 40) {
  statusBadge.textContent = "Good";
  statusBadge.className = "status-badge status-good";
} else if (data.temperature >= 30 || data.humidity < 40) {
  statusBadge.textContent = "Moderate";
  statusBadge.className = "status-badge status-moderate";
} else {
  statusBadge.textContent = "Poor";
  statusBadge.className = "status-badge status-poor";
}

const relayState = document.getElementById("relayState");
const relayReason = document.getElementById("relayReason");
```

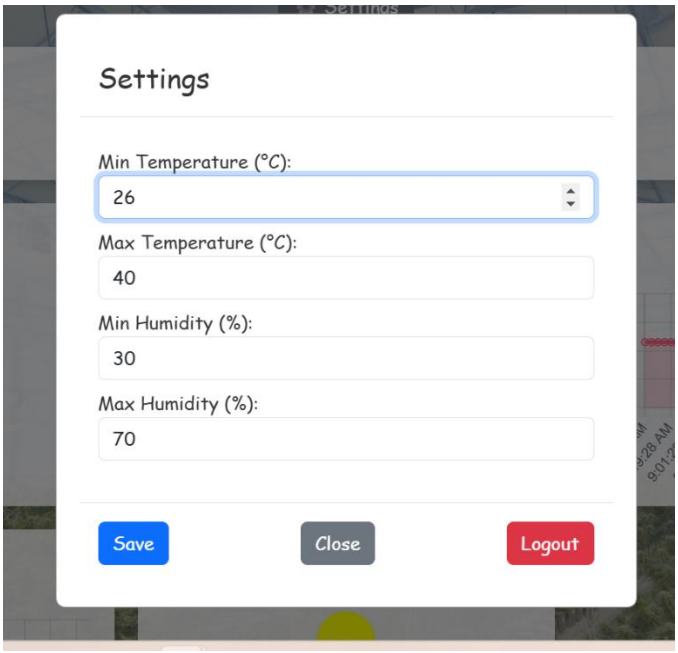
In our application, users have the option to download a CSV file containing historical data for review and analysis. This functionality allows users to export key environmental data from the greenhouse system. Users can download historical data for offline analysis, formatted with columns for date, time, temperature, humidity, motion status, and relay reasons.

The data is organized in a tabular format, which can be opened in any spreadsheet software (such as Microsoft Excel or Google Sheets) for further inspection or record-keeping.

	A	B	C	D	E	F	G	H
	Date	Time	Temperature	Humidity	Distance	Motion	Relay Stat	Relay Reason
	12/14/2024	11:19:41 AM	20.1	74.7	105.13	No Motion	ON	Temperature or humidity is low.
	12/14/2024	11:19:41 AM	20.1	74.7	105.13	No Motion	ON	Temperature or humidity is low.
	12/14/2024	11:19:43 AM	20.1	74.7	105.11	No Motion	ON	Temperature or humidity is low.
	12/14/2024	11:19:43 AM	20.1	74.7	105.11	No Motion	ON	Temperature or humidity is low.
	12/14/2024	11:19:45 AM	20.1	74.7	87.87	No Motion	ON	Temperature or humidity is low.
	12/14/2024	11:19:45 AM	20.1	74.7	87.87	No Motion	ON	Temperature or humidity is low.
	12/14/2024	11:19:47 AM	20.1	74.7	105.47	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:19:48 AM	20.1	74.7	105.47	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:19:56 AM	20.1	75.2	72.42	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:19:57 AM	20.1	75.2	104.36	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:19:57 AM	20.1	75.2	104.36	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:19:59 AM	20	75.3	71.03	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:20:00 AM	20	75.3	71.03	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:20:01 AM	20	75.3	103.34	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:20:02 AM	20	75.3	103.34	Detected	ON	Temperature or humidity is low.
	12/14/2024	11:20:03 AM	20	75.3	105.26	No Motion	ON	Temperature or humidity is low.
	12/14/2024	11:20:03 AM	20	75.4	105.26	No Motion	ON	Temperature or humidity is low.

```
function downloadCSV() {
  let csvContent =
    "data:text/csv;charset=utf-8,Date,Time,Temperature,Humidity,Distance,Motion,Relay State,Relay Reason\n";
  for (let i = 0; i < dateData.length; i++) {
    csvContent += `${dateData[i]},${timeData[i]},${temperatureData[i]},${humidityData[i]},${distanceData[i]},${motionData[i]},${relay
  }
  const encodedUri = encodeURI(csvContent);
  const link = document.createElement("a");
  link.setAttribute("href", encodedUri);
  link.setAttribute("download", "greenhouse_data.csv");
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
}
```

The **settings modal** allows users to customize the acceptable thresholds for temperature and humidity. This feature ensures the application adapts to specific greenhouse requirements. Upon saving these settings, the system updates the dashboard's visual indicators and communicates the new configurations to the backend using an AJAX POST request.:

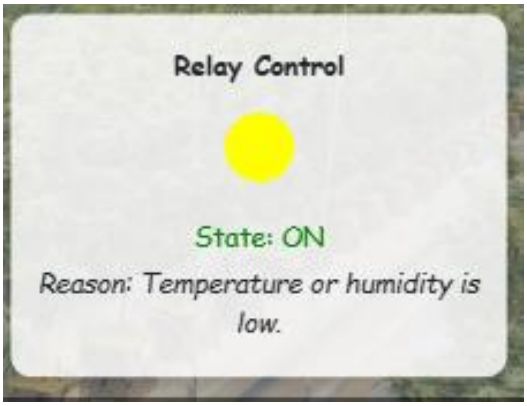


The code snippet also illustrates seamless integration of real-time updates using WebSockets and periodic API polling. Sensor data, such as motion detection and distance measurements, update every second. The **motion card** uses an icon to visually differentiate between detected motion (🚶) and no motion (⛔), while distance data is displayed in centimeters.

```
document.getElementById("distanceValue").textContent =
  data.distance + " cm";
if (data.motion === "Motion Detected") {
  document.getElementById("motionIcon").textContent = " 🚶 ";
} else {
  document.getElementById("motionIcon").textContent = " ⛔ ";
}
```

We have also a card to check the **state of the rely** that we have , we use as a lamp if it is on it will be yellow , otherwise it will be gray , also we display the reason of why it is automatically on or off depending on the same conditions that we already implement for our project :

```
# Determine relay state based on conditions
relay_state = "OFF"
reason = "Conditions do not match any ON state."
if temp_state == "normal" and motion_state == "detected":
    relay_state = "ON"
    reason = "Temperature normal and motion detected."
elif temp_state == "low" or humid_state == "low":
    relay_state = "ON"
    reason = "Temperature or humidity is low."
elif temp_state == "high" or humid_state == "high" or motion_state == "not detected":
    relay_state = "OFF"
    reason = "Temperature/humidity too high or no motion detected."
elif distance_state is not None and distance_state < 10:
    relay_state = "ON"
    reason = "Object detected within 10 units of distance."
```



- For a polished user experience, the application includes features like a **dark mode toggle**, responsive charts for temperature and humidity trends using the Chart.js library, and CSV data export functionality as we mention before.

Backend Implementation

1. User Authentication

The backend uses **Flask-Login** to manage user sessions and handle authentication securely. User credentials, including usernames and passwords, are temporarily stored in a Python dictionary named users for in-memory processing. Two primary routes, /login and /register, serve login and registration forms, respectively. During the login process, password validation is performed to ensure secure access, and a session is initiated upon successful authentication. This approach provides a straightforward and efficient mechanism for managing user access in the system.

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        data = request.get_json()
        username = data.get('username')
        password = data.get('password')

        if username in users:
            return jsonify({"message": "User already exists!"}), 400


        users[username] = {"password": password}
        return jsonify({"message": "Registration successful. Please log in.", "redirect": "/login"

    return render_template('register.html')
```

2. Real-Time Data Handling

Real-time communication is enabled using the **Flask-MQTT** library, which subscribes to the MQTT topics (e.g., greenhouse/#) to process incoming sensor data. As sensor updates are received, MQTT messages are decoded and stored in a global Python dictionary to maintain current environmental parameters. Additionally, **Flask-SocketIO** is used to broadcast these updates to all connected clients dynamically, ensuring that the frontend dashboard remains synchronized with the latest data without requiring manual refreshes.

```python
@mqtt.on_message()
def handle_mqtt_message(client, userdata, message):
    global data
    topic = message.topic
    try:
        payload = message.payload.decode()
        with lock:

            if topic == "greenhouse/temperature":
                data["temperature"] = float(payload)
            elif topic == "greenhouse/humidity":
                data["humidity"] = float(payload)
            elif topic == "greenhouse/motion":
                data["motion"] = payload
            elif topic == "greenhouse/distance":
                data["distance"] = float(payload)
            elif topic == "greenhouse/relay":
                data["relay"] = payload


            socketio.emit('update_data', data)
            print(f"Updated data from topic '{topic}': {data}")
```

3. Settings Configuration

The system provides users with the ability to define acceptable ranges for temperature and humidity through a dedicated API. These settings are stored in the global data dictionary and remain accessible throughout the user session. This feature allows for customized control and monitoring of greenhouse conditions, ensuring the system adapts to specific user-defined environmental requirements.

```python
@app.route('/api/settings', methods=['POST'])
def save_settings():
    global data
    try:

        min_temp = request.json.get('minTemp')
        max_temp = request.json.get('maxTemp')
        min_humid = request.json.get('minHumid')
        max_humid = request.json.get('maxHumid')


        if min_temp is not None:
            data['minTemp'] = min_temp
        if max_temp is not None:
            data['maxTemp'] = max_temp
        if min_humid is not None:
            data['minHumid'] = min_humid
        if max_humid is not None:
            data['maxHumid'] = max_humid


        return jsonify({
            'message': 'Settings saved successfully',
            'settings': data
```

Conclusion

In conclusion, the Greenhouse Monitoring and Control Web Application offers a comprehensive solution for real-time tracking and management of greenhouse environmental parameters. By integrating IoT technology, MQTT communication, and a user-friendly interface, the application enables seamless monitoring of key factors like temperature, humidity, motion, and distance. Real-time updates and remote control capabilities ensure a responsive and interactive user experience. Additionally, the ability to export historical data in CSV format further enhances the system's utility, providing users with valuable insights for analysis and record-keeping.

Limitations

Despite its many strengths, the system does have some limitations. The current reliance on in-memory data storage for user credentials and settings is not scalable in the long term. To address

this, transitioning to a persistent database would be a necessary step for improving data storage reliability. Additionally, the system's performance depends on the proper functioning of the IoT sensors and MQTT communication infrastructure. Any disruption in these components could lead to failures in data collection or real-time updates, highlighting the need for continuous monitoring and maintenance of these devices.

Future Improvements

While the application currently meets the requirements for greenhouse monitoring, there are several areas for future enhancement. A major opportunity for improvement lies in the integration of a persistent database such as SQLite or PostgreSQL to replace the current in-memory storage. This would provide more robust data retention and scalability for a growing user base. Additionally, incorporating advanced data analytics tools could allow users to gain deeper insights from historical trends and predictive analysis, potentially enhancing decision-making. A mobile app version of the system could expand the application's accessibility, enabling users to monitor and control the greenhouse environment while on the go. Furthermore, expanding the automation features to include more complex logic, such as energy optimization or weather forecasting, would add significant value to the system, creating a more efficient greenhouse environment.

Implemented by:

Sami sulaiman & Mary Hannoush