Smart Environment Monitoring System Report

Data Collection and Storage:

The system applies a Raspberry Pi (or an emulator) with Sense HAT to obtain environmental readings of temperature, humidity and pressure. The script sensor_read.py retrieves sensor data through an attempt to scan live measurements but if this action fails it generates random moving sensor values instead. The system saves all collected data points to the local SQLite database named environment.db. The “readings” table in the database comes into existence upon system startup before any data recording takes place. Sensor_read.py executes error-handling mechanisms through try/except structures and handles failed sensor reads while also safely disconnecting database resources upon program termination.

Web Interface:

The web interface includes three key components which are the Flask server named app.py and index.html template and the client-side JavaScript known as app.js. The interface displays:

Current sensor data (timestamp, temperature, humidity, and pressure) in real time.

The warning threshold adjustment feature exists within the application settings panel. The server receives all AJAX requests that transmit changes from users which immediately produce outcomes.

Web users can visualize dynamic updated temperature history through a Chart.js-produced chart that operates without requiring a complete page reload.

The application calculates the estimated time for temperature exceeding its maximum threshold by applying linear regression analysis to historical recorded data then extending the identified trend pattern.

The interface depends on Socket.IO to send real-time updates from the server to the client which results in immediate display of sensor information on the browser.

Warning System:

The system compares sensor readings with thresholds that users set in advance. The system triggers a warning whenever measurement data exceeds the specified thresholds between 0°C to 40°C temperature and 10% to 90% humidity and 970 hPa to 1030 hPa pressure. Users can find temperature warnings displayed on web interface screens and LED matrix alerts when using the Sense HAT that show temperature changes through color variations. The adjustable thresholds can be modified from the web interface and warnings appear instantly.

Risk Analysis:

Several security problems emerged in the most updated programming code. The system faced session tampering risks because it lacked a secure system to control its secret key which signs session cookies and secures form data. Network attackers could forge requests to change state through state-changing endpoints because of the lack of CSRF protection on the threshold update route. The system faced three critical issues stemming from unsecured network data exposure because of unprotected traffic along with potential attack vectors from unhandled input data as well as restrictive policies which could block site resources. The system became exposed to vulnerability because critical database functions did not have proper error management procedures so data incidents could become present with operations failures.

Security Measures Implemented:

These issues required the code to receive updates that implemented secure measures for protection. The session data remains safe because the secret key draws from environment variables while having a secure backup system. The application implements CSRF protection through csrf protect from Flask-WTF to prevent form submission attacks while also excluding JSON endpoints from protection requirements which will receive optimization before production deployment. The deployment of Flask-Talisman serves two functions: it enables HTTPS encryption while adding secure HTTP headers; the CSP security policy is set to None for development purposes to let necessary external resources access the application. The conversion of input values to floats works as a validation process which protects against injection attacks. The code implements try/except blocks along with proper logging to prevent system crashes or data exposure which ensures stability and reliability of the system.

Robust Error Handling:

Error management within the system functions through try/except blocks that protect vital operations including sensor reading and database communications and network interfaces. The Python logging module tracks errors which occur when sensors read incorrectly along with database query failures in these specific areas. The system continues operating after detecting an error and manages it in a controlled manner without failing. The system operates harmlessly through exception blocking while recording diagnostic elements for future problem solving and fix implementation.

System reliability and availability receive significant improvement through this method because single errors cannot affect the overall functionality which is essential for continuous monitoring systems. Through detailed logs the system helps developers conduct easier debugging and maintenance operations by recording the errors with their corresponding timestamps. Through robust error handling the system maintains its functionality when facing sensor or database problems so it continues to gather data and update web interface data in real time.

Data Analysis and Predictive Features:

Identifying Sudden changes and Recognizing Trends:

The most recent code version retrieves and processes temperature data from the SQLite database to perform analysis. The system collects timestamps together with temperature values before turning timestamps into numerical data that supports linear regression evaluation. The code determines quick variations by analyzing a series of measurements where the actual difference between pairs exceeds particular conditions. The system employs NumPy's linear regression functions to calculate slope and intercepts that demonstrate steady trends in the data using np.polyfit. When the slope value is positive the temperature shows an upward direction but negative slope values show downward temperature trends. The obtained trend line becomes visible along with the actual temperature data through Chart.js graphical representation which shows long-term trends together with short-term anomalies.

Predicting Future Threshold Exceedance:

After conducting regression analysis the system provides predictions regarding the conditions in which temperature will breach specified thresholds. After deriving slope (m) and intercept (b) values through regression the system computes the future timestamp when the temperature will exceed the maximum threshold based on the rearranged linear equation. The system makes this projection possible by extending the trend line until it calculates an end point which is converted to a traditional time format. Users can access web interface visualizations containing both the predicted time point and historical trends with data points which provide actionable knowledge about potential temperature critical thresholds. This system monitoring feature allows users to make proactive decisions because it enables timely interventions which lead to improved system reliability.

Setup and Configuration Instructions:

Raspberry Pi OS or its equivalent distribution and an identical emulator serve as necessary prerequisites to establish 24001041 file. Start creating a Python virtual environment through the command python3 -m venv myenv executed in your project base folder then use source myenv/bin/activate to start it. Install every necessary Python package by running pip with an additional parameter --break-system-packages when required. Your project must follow this file organization: "24001042" as the root folder with app.py, sensor_read.py, analysis.py optional and environment.db as its automatically created SQLite database and the templates and static directories containing index.html and subdirectories style.css and app.js respectively. To function, the system requires running the sensor logging script (sensor_read.py) first in one terminal to fill the database followed by starting the Flask server (app.py) in a different terminal. After opening your web browser type http://<your_pi_ip>:5000 to access the dashboard.

User Manual for the Web Interface and Additional Applications:

24001042 uses a web interface that provides an extensive monitoring dashboard functionality. The page shows real-time environmental sensor updates through Socket.IO as the dashboard presents timestamps and settings of temperature, humidity and pressure data. The system presents a threshold settings interface for users to set valid ranges for the sensor measurements. Users can submit alterations to the threshold settings panel by using an AJAX-form that automatically drills the changes into the system warning indicators upon server processing. Users can access temperature data history through a Chart.js-based line chart displaying past readings with a linear regression trend line. The temperature prediction appears beneath the graph where the chart updates every ten seconds. The system’s interface provides easy monitoring together with straightforward configuration capabilities through its responsive design.

Troubleshooting Guide for Common Issues:

Check the operation of sensor_read.py script and environment.db file reading accumulation by examining results through sqlite3 commands like sqlite3 environment.db "SELECT * FROM readings;" in terminal sessions. Check the server logs for errors when the Flask server (app.py) fails to run properly along with verifying that all interfaces operate without errors. The Developer Console function of your browser accessible through F12 enables you to find JavaScript problems and network errors which block real-time updates. Inspections of the historical temperature graph and trend line should verify the availability of at least two data points by accessing the /history endpoint manually. Complete threshold updates require proper numerical formatting of input data while examining server logs for validation error messages. The system stability can be preserved through regular script and server restarts that address brief operational problems.