# Raspberry Pi Weather Logger System

A full-stack weather logging and visualization system using a Raspberry Pi and DHT11 sensor

## Table of Contents

# 1. Introduction

The Raspberry Pi Weather Logger is a full-stack IoT project designed to collect, store, and visualize environmental data. Using a DHT11 sensor connected to a Raspberry Pi, the system records temperature and humidity readings every hour, stores them in a MariaDB database, and provides a web interface for users to view and export the data. This project demonstrates the integration of hardware, backend, and frontend technologies for real-world data monitoring.

# 2. System Setup

## Hardware

- **Raspberry Pi 3 Model B Plus Rev 1.3**
- **DHT11 sensor**
- **Wiring:**
  - VCC to 3.3V
  - GND to GND
  - DATA to GPIO4 (board.D4)

## Software

- **OS:** Debian GNU/Linux 12 (bookworm)
- **Packages:**
  - Apache2, Apache/2.4.62 (Debian), Apache/2.4.62 (Debian), PHP 8.2.28, Ver 15.1 Distrib 10.11.11-MariaDB
  - Python 3.11.2, adafruit-circuitpython-dht, mysql-connector-python

## Database

- **Database:** weather_data
- **Table**: sensor_readings
- **Fields:**
  - id (INT, AUTO_INCREMENT, PRIMARY KEY)
  - datetime (TIMESTAMP)
  - temperature (DECIMAL(5,2))
  - humidity (DECIMAL(5,2))
- **Schema Import:** mysql -u root -p weather_data < database/weather_data.sql

## Installation Steps

sudo apt-get install apache2 mariadb-server php php-mysql python3-pip
sudo pip3 install adafruit-circuitpython-dht mysql-connector-python

# 3. Usage Instructions

### Start Data Collection

Set up a cron job to run scripts/collect_data.php every hour:

0 * * * * /usr/bin/php /var/www/html/raspberry-pi-weather-logger/scripts/collect_data.php

### Access Web Interface

Open your browser and go to: http://10.0.0.243/raspberry-pi-weather-logger/

### Export Data

Download a CSV file by visiting: http://10.0.0.243/phpmyadmin/

### View Logs

Check weather_data_collection.log for collection status and errors.

# 4. Source Code Overview

- scripts/collect_data.php: Collects sensor data and stores it in the database.

```php
<?php
// Database configuration
$db_host = 'localhost';
$db_user = 'root';
$db_pass = 'UyenMai2025';
$db_name = 'weather_data';

// Connect to database
$conn = new mysqli($db_host, $db_user, $db_pass, $db_name);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Set timezone to UTC
$conn->query("SET time_zone = '+00:00'");

// Function to read DHT sensor data
function readDHT($pin) {
    // Using Adafruit DHT library
    $result = shell_exec("python3 -c 'import adafruit_dht; import board; dht = adafruit_dht.DHT11(board.D4); print(f\"{dht.temperature} {dht.humidity}\")'");
```

```
    $data = explode(" ", trim($result));

    if (count($data) == 2) {
        return array(
            'temperature' => floatval($data[0]),
            'humidity' => floatval($data[1])
        );
    } else {
        // If reading fails, return null values
        return array(
            'temperature' => null,
            'humidity' => null
        );
    }
}

// Read sensor data
$sensor_data = readDHT(4); // Using GPIO4 (board.D4)

// Only insert data if we got valid readings
if ($sensor_data['temperature'] !== null && $sensor_data['humidity'] !== null)
{
    // Prepare and execute the SQL statement
    $stmt = $conn->prepare("INSERT IGNORE INTO sensor_readings (temperature,
humidity) VALUES (?, ?)");
    $stmt->bind_param("dd", $sensor_data['temperature'],
$sensor_data['humidity']);

    if ($stmt->execute()) {
        echo "Data collected and stored successfully\n";
    } else {
        echo "Error storing data: " . $stmt->error . "\n";
    }

    $stmt->close();
} else {
    echo "Error reading sensor data\n";
}

$conn->close();
?>
```

- scripts/query_data.php: Fetches data for the web interface, supports daily and hourly queries.

```
<?php
header('Content-Type: application/json');

// Database configuration
$db_host = 'localhost';
$db_user = 'root';
$db_pass = 'UyenMai2025';
```

```php
$db_name = 'weather_data';

// Connect to database
$conn = new mysqli($db_host, $db_user, $db_pass, $db_name);

if ($conn->connect_error) {
    die(json_encode(['error' => 'Connection failed: ' . $conn->connect_error]));
}

// Set timezone to UTC
$conn->query("SET time_zone = '+00:00'");

// Get form data
$startDate = $_POST['startDate'] . ' 00:00:00';
$endDate = $_POST['endDate'] . ' 23:59:59';
$selectedData = $_POST['dataType'];

// Prepare response array
$response = [
    'daily' => [],
    'hourly' => []
];

// Get daily averages
$dailyQuery = "SELECT
    DATE(datetime) as date,
    AVG(temperature) as temperature,
    AVG(humidity) as humidity
FROM sensor_readings
WHERE datetime BETWEEN ? AND ?
GROUP BY DATE(datetime)
ORDER BY date";

$stmt = $conn->prepare($dailyQuery);
$stmt->bind_param("ss", $startDate, $endDate);
$stmt->execute();
$result = $stmt->get_result();

while ($row = $result->fetch_assoc()) {
    $response['daily'][] = [
        'date' => $row['date'],
        'temperature' => round($row['temperature'], 2),
        'humidity' => round($row['humidity'], 2)
    ];
}

// Get hourly values
$hourlyQuery = "SELECT
    datetime,
    temperature,
```

```
    humidity
FROM sensor_readings
WHERE datetime BETWEEN ? AND ?
ORDER BY datetime";

$stmt = $conn->prepare($hourlyQuery);
$stmt->bind_param("ss", $startDate, $endDate);
$stmt->execute();
$result = $stmt->get_result();

while ($row = $result->fetch_assoc()) {
    $response['hourly'][] = [
        'datetime' => $row['datetime'],
        'temperature' => round($row['temperature'], 2),
        'humidity' => round($row['humidity'], 2)
    ];
}

$stmt->close();
$conn->close();

echo json_encode($response);
?>
```

- index.html: Main dashboard for data visualization, includes interactive charts and data selection forms.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Raspberry Pi Weather Logger</title>
    <link rel="stylesheet" href="css/style.css">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/moment"></script>
</head>
<body>
    <div class="container">
        <h1>Raspberry Pi Weather Logger</h1>

        <form id="dataForm">
            <div class="form-group">
                <label for="startDate">Start Date:</label>
                <input type="date" id="startDate" name="startDate" required>

                <label for="endDate">End Date:</label>
                <input type="date" id="endDate" name="endDate" required>
            </div>

            <div class="form-group">
                <h3>Select Data to Display:</h3>
```

```
                <div class="checkbox-group">
                    <label>
                        <input type="checkbox" name="dataType"
value="temp_daily"> Temperature - Daily Average
                    </label>
                    <label>
                        <input type="checkbox" name="dataType"
value="temp_hourly"> Temperature - Hourly Values
                    </label>
                    <label>
                        <input type="checkbox" name="dataType"
value="humidity_daily"> Humidity - Daily Average
                    </label>
                    <label>
                        <input type="checkbox" name="dataType"
value="humidity_hourly"> Humidity - Hourly Values
                    </label>
                </div>
            </div>

            <button type="submit" id="submitBtn" disabled>Generate
Charts</button>
        </form>

        <div class="charts-container">
            <div class="chart-wrapper">
                <canvas id="dailyChart"></canvas>
            </div>
            <div class="chart-wrapper">
                <canvas id="hourlyChart"></canvas>
            </div>
        </div>
    </div>

    <script src="assets/validate.js"></script>
    <script src="assets/charts.js"></script>
</body>
</html>
```

- assets/validate.js: Client-side form validation for user input.

```
document.addEventListener('DOMContentLoaded', function() {
    const form = document.getElementById('dataForm');
    const startDate = document.getElementById('startDate');
    const endDate = document.getElementById('endDate');
    const checkboxes = document.querySelectorAll('input[name="dataType"]');
    const submitBtn = document.getElementById('submitBtn');

    // Set max date to today
    const today = new Date().toISOString().split('T')[0];
    startDate.max = today;
    endDate.max = today;

```

```
    // Set min date to 7 days ago
    const sevenDaysAgo = new Date();
    sevenDaysAgo.setDate(sevenDaysAgo.getDate() - 7);
    startDate.min = sevenDaysAgo.toISOString().split('T')[0];
    endDate.min = sevenDaysAgo.toISOString().split('T')[0];

    function validateForm() {
        // Check if dates are valid
        const start = new Date(startDate.value);
        const end = new Date(endDate.value);
        const dateRangeValid = start <= end &&
                               (end - start) / (1000 * 60 * 60 * 24) <= 7;

        // Check if at least one checkbox is selected
        const checkboxSelected = Array.from(checkboxes).some(cb => cb.checked);

        // Enable/disable submit button
        submitBtn.disabled = !(dateRangeValid && checkboxSelected);
    }

    // Add event listeners
    startDate.addEventListener('change', validateForm);
    endDate.addEventListener('change', validateForm);
    checkboxes.forEach(checkbox => {
        checkbox.addEventListener('change', validateForm);
    });

    // Form submission
    form.addEventListener('submit', function(e) {
        e.preventDefault();

        const formData = new FormData(form);
        const selectedData = Array.from(formData.getAll('dataType'));

        // Call the PHP script to get data
        fetch('scripts/query_data.php', {
            method: 'POST',
            body: formData
        })
        .then(response => response.json())
        .then(data => {
            // Update charts with the received data
            updateCharts(data, selectedData);
        })
        .catch(error => {
            console.error('Error:', error);
            alert('Error fetching data. Please try again.');
        });
    });
});
```

- assets/charts.js: Handles rendering of interactive charts.

```
let dailyChart = null;
let hourlyChart = null;

function updateCharts(data, selectedData) {
    // Destroy existing charts if they exist
    if (dailyChart) dailyChart.destroy();
    if (hourlyChart) hourlyChart.destroy();

    // Prepare data for daily averages
    const dailyData = {
        labels: data.daily.map(d => d.date),
        datasets: []
    };

    // Prepare data for hourly values
    const hourlyData = {
        labels: data.hourly.map(h => h.datetime),
        datasets: []
    };

    // Add temperature daily average if selected
    if (selectedData.includes('temp_daily')) {
        dailyData.datasets.push({
            label: 'Temperature (°C) - Daily Average',
            data: data.daily.map(d => d.temperature),
            borderColor: 'rgb(255, 99, 132)',
            backgroundColor: 'rgba(255, 99, 132, 0.2)',
            tension: 0.1
        });
    }

    // Add humidity daily average if selected
    if (selectedData.includes('humidity_daily')) {
        dailyData.datasets.push({
            label: 'Humidity (%) - Daily Average',
            data: data.daily.map(d => d.humidity),
            borderColor: 'rgb(54, 162, 235)',
            backgroundColor: 'rgba(54, 162, 235, 0.2)',
            tension: 0.1
        });
    }

    // Add temperature hourly values if selected
    if (selectedData.includes('temp_hourly')) {
        hourlyData.datasets.push({
            label: 'Temperature (°C) - Hourly',
            data: data.hourly.map(h => h.temperature),
            borderColor: 'rgb(255, 99, 132)',
            backgroundColor: 'rgba(255, 99, 132, 0.2)',
            tension: 0.1
        });
```

```
        }

        // Add humidity hourly values if selected
        if (selectedData.includes('humidity_hourly')) {
            hourlyData.datasets.push({
                label: 'Humidity (%) - Hourly',
                data: data.hourly.map(h => h.humidity),
                borderColor: 'rgb(54, 162, 235)',
                backgroundColor: 'rgba(54, 162, 235, 0.2)',
                tension: 0.1
            });
        }

        // Create daily chart if there are daily datasets
        if (dailyData.datasets.length > 0) {
            dailyChart = new Chart(document.getElementById('dailyChart'), {
                type: 'line',
                data: dailyData,
                options: {
                    responsive: true,
                    plugins: {
                        title: {
                            display: true,
                            text: 'Daily Averages'
                        },
                        tooltip: {
                            mode: 'index',
                            intersect: false
                        }
                    },
                    scales: {
                        y: {
                            beginAtZero: false
                        }
                    }
                }
            });
        }

        // Create hourly chart if there are hourly datasets
        if (hourlyData.datasets.length > 0) {
            hourlyChart = new Chart(document.getElementById('hourlyChart'), {
                type: 'line',
                data: hourlyData,
                options: {
                    responsive: true,
                    plugins: {
                        title: {
                            display: true,
                            text: 'Hourly Values'
                        },
```

```
                    tooltip: {
                        mode: 'index',
                        intersect: false
                    }
                },
                scales: {
                    y: {
                        beginAtZero: false
                    }
                }
            }
        });
    }
}
```

- css/style.css: Provides custom styles for the web interface, ensuring a clean and modern look.

```
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    background-color: #f4f4f4;
    padding: 20px;
}

.container {
    max-width: 1200px;
    margin: 0 auto;
    background-color: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
}

h1 {
    text-align: center;
    color: #333;
    margin-bottom: 30px;
}

.form-group {
    margin-bottom: 20px;
}

label {
    display: block;
```

```
        margin-bottom: 5px;
        color: #555;
    }

    input[type="date"] {
        width: 100%;
        padding: 8px;
        margin-bottom: 10px;
        border: 1px solid #ddd;
        border-radius: 4px;
    }

    .checkbox-group {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
        gap: 10px;
    }

    .checkbox-group label {
        display: flex;
        align-items: center;
        gap: 5px;
    }

    button {
        display: block;
        width: 100%;
        padding: 10px;
        background-color: #4CAF50;
        color: white;
        border: none;
        border-radius: 4px;
        cursor: pointer;
        font-size: 16px;
    }

    button:disabled {
        background-color: #cccccc;
        cursor: not-allowed;
    }

    .charts-container {
        margin-top: 30px;
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(500px, 1fr));
        gap: 20px;
    }

    .chart-wrapper {
        background-color: white;
        padding: 15px;
```

```
    border-radius: 8px;
    box-shadow: 0 0 5px rgba(0,0,0,0.1);
}

@media (max-width: 768px) {
    .charts-container {
        grid-template-columns: 1fr;
    }

    .checkbox-group {
        grid-template-columns: 1fr;
    }
}
```

- database/weather_data.sql: Database schema and table creation script.

```
-- Create the weather_data database
CREATE DATABASE IF NOT EXISTS weather_data;
USE weather_data;

-- Create the sensor_readings table
CREATE TABLE IF NOT EXISTS sensor_readings (
    id INT AUTO_INCREMENT PRIMARY KEY,
    datetime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    temperature DECIMAL(5,2) NOT NULL,
    humidity DECIMAL(5,2) NOT NULL,
    UNIQUE KEY unique_reading (datetime)
);

-- Create indexes for better query performance
CREATE INDEX idx_datetime ON sensor_readings(datetime);
```

- cronjobs/data_collection_cron.sh: Shell script to automate data collection.

```
#!/bin/bash

# Log file for the cron job
LOG_FILE="/var/log/weather_data_collection.log"

# Get the current timestamp
TIMESTAMP=$(date +"%Y-%m-%d %H:%M:%S")

# Run the PHP script and log the output
echo "[$TIMESTAMP] Starting data collection..." >> $LOG_FILE
php /var/www/html/raspberry-pi-weather-logger/scripts/collect_data.php >>
$LOG_FILE 2>&1
echo "[$TIMESTAMP] Data collection completed." >> $LOG_FILE
```

## Data Flow

1. Sensor data is read by the PHP script and stored in MariaDB.
2. Web interface fetches and displays data using Chart.js.
3. Users can export data as CSV for further analysis.

# 5. Screenshots & Visualizations

Below are screenshots of the Raspberry Pi Weather Logger system in operation:

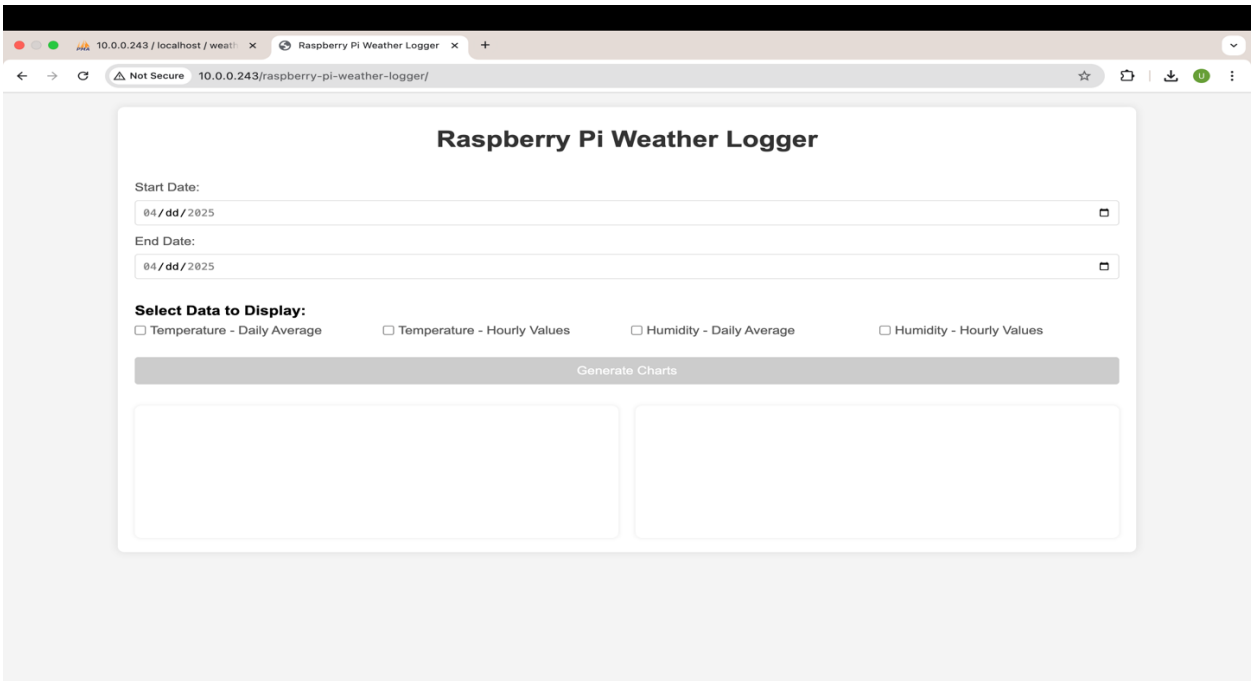
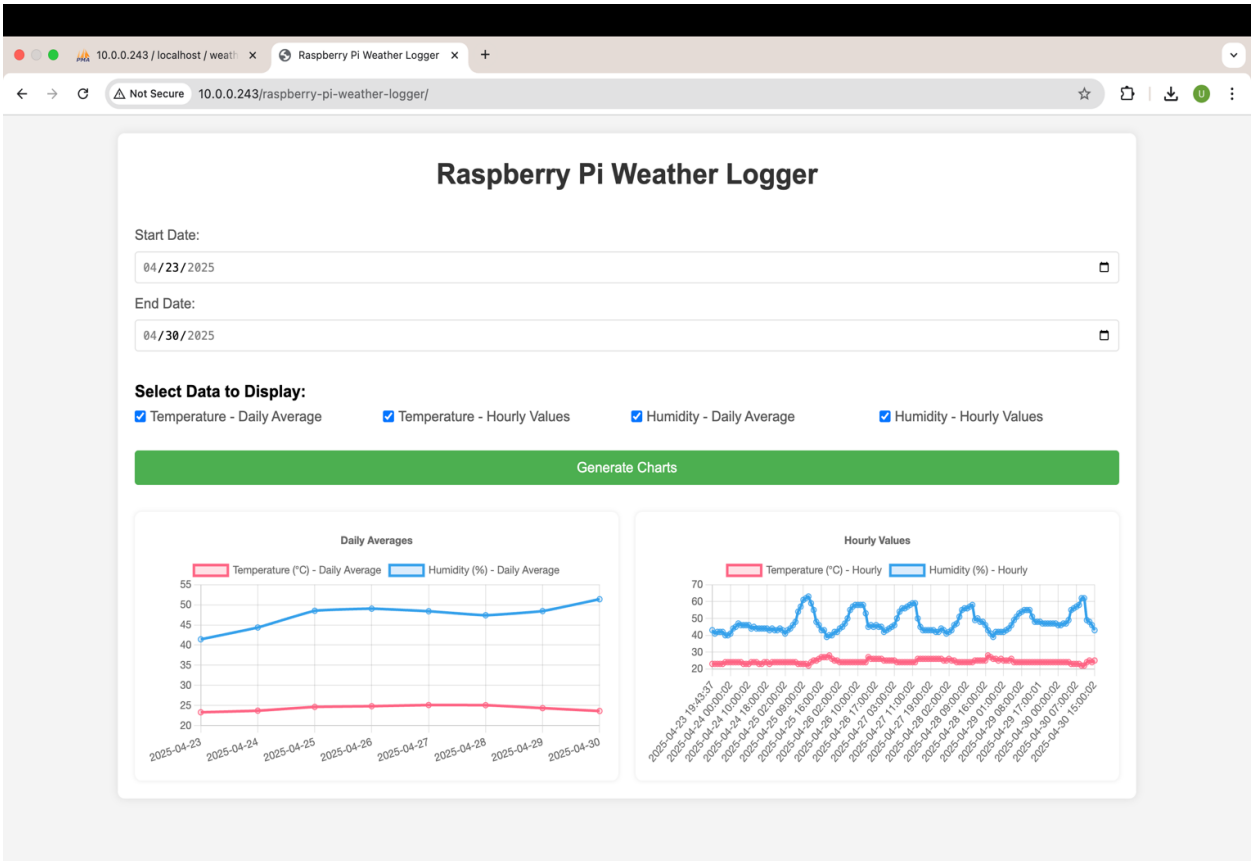



*Figure 1: HTML Form for data selection*



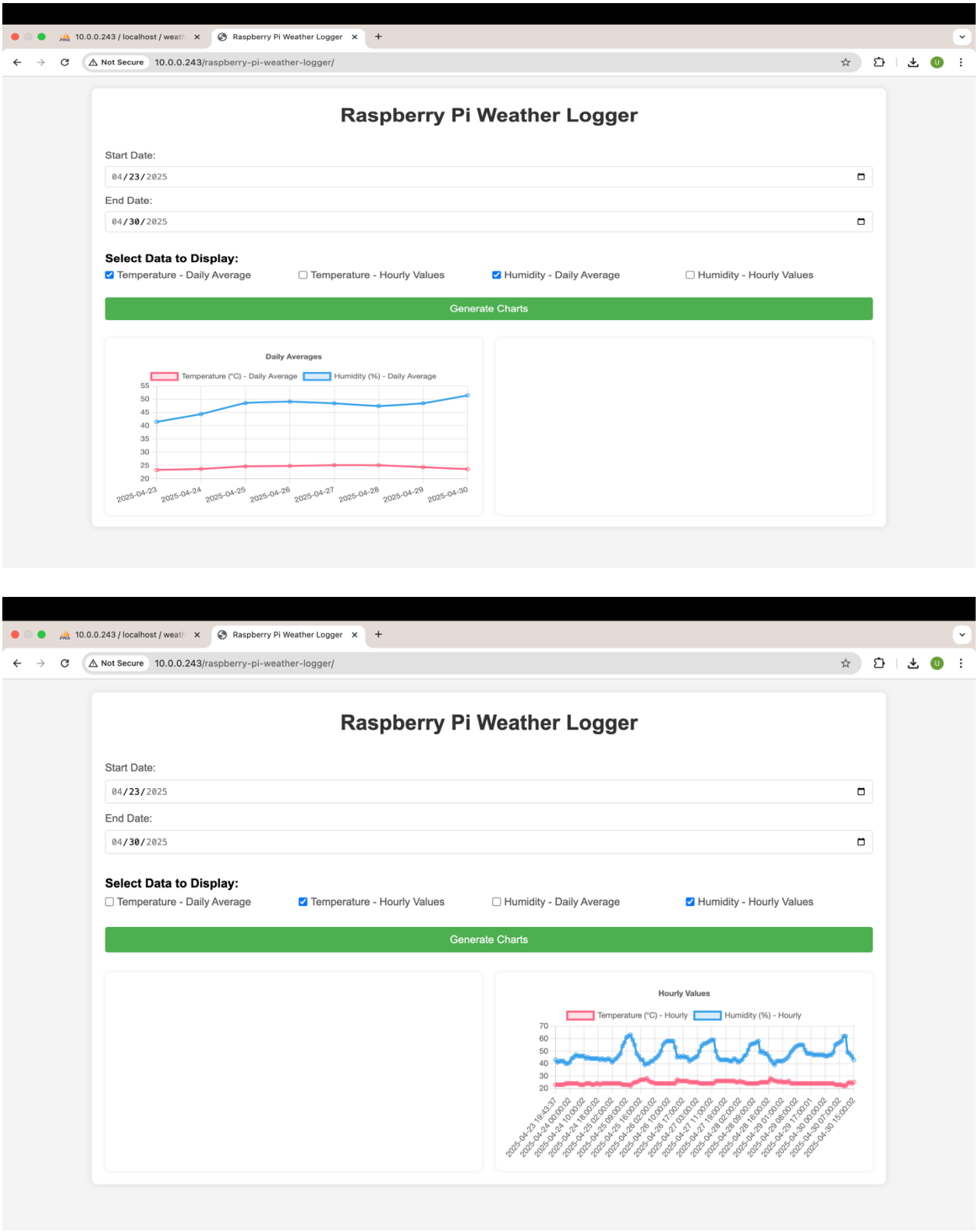



*Figure 2-4: Dashboard with interactive temperature and humidity charts*

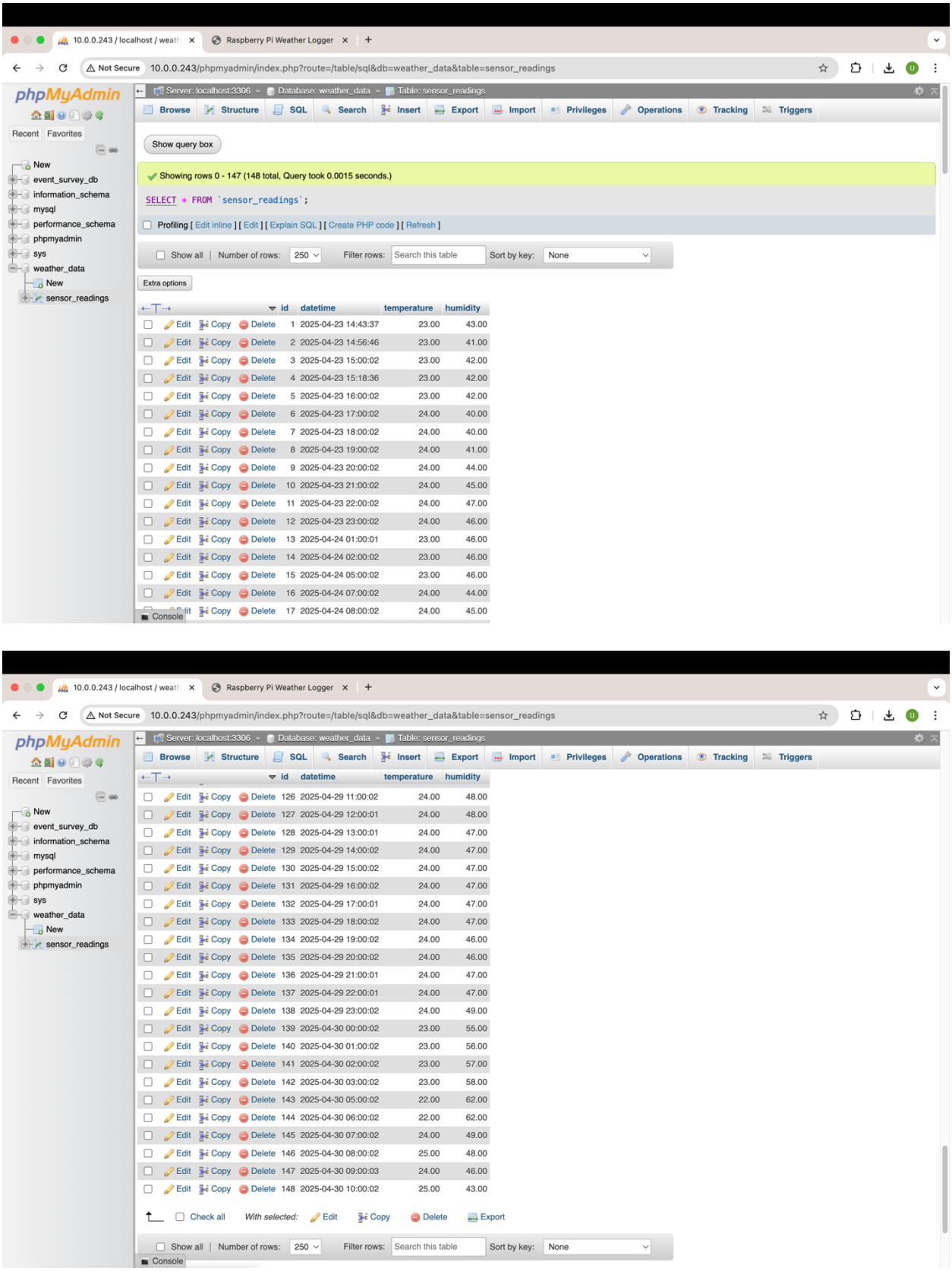






*Figure 5-6: Data Export/Download Page*

# 6. Appendix

## Full code listings

See the HTML, PHP, and JavaScript Code.zip for all source code.

## Additional logs or data

See data_collection_log.txt for sensor data collection.

## Database dump

See weather_data_dump.sql and weather_data_export.csv for schema and data.

## HTML, PHP, and JavaScript Code.zip File Structure

```
HTML, PHP, and JavaScript Code/
├── assets/
│   ├── charts.js
│   └── validate.js
├── cronjobs/
│   └── data_collection_cron.sh
├── css/
│   └── style.css
├── database/
│   ├── weather_data.sql
├── scripts/
│   ├── collect_data.php
│   ├── query_data.php
├── index.html
└── README.md
```